الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

جامعة محمد خيضر بسكرة
كلية العلوم الدقيقة وعلوم الطبيعة والحياة
قسم الإعلام الآلي

Université Mohamed Khider Biskra
Faculté des sciences exactes et sciences des sciences de la nature et de la vie
Département d'informatique

الرقم: 174/ق إ آ/2024 بسكرة في 2024/05/13

مستخرج من محضر اجتماع اللجنة العلمية رقم 2024/02

المنعقد يوم 2024/05/13 على الساعة التاسعة والنصف صباحا

طبقا لمحضر إجتماع اللجنة العلمية للقسم رقم 2024/02، وافقت اللجنة على مطبوعة بيداغوجية باللغة الانجليزية المقترحة من طرف الأستاذ **تبرماسين أحمد** تحت عنوان :

"Machine Learning" والموجه لطلبة **لطلبة السنة الأولى ماستر إعلام آلي تخصص شبكات وتقنيات الإعلام والإتصال.**

رئيس اللجنة العلمية

الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي و البحث العلمي
Ministry Of Higher Education And Scientific Research

Mohamed Khider University - Biskra
Faculty Of Exact Sciences Natural and Life Sciences
Department of Computer Science

جامعة محمد خيضر بسكرة
كلية العلوم الدقيقة وعلوم الطبيعة و الحياة
قسم: الاعلام الآلي

UNIVERSITY COURSE HANDOUT

Specialization: M2-RTIC

Machine Learning

Courses and Applications

Prepared and Presented by:

Dr. Ahmed TIBERMACINE

Academic Year 2023/2024

Table of Contents

Chapter 3: Deep Learning

Chapter 4: Bayesian Network

Chapter 5: Hidden Markov model

Chapter 6: Applications of ML in communication networks

Chapter 1:

Introduction to Machine Learning

1. Introduction

From translation apps to autonomous vehicles, Machine Learning is the driving force behind these innovations. It's all about training algorithms to make predictions from data. In this chapter, we'll explore different types of machine learning and key terminology.

2. Artificial Intelligence and Machine Learning?

Artificial Intelligence (AI) and Machine Learning (ML) are closely related fields in computer science that focus on developing systems and algorithms to enable computers to perform tasks that typically require human intelligence. While they are related, they have distinct characteristics and purposes:

2.1. Artificial Intelligence (AI)

AI is a computer science field that aims to make machines smart like humans, allowing them to do things independently by learning from experience, understanding language, and solving problems. The main goal of AI is to create machines that can think and make intelligent decisions based on what they know [1, 2].

2.2. Machine Learning (ML)

ML is a part of AI that helps computers get smarter by learning from data and experience instead of needing explicit instructions. The main aim of ML is to make algorithms and models that can understand patterns in data, so they can make predictions or choices based on new information they haven't seen before. ML algorithms are good at figuring out things from data and can apply this knowledge to new situations [3].

2.3. Differences between AI and ML

Here are some key differences between AI and ML:

Scope: AI is a broader field encompassing various techniques, including rule-based systems, expert systems, natural language processing (NLP), computer vision, robotics, and more. ML is a subset of AI that specifically deals with learning patterns from data.

Learning: In AI, techniques can be rule-based, heuristic, or rely on pre-programmed knowledge, whereas ML focuses on learning patterns from data.

Training: ML models require training on labeled data to improve their performance. AI systems may not always involve explicit training but may rely on predetermined rules and knowledge.

Flexibility: AI systems can be rule-based and may not adapt to new data without manual intervention. ML models can adapt and improve their performance with more data.

In practice, AI and ML often overlap, with ML techniques being a crucial component of many AI systems. AI systems can incorporate various ML algorithms to enhance their decision-making and adaptability. As technology advances, the boundaries between AI and ML continue to blur, and both fields play a significant role in shaping the future of automation and intelligent systems.

3. Types of Machine Learning

Machine learning algorithms are often divided into three general categories (though other classification schemes are also used): supervised learning, unsupervised learning, and reinforcement learning.

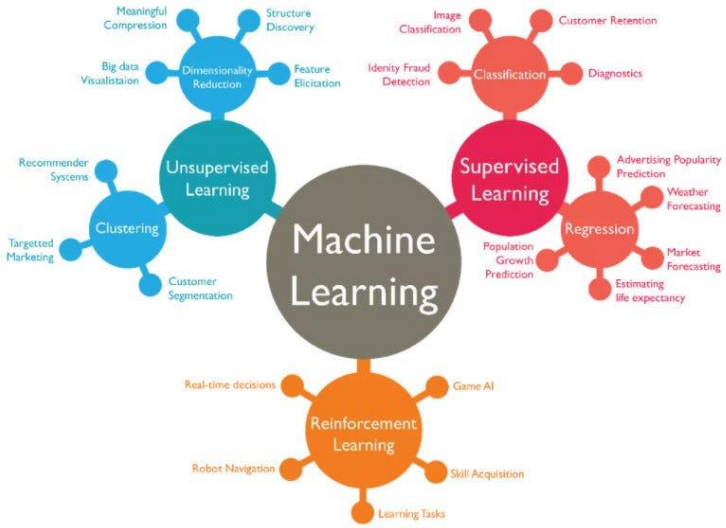


Figure 1: types of machine learning

3.1. Supervised Learning

Supervised Learning is one of the fundamental paradigms in Machine Learning (ML). It is a type of machine learning where an algorithm learns from a labeled dataset, which means that each example in the training data consists of an input (or feature) and the corresponding correct output (or label). The goal of supervised learning is to learn a mapping from inputs to outputs, so that the algorithm can make accurate predictions on new, unseen data.

There are two primary tasks in supervised learning:

3.1.1. Regression

Regression is a type of supervised learning where the target variable is continuous. It's used when you want to predict numerical values. In regression, the algorithm learns a function that maps input features to a continuous output. Key points about regression include:

Examples: Predicting house prices, stock prices, or temperature based on historical data.

Algorithms: Linear regression, polynomial regression, support vector regression, and neural networks.

Evaluation: Common evaluation metrics include mean squared error (MSE) and R-squared (R^2).

3.1.2. Classification

Classification, also a type of supervised learning, is used when the target variable is categorical, falling into predefined classes or categories. In classification, the algorithm assigns data points to discrete classes based on their features. There are two common subtypes:

a. Binary Classification

Binary classification is a specific type of classification where the target variable has only two possible classes. Examples include: Spam email detection (classifying emails as spam or not spam), Medical diagnosis (classifying patients as having a disease or not). In binary classification, the model aims to distinguish between two classes.

b. Multiple Classification

Multiple classification, often referred to as multiclass classification, deals with scenarios where the target variable has more than two classes. Examples include: Identifying the species of flowers (e.g., classifying flowers as roses, daisies, or tulips). In multiple classification, the model assigns data points to one of several possible classes.

3.1.3. Supervised Learning Algorithms

There are various algorithms used in supervised learning, including:

- Linear Regression: Used for regression tasks where the output is a continuous value.
- Logistic Regression: Used for binary classification tasks.
- Decision Trees: Used for both classification and regression tasks.
- Support Vector Machines (SVM): Used for classification and regression.

- Neural Networks: Deep learning models that can handle complex tasks, including image recognition and natural language processing.

3.2. Unsupervised Learning

Unsupervised learning is a category of machine learning where the algorithm learns patterns and structures in data without explicit supervision or labeled output. In other words, it doesn't rely on predefined labels or target values to make predictions or decisions. Instead, unsupervised learning algorithms seek to uncover hidden patterns, group similar data points, or reduce the dimensionality of data.

There are two primary types of unsupervised learning:

Clustering: Clustering algorithms aim to group similar data points together into clusters or segments based on their inherent similarities. The goal is to discover natural groupings in the data. Common clustering algorithms include K-Means, Hierarchical Clustering, and DBSCAN.

Dimensionality Reduction: Dimensionality reduction techniques aim to reduce the number of features or variables in a dataset while preserving its essential characteristics. This is particularly useful when working with high-dimensional data or when trying to visualize data in lower dimensions. Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) are popular dimensionality reduction methods.

3.3. Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning paradigm where an agent learns to make sequential decisions by interacting with an environment to achieve a specific goal. Unlike supervised learning, where the model is trained on labeled data, and unsupervised learning, where the model identifies patterns in unlabeled data, RL is about learning through trial and error. The agent learns to take actions in an environment to maximize a cumulative reward signal.

Here are some key components of reinforcement learning:

Agent: The learner or decision-maker that interacts with the environment. It takes actions based on a policy.

Environment: The external system with which the agent interacts. It provides feedback to the agent in the form of rewards and new states.

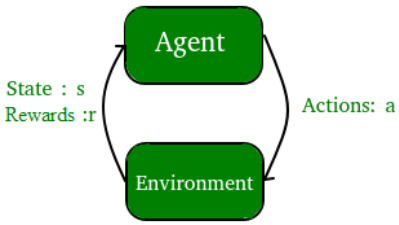
State: A representation of the current situation or configuration of the environment. The state provides information to the agent about its surroundings.

Action: The set of possible choices or decisions that the agent can make in each state. Actions lead to transitions to new states.

Policy: A strategy or mapping that defines the agent's behavior. It determines which actions the agent should take in each state.

Reward: A numerical signal provided by the environment after each action taken by the agent. The reward indicates the immediate benefit or desirability of the action.

Value Function: A function that estimates the expected cumulative reward an agent can achieve from a given state or state-action pair. It helps the agent evaluate the desirability of states or actions.



The goal of reinforcement learning is for the agent to find an optimal policy that maximizes the expected cumulative reward over time. This often involves a trade-off between immediate rewards and long-term goals.

Popular algorithms in reinforcement learning include Q-learning, Deep Q-Networks (DQN), Policy Gradient methods, and more recently, advances in deep reinforcement learning, which combines deep neural networks with RL techniques to handle complex tasks and high-dimensional state spaces.

4. Terminologies of Machine Learning

Here are the key characteristics and concepts associated with ML:

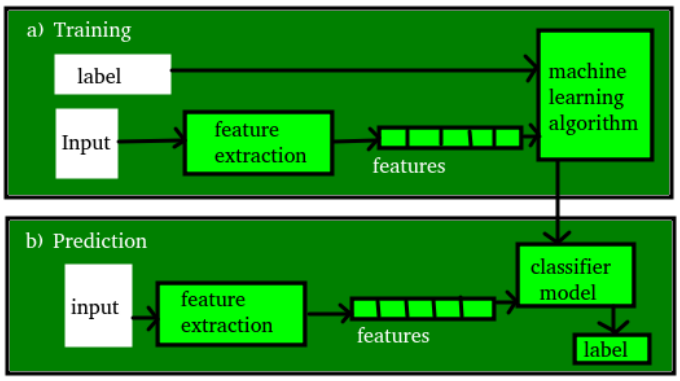
- **Model** A model is a **specific representation** learned from data by applying some machine learning algorithm. A model is also called a **hypothesis**.

- **Feature:** is an individual measurable property of our data. A set of numeric features can be conveniently described by a **feature vector**. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, **etc.**

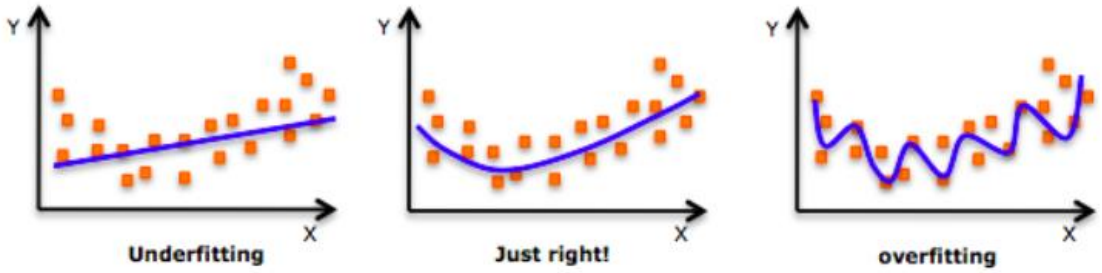
Note: Choosing informative, discriminating and independent features is a crucial step for effective algorithms. We generally employ a **feature extractor** to extract the relevant features from the raw data.

- **Target (Label):** is the value to be predicted by our model. For the fruit example discussed in the features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- **Training Process:** during the training phase, the algorithm learns from the labeled data. It tries to find patterns, relationships, or a mapping function that connects the input data to the correct output. This is done by adjusting the model's parameters iteratively to minimize a predefined loss or error function.
- **Evaluation:** after training, the model's performance is evaluated on a separate dataset called the validation or test set. This helps assess how well the model generalizes to new, unseen data. Common evaluation metrics include accuracy, precision, recall, F1 score, and mean squared error, depending on the type of task (classification or regression….).
- **Prediction** Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output (label). But make sure if the machine performs well on unseen data, then only we can say the machine performs well.

The figure shown below clears the above concepts:



- **Overfitting and Underfitting**: Overfitting occurs when a model learns the training data too well and performs poorly on new data. Underfitting occurs when a model is too simple to capture the underlying patterns in the data



5. Data in Machine Learning

Data is a crucial component in the field of Machine Learning. It refers to the set of observations or measurements that can be used to train a machine-learning model. Data can be any unprocessed fact, value, text, sound, or picture that is not being interpreted and analyzed. The quality and quantity of data available for training and testing play a significant role in determining the performance of a machine-learning model.

Machine learning algorithms use data to learn patterns and relationships between input variables and target outputs, which can then be used for prediction or classification tasks.

5.1. How do we split data in Machine Learning?

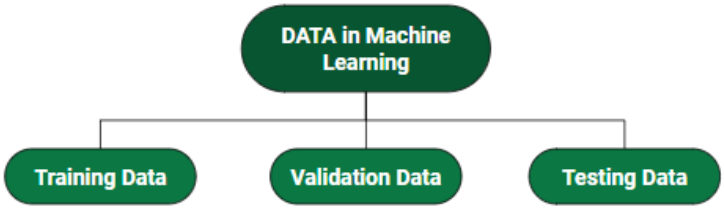
Data can be divided into training, validation and testing sets:

Training Set: Used to train the model (70-80% of data).

Validation Set: Used for hyperparameter tuning and model selection (10-15% of data).

Testing Set: Used to assess the model's performance on unseen data (10-15% of data).

These splits help ensure that the model generalizes well to new, unseen data and



5.2. Different Forms of Data

Data can be categorized into four main types based on its structure and nature: structured, unstructured, quantitative, and qualitative data. Here's an overview of each type:

- **Structured Data** is highly organized and typically presented in tables with rows and columns, making it easy to analyze and query. This type of data is commonly found in databases, spreadsheets, and transaction records. Structured data is well-suited for numerical analysis, reporting, and structured queries, making it ideal for applications involving well-defined data structures.
- **Unstructured Data**, on the other hand, lacks a specific structure and format. It can come in various forms like text, audio, images, or video and doesn't conform to a fixed schema. Examples include text documents, social media posts, and multimedia content. Analyzing unstructured data often requires specialized techniques like natural language processing (NLP) and computer vision to extract meaning and insights from the raw information.
- **Quantitative Data** consists of numerical values that represent quantities or measurements. This type of data is numeric, measurable, and often subject to mathematical operations. Examples include temperature readings, stock prices, age, and test scores. Quantitative data is widely used for statistical analysis, hypothesis testing, regression, and data visualization, making it crucial for quantitative research.
- **Qualitative Data**, on the other hand, is non-numeric and descriptive, representing qualities, characteristics, or categories. It is often expressed as text or labels and includes data like gender, product categories, customer feedback, or survey responses with open-ended questions. Qualitative data analysis involves methods such as categorization, coding, sentiment analysis, and thematic analysis, often used to gain insights into human behavior and preferences.

In real-world scenarios, datasets may contain a mix of these data types, requiring data scientists and analysts to use appropriate techniques and tools for each type to extract valuable insights and make informed decisions.

6. Machine Learning Workflow: From Data to Deployment

Using a machine learning model typically involves several steps, from data preparation to model evaluation and deployment. Here's a high-level overview of the common steps involved in using a machine learning model:

1. **Define the Problem:**

Clearly define the problem you want to solve with machine learning. Understand the objectives and goals of your project.

2. **Gather and Prepare Data:**

Collect and organize the data you will use to train and evaluate your model. This may involve data cleaning, preprocessing, and feature engineering to make the data suitable for modeling.

3. **Split the Data:**

Divide your dataset into training, validation, and test sets. The training set is used to train the model, the validation set is used for hyperparameter tuning, and the test set is used to evaluate the model's performance.

4. **Choose a Machine Learning Algorithm:**

Select a machine learning algorithm or model architecture that is appropriate for your problem. The choice of algorithm depends on the nature of the data and the problem type (classification, regression, clustering, etc.).

5. **Feature Selection/Extraction:**

If necessary, select relevant features or perform feature extraction to reduce dimensionality and improve model performance.

6. **Model Training:**

Train the selected machine learning model using the training data. This involves optimizing the model's parameters to minimize the prediction error.

7. **Hyperparameter Tuning:**

Fine-tune the hyperparameters of your model using the validation set to optimize its performance. This may involve techniques like grid search or random search.

8. **Model Evaluation:**

Assess the model's performance using evaluation metrics appropriate for your problem (e.g., accuracy, F1-score, Mean Absolute Error, etc.). Use the test set for this evaluation to get an unbiased estimate of the model's performance.

9. Iterate and Refine:

If the model's performance is not satisfactory, iterate through the process, making changes such as selecting different algorithms, adjusting hyperparameters, or gathering more data.

10. Model Deployment:

Once you are satisfied with your model's performance, deploy it in a production environment where it can make predictions on new, unseen data. This may involve integrating the model into a web application, mobile app, or other systems.

11. Monitoring and Maintenance:

Continuously monitor the deployed model's performance in real-world scenarios. Update the model as needed to maintain its accuracy and effectiveness.

7. Applications of Machine Learning

Machine learning has a wide range of applications across various industries, including but not limited to:

- Healthcare: Predicting disease outcomes, diagnosing medical conditions, and drug discovery.
- Finance: Credit scoring, fraud detection, and algorithmic trading.
- Natural Language Processing (NLP): Language translation, sentiment analysis, and chatbots.
- Computer Vision: Image and video recognition, object detection, and facial recognition.
- Recommendation Systems: Personalized product recommendations in e-commerce and content recommendations in streaming services.
- Autonomous Vehicles: Self-driving cars and drones that use machine learning for navigation and decision-making.

8. Conclusion

This introductory chapter has provided an overview of machine learning, its types, applications, and key concepts. In the following chapters, we will delve deeper into various techniques of machine learning.

Machine learning is a rapidly evolving field, and staying updated with the latest developments and techniques is essential for success. As we progress through this course, you will gain the knowledge and skills needed to apply machine learning to real-world problems.

Chapter 2:
Artificial Neural Networks

1. Introduction

This chapter provides a comprehensive introduction to neural networks, starting with the foundational perceptron and its limitations. We'll also explore the necessity of multi-layer perceptrons (MLPs) for solving complex tasks, equipping you with essential knowledge for understanding neural networks.

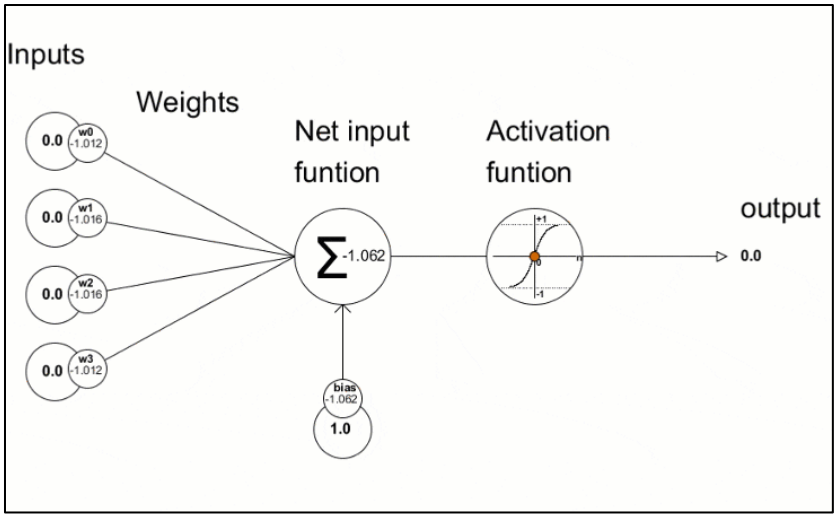
2. Perceptron

2.1. The Building Blocks

2.1.1. Structure of a Perceptron

A perceptron is a simplified model of a biological neuron, inspired by the way neurons in the human brain work. At its core, a perceptron takes multiple inputs, applies weights to these inputs, sums them up, and passes the result through an activation function to produce an output[4, 5]. The structure of a perceptron can be mathematically defined as follows:

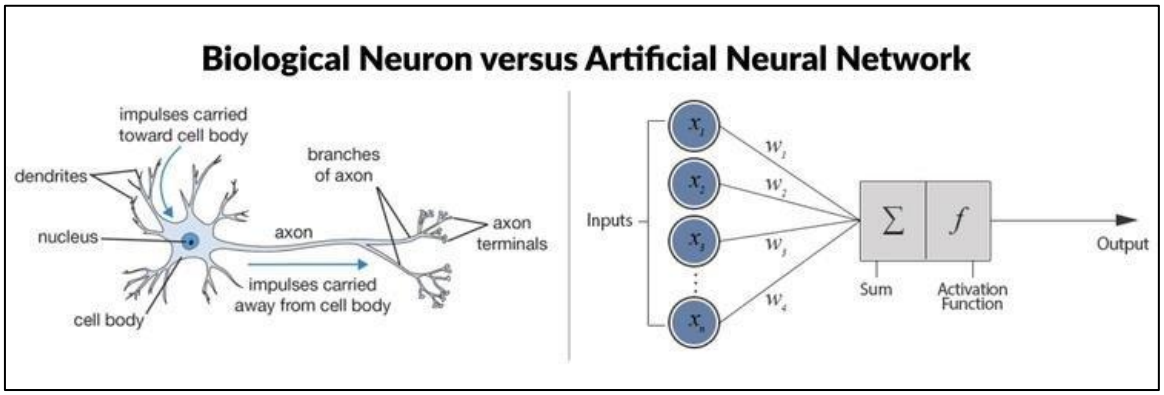
- Inputs: x1,x2,x3,…,xn
- Weights: w1,w2,w3,…,wn
- Bias: b
- Activation function: f(z), where z is the weighted sum of inputs and bias:
- $z = \sum_{i=1}^{n}(wi \cdot xi) + b$
- Output: y=f(z)



2.1.2. Biological Inspiration

The concept of perceptron was inspired by the way biological neurons work. In a biological neuron, dendrites receive electrical signals (analogous to inputs), synapses assign weights to these signals, and the cell body (soma) computes a sum of these weighted inputs. When the sum reaches a certain threshold, the neuron fires, transmitting an electrical signal down its axon to other neurons.

Perceptron mimic this process by summing weighted inputs and applying an activation function to determine whether the perceptron "fires" (produces an output of 1) or not (produces an output of 0).



2.2. Training a Perceptron

2.2.1. Learning Algorithm: The Perceptron Rule

The training process for a perceptron involves finding the optimal weights and bias that allow it to correctly classify input data. The Perceptron Learning Rule is a simple algorithm used for training perceptrons for binary classification tasks. The key idea is to update the weights and bias iteratively until the perceptron makes no more mistakes on the training data.

I. Initialize weights w_1, w_2,…,w_n and bias b with small random values.

II. For each training example (x_1, x_2,…, x_n) with known label y_{true}:

- Compute the weighted sum z as:

$$z = \sum_{i=1}^{n} (wi \cdot xi) + b$$

- Apply the activation function to z to get the predicted output y_{pred} = f(z).
- Update the weights and bias:

- $w_i = w_i + \text{learning rate} \cdot (y_{true} - y_{pred}) \cdot xi$
- $b = b + \text{learning rate} \cdot (y_{true} - y_{pred})$

III. Repeat step 2 for a fixed number of epochs or until the perceptron converges (makes no mistakes on the training data).

2.2.2. Geometric Interpretation

The training process can be visually understood as finding a decision boundary that separates data points of different classes in the input space. The weights and bias of the perceptron define the orientation and position of this decision boundary.

Imagine a 2D input space with two input features (x_1 and x_2) and a perceptron with weights w_1 and w_2 and bias b. The decision boundary is a line defined by the equation $w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$. The perceptron classifies points on one side of the boundary as one class (e.g., class 0) and points on the other side as the other class (e.g., class 1).

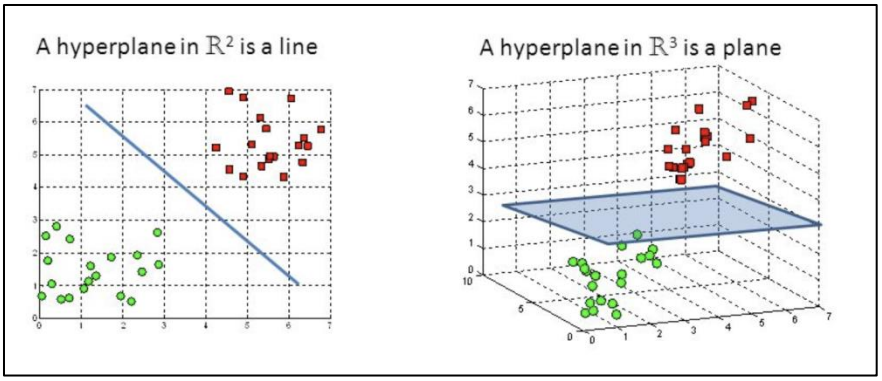
- $w1 \cdot x1 + w2 \cdot x2 + b > 0$, the perceptron predicts class 1.
- $w1 \cdot x1 + w2 \cdot x2 + b < 0$, the perceptron predicts class 0.

In this geometric interpretation, the weights w_1 and w_2 determine the slope of the decision boundary, and the bias b shifts it up or down along the vertical axis.

The perceptron learning algorithm adjusts these weights and bias during training to find a decision boundary that correctly separates the classes in the input space. If the data is linearly separable, the algorithm will eventually converge to a solution; otherwise, it may not converge.

2.2.3. Perceptron for Binary Classification

With this discrete output, controlled by the activation function, the perceptron can be used as a binary classification model, defining a linear decision boundary. It finds the separating hyperplane that minimizes the distance between misclassified points and the decision boundary.



2.3. Exercises

❖ **<u>Problem 1</u>: Implementing a Perceptron for Logical AND**

Let's consider a simple example of using a perceptron to classify a logical AND function. The AND function takes two binary inputs (0 or 1) and produces a binary output. The goal is to create a perceptron that correctly classifies the AND function. The initial weight and threshold are set to zero. The learning rate is set equal to 1.

input		**f(z) = y_{true}**
x_1	**x_2**	
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

❖ **<u>Problem 2</u>:**

Find the weights required to perform the following classification using perceptron network. The vectors (1,1,1,1) and (-1,-1,-1,-1) are belonging to the class 1, vectors (1,1,1,-1) and (1,-1,-1,1) are belonging to the class -1. Assume learning rate as 1, bias and initial weights as 0.

input				**f(z) = y_{true}**
x_1	**x_2**	**x_3**	**x_4**	
1	1	1	1	1
-1	1	-1	-1	1
1	1	1	-1	-1
1	-1	-1	1	-1

2.4. Limitations of the perceptron

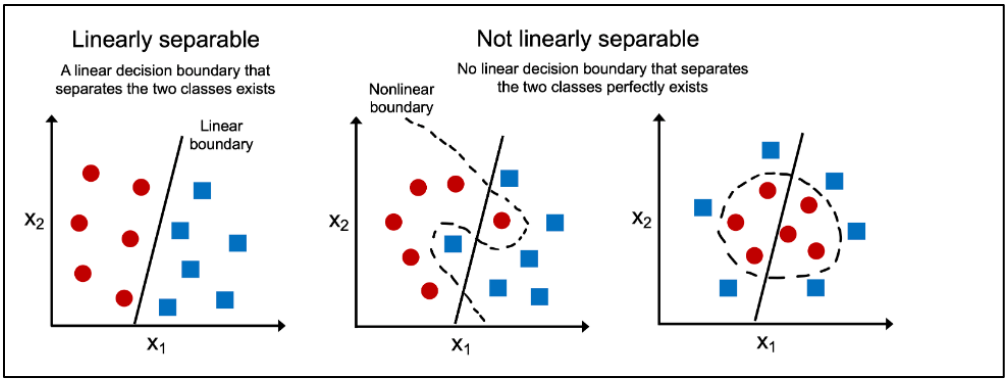
It's important to note that the perceptron can only find a linear decision boundary. If the data is not linearly separable, the perceptron will not converge to a solution. This limitation led to the development of more complex neural network architectures, such as multi-layer perceptron (MLPs), which can handle nonlinear data by introducing hidden layers with nonlinear activation functions

2.4.1. Example Scenarios: Xor Problem

The Xor (Exclusive OR) problem is a classic example that showcases the perceptron's limitations. In Xor, there are four possible input combinations of two binary values (0 or 1), and the target output is 1 if there is an odd number of 1s in the input.

input		**f(z) = y_{true}**
x_1	**x_2**	
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

The Xor problem is not linearly separable, and a single perceptron cannot correctly classify all four cases.



3. Multi-Layer Perceptron (MLP)

3.1. What is an MLP?

Multi-Layer Perceptrons, often referred to as neural networks or feedforward neural networks, are a class of artificial neural networks designed to overcome the limitations of the

single-layer perceptron. They are characterized by their ability to handle complex, non-linear relationships in data. They consist of multiple layers of interconnected neurons, each layer having its set of weights and biases. MLPs can be used for a wide range of tasks, including classification, regression, and even complex pattern recognition [4, 5].

3.2. Architecture of MLP

3.2.1. Structure of an MLP

MLP consist of multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer

i. **Input Layer:** The input layer of an MLP is responsible for accepting the raw input data. Each neuron in this layer corresponds to a feature in the input data. If you have n features, the input layer will have n neurons.

ii. **Hidden Layers**: Between the input and output layers, MLPs can have one or more hidden layers. These hidden layers are where the real computation happens. They introduce non-linearity into the network, enabling it to model complex, non-linear relationships in data. Each neuron in a hidden layer receives input from all neurons in the previous layer, applies a weighted sum, adds a bias term, and then applies an activation function.

- **Weighted Sum for Neuron *j* in Layer (*l*):**

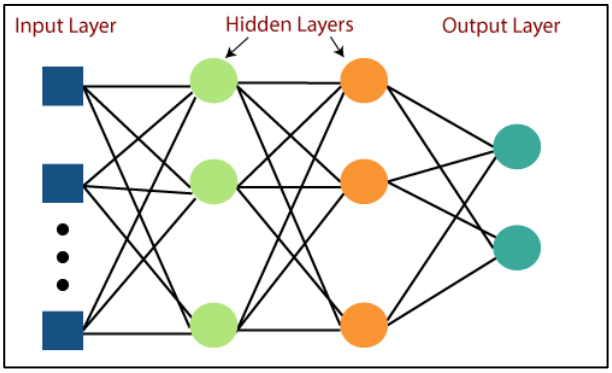
$$z_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)}\, a_i^{(l-1)} + b_j^{(l)}$$

Here, $N^{(l-1)}$ is the number of neurons in the previous layer, $w_{ij}^{(l)}$ is the weight connecting neuron i in layer l−1 to neuron j in layer l, $a_i^{(l-1)}$ is the output (activation) of neuron i in layer l−1, and $b_j^{(l)}$ is the bias for neuron j in layer l.

- **Activation Function:** The output of neuron j in layer l after applying an activation function, denoted as$a_j^{(l)}$, is typically a non-linear function of the weighted sum $z_j^{(l)}$. Common activation functions include the sigmoid function, hyperbolic tangent (tanh), and the rectified linear unit (ReLU).

iii. **Output Layer:** The output layer produces the final prediction or classification. The number of neurons in this layer depends on the problem you are trying to solve. For binary classification, there is typically one neuron with a sigmoid activation function. For multi-class

classification, there is usually one neuron per class with softmax activation. For regression

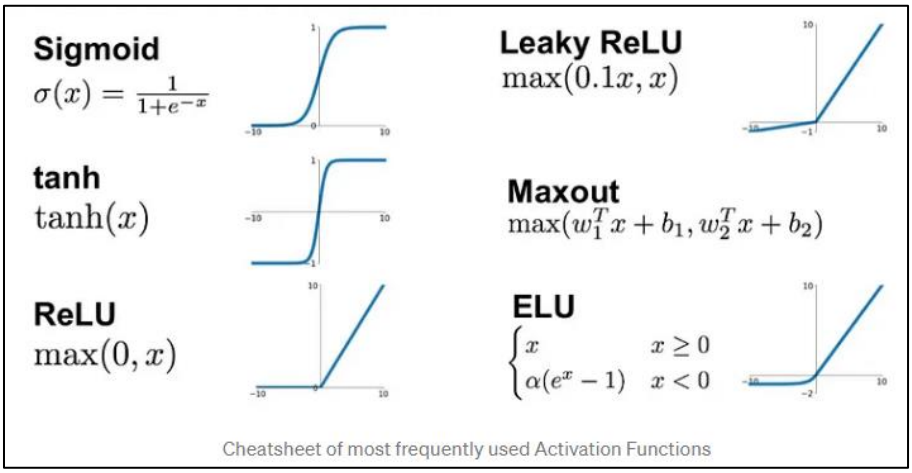


tasks, you might have a single neuron with a linear activation.

Figure: Multilayer Perceptron (MLP) neural network architecture.

3.2.2. Activation Functions

There are two types of activation functions: *Linear Activation Functions* (f(x)=x, which basically passes the input without much modification) and *Non-Linear Activation Functions*. I will be discussing most frequently used activation functions nowadays.



3.3. Training a Multi-Layer Perceptron (MLP)

The main algorithm used to train MLPs is a combination of two key techniques: backpropagation and gradient descent. This combination is often referred to as "Backpropagation with Gradient Descent." The primary goal is to adjust the weights and biases of the network in

such a way that it can make accurate predictions. Here's an overview of the main steps in training an MLP using this algorithm:

1. **Initialization:**

Initialize the weights and biases of the MLP, typically with small random values. These parameters are learned during training.

2. **Forward Propagation**

Forward propagation is the process of computing predictions made by the MLP for a given input. It involves passing the input data through the network's layers, applying weights, biases, and activation functions to calculate the output.

Input Layer: The input data is passed to the input layer, where each neuron corresponds to one feature. Each neuron in the input layer simply passes its input value to the neurons in the first hidden layer.

Hidden Layers: In the hidden layers, each neuron calculates a weighted sum of the outputs from the previous layer and adds a bias term. This is known as the weighted sum $z_j^{(l)}$ for neuron *j* in layer *l*:

$$z_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

Here, $N^{(l-1)}$ is the number of neurons in the previous layer, $w_{ij}^{(l)}$ is the weight connecting neuron *i* in layer *l*-1 to neuron *j* in layer *l*, $a_i^{(l-1)}$ is the output (activation) of neuron *i* in layer *l*-1, and $b_j^{(l)}$ is the bias for neuron *j* in layer *l*.

Then, an activation function (e.g., sigmoid, tanh, ReLU) is applied to the weighted sum to produce the output (activation) of the neuron a_j^l:

$$a_j^l = Activation\ z_j^{(l)}$$

Output Layer: Similar to the hidden layers, the output layer computes a weighted sum and applies an activation function. However, the choice of activation function depends on the type of task:

- For binary classification, a sigmoid activation function is typically used.
- For multi-class classification, the softmax activation function is common.
- For regression tasks, a linear activation function may be used.

3. Loss Computation

After forward propagation, the next step is to compute the loss, which measures the error between the predicted outputs and the ground truth (target) values. The choice of loss function depends on the task, but common choices include mean squared error (MSE) for regression and cross-entropy for classification.

- **Mean Squared Error (MSE) Loss (Regression):**

If the MLP is used for regression, the MSE loss is commonly used:

$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Where N is the number of data points, y_i is the true target value for data point *i*, and $\hat{y}_i$ is the predicted value for data point *i*.

- **Cross-Entropy Loss (Classification):**

For classification problems, the cross-entropy loss is widely used:

$$Cross - Entropy\ Loss\ = \frac{1}{N}\sum_{i=1}^{N}(y_i log(\hat{y}_i) + (1 + y_i)\log(1 - \hat{y}_i))$$

Here, N is the number of data points, y_i is the true class label (0 or 1), and $\hat{y}_i$ is the predicted probability of belonging to class 1 for data point *i*.

4. Back-propagation (gradient calculation):

Calculate the gradient of the loss with respect to the network's weights and biases by propagating errors backward through the network. This involves computing the error terms (δ) for each neuron in each layer. Update these error terms recursively, starting from the output layer and working backward to the input layer.

❖ **Error Computation**

Output Layer:

For the output layer, the error $\delta_j^{(L)}$ for neuron *j* is calculated as the derivative of the loss function with respect to the weighted sum$z_j^{(L)}$ and the derivative of the activation function:

$$\delta_j^{(L)} = \frac{\partial Loss}{\partial z_j^{(L)}} * \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$$

Hidden Layers:

For the hidden layers, the error ($\delta_j^{(l)}$) for neuron *j* in layer *l* is computed based on the errors from the layer *l*+1 and the weights connecting the neurons in layers *l* and *l*+1. This is done recursively from the output layer to the input layer.

$$\delta_j^{(l)} = \frac{\partial z_j^{(l)}}{\partial a_j^{(l)}} \sum_k \delta_k^{(l+1)} * \frac{\partial a_k^{(l+1)}}{\partial z_k^{(l+1)}}$$

Here, $\frac{\partial z_j^{(l)}}{\partial a_j^{(l)}}$ is the derivative of the weighted sum with respect to the activation for neuron *j* in layer *l*.

❖ **Weight and Bias Update**

Once the errors ($\delta_j^{(l)}$) for all neurons in all layers have been computed, the weights and biases can be updated using these error values.

- Weight Update:

The update rule for a weight $w_{ij}^{(l)}$ connecting neuron *i* in layer *l*−1 to neuron *j* in layer *l* is typically based on the error ($\delta_j^{(l)}$) for neuron *j* and the output ($a_i^{(l-1)}$) from neuron *i* in the previous layer. The learning rate η is also a part of the update rule:

$$\Delta w_{ij}^{(l)} = -\eta * \delta_j^{(l)} * a_i^{(l-1)}$$

$$w_{ij}^{(l)} = w_{ij}^{(l)} + \Delta w_{ij}^{(l)}$$

- Bias Update:

The bias update is similar to the weight update, but there is no input from the previous layer. The update rule for bias $b_j^{(l)}$ in layer *l* is:

$$\Delta b_{ij}^{(l)} = -\eta * \delta_j^{(l)}$$

$$b_j^{(l)} = b_j^{(l)} + \Delta b_j^{(l)}$$

5. Gradient Descent:

Use the computed gradients to update the weights and biases of the network. Gradient descent optimization algorithms (e.g., stochastic gradient descent, Adam, RMSprop) are employed to adjust the parameters. The learning rate hyperparameter controls the step size in weight and bias updates. It determines the balance between convergence speed and stability.

6. **Iteration**:

Repeat steps "2 to 5 for multiple iterations (epochs). Each iteration refines the model's parameters to reduce the loss. Training continues until a predefined stopping criterion is met, such as a maximum number of epochs or when the loss converges to a satisfactory level.

7. **Evaluation**:

After training, evaluate the trained MLP on a separate validation or test dataset to assess its generalization performance. Fine-tuning and hyperparameter tuning may be performed based on validation results.

The Backpropagation with Gradient Descent algorithm iteratively adjusts the MLP's parameters to minimize the loss function, allowing the network to learn from data and make accurate predictions. This process is fundamental to training neural networks, including MLPs, for various machine learning tasks.

3.4. Applications of MLPs

MLPs find extensive use in a wide range of real-world applications, including but not limited to:

Image Classification: MLPs are used to classify images into various categories, making them vital components of computer vision systems.

Natural Language Processing (NLP): In NLP tasks, such as text classification and sentiment analysis, MLPs are employed to process and analyze text data.

Financial Forecasting: MLPs are used for time series analysis and forecasting, helping predict stock prices, currency exchange rates, and more.

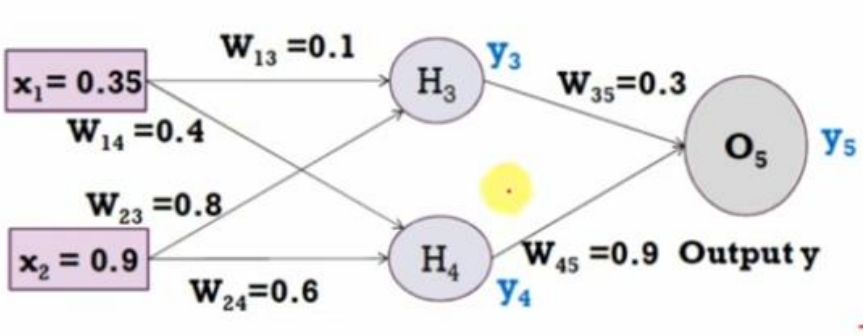
Healthcare: In healthcare, MLPs are used for disease diagnosis, medical image analysis, and drug discovery.

Recommendation Systems: MLPs play a significant role in recommendation systems, such as those used by streaming platforms to suggest movies or songs.

3.5. Exercise

Let's assume that the neurons within the MLP are equipped with sigmoid activation functions. Your task is to perform a forward pass, followed by a backward pass. Additionally, consider that

the network's current output, labeled 'y,' is set to 0.5, and the learning rate is fixed at 1. To gain insight into the network's behavior, you will also conduct another forward pass.



3.6. Challenging problem with MLP

As the quantity of data increased the parameters of ANN also increased. With advancement in technology, Classification tasks were also required for image and text files but on using ANN, it was found that the computational powers sky-rocketed as the parameters increased to 100 thousands in numbers even for a small 8-bit image. Therefore, there was a need for an another type of Neural Network which would compensate for the sudden increase in computational powers required for applying Classifictaion problems to Images and Text Files.

Let's see the problems ourselves !

Suppose we want to apply classifiction problems to a 16-bit image. A 2D-grayscale image would 16 * 16 = 256 pixels. Therefore, it will require a neural network with a input layer of 256 Neurons. Suppose, the image is colored then it will require 16 * 16 * 16 = 4096 Neurons in the input layer itself. Thus a huge amount of computational power is required for this task**.**

4. Conclusion

This chapter introduced the foundational elements of neural networks: the perceptron and the multi-layer perceptron (MLP). We explored the perceptron's basic structure and limitations, while highlighting the essential role of MLPs in addressing complex real-world problems. This knowledge serves as a fundamental stepping stone for deeper understanding in the realm of neural networks and paves the way for further exploration in the field of deep learning.

Chapter 3:

Deep Learning

1. Introduction

In this chapter, we'll dive into the foundations of deep learning, placing a special spotlight on Convolutional Neural Networks (CNNs). You'll get a grasp of what deep learning is all about, especially when it comes to dealing with images. By the end of this chapter, you'll be well-versed in how CNNs function and their crucial role in tasks like recognizing objects in pictures.

2. Deep Learning

2.1. What is Deep Learning?

Deep Learning is a subfield of machine learning and artificial intelligence that revolves around training artificial neural networks, particularly deep neural networks, to perform complex tasks. These deep neural networks consist of multiple layers, which enable them to capture intricate patterns in data[6].

2.2. Deep Neural Networks

Deep neural networks are characterized by having many hidden layers. These layers introduce a hierarchical representation of data. A deep neural network consists of multiple layers, where:

Input Layer: Receives input data as a vector or tensor.

Hidden Layers: These intermediate layers apply linear transformations and activation functions to the input data sequentially.

Output Layer: Produces the final prediction or output.

Each layer is represented mathematically as a function. Given an input vector x for a layer i, the output h_i is computed as follows: $hi=f(Wi \cdot x+bi)$

Where:

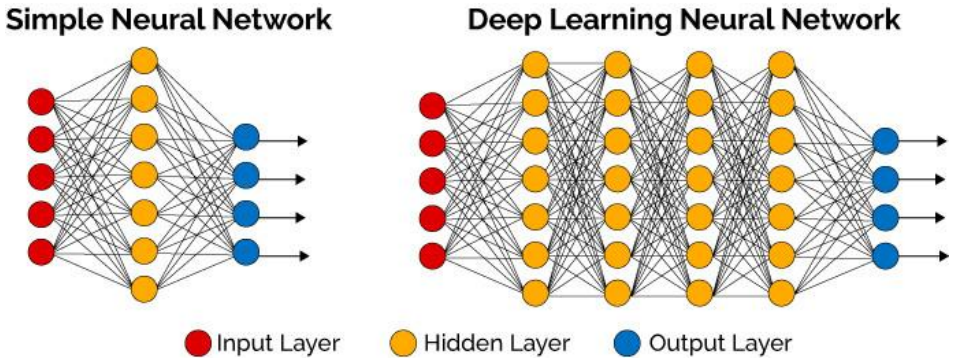
- h_i is the output of layer i.
- W_i represents the weights associated with layer i.
- b_i is the bias term for layer i.
- $f(\cdot)$ is the activation function applied element-wise.

Mathematically, a deep neural network can be represented as a composition of functions:

$$F(x)=f_n(f_{n-1}(\dots f_2(f_1(x;W_1,b_1);W_2,b_2)\dots;W_{n-1},b_{n-1});W_n,b_n)$$

Where:

- $F(x)$ represents the overall function mapping input x to the final prediction.



2.3. Panorama of Prominent Deep Learning Models

Deep learning, a subset of machine learning, boasts a diverse family of common models that serve as the building blocks for groundbreaking applications across various domains. Here, we'll introduce some of the most prominent and widely used models in deep learning:

- Convolutional Neural Networks (CNNs): CNNs have revolutionized computer vision and image analysis. They are adept at recognizing patterns and features within images, making them indispensable in tasks like image classification, object detection, and facial recognition.
- Long Short-Term Memory Networks (LSTMs): LSTMs are tailored for sequential data and are invaluable in tasks involving time series, natural language processing, and speech recognition. They excel at capturing dependencies and context within sequences.
- Autoencoders: Autoencoders are unsupervised learning models used for feature extraction and dimensionality reduction. They find applications in image denoising, anomaly detection, and data compression.
- Generative Adversarial Networks (GANs): GANs are famous for generating realistic data, often used in image generation, style transfer, and data augmentation. They consist of a generator and a discriminator network, locked in a competitive learning process.
- Transformers: Transformers have brought a revolution in natural language processing. They utilize self-attention mechanisms to capture contextual information effectively and are the cornerstone of state-of-the-art language models like BERT and GPT.
- Deep Reinforcement Learning (DRL): DRL combines deep learning with reinforcement learning, enabling the development of intelligent agents that can learn to perform tasks

through trial and error. These agents are utilized in gaming, robotics, and autonomous systems.

- Capsule Networks (CapsNets): CapsNets are designed to overcome the limitations of CNNs in understanding spatial hierarchies in images. They are emerging as a promising alternative for various computer vision tasks.

These models represent just a fraction of the vast and evolving landscape of deep learning. Each one has its unique characteristics and applications, allowing researchers and engineers to choose the most suitable tool for their specific tasks and domains.

3. Differences between deep neural network and machine learning

Deep neural networks (DNNs) and machine learning are related fields, but there are distinct differences between them. Here are some key differences:

Scope:

- Machine Learning: Machine learning is a broader field that encompasses various techniques and algorithms for teaching a computer system to learn from data and make predictions or decisions. It includes both traditional statistical methods and modern techniques like deep learning.
- Deep Neural Networks: DNNs are a specific subset of machine learning techniques that are based on artificial neural networks with multiple hidden layers. They are a more specialized and recent development within the field of machine learning.

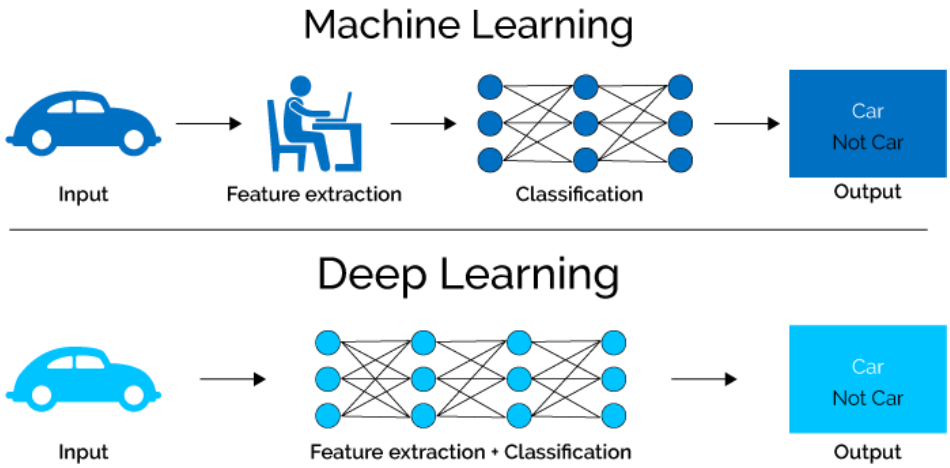
Model Complexity:

- Machine Learning: Machine learning models can be relatively simple, such as linear regression, decision trees, or support vector machines. They do not necessarily involve deep architectures.
- Deep Neural Networks: DNNs are characterized by their deep architectures, typically consisting of many layers (hence the term "deep"). They are designed to automatically learn hierarchical representations of data.

Feature Engineering:

- Machine Learning: In traditional machine learning, feature engineering is often required. This involves selecting and transforming relevant input features to make them suitable for the learning algorithm.

- Deep Neural Networks: DNNs can automatically learn relevant features from raw data, reducing the need for extensive manual feature engineering. This is one of their significant advantages.



Data Requirements:

- Machine Learning: Traditional machine learning models can work with smaller datasets effectively, as they often rely on hand-crafted features.
- Deep Neural Networks: DNNs tend to require larger datasets for training, as they need substantial amounts of data to learn complex patterns.

Training Process:

- Machine Learning: Training traditional machine learning models is often quicker and less computationally intensive compared to training deep neural networks.
- Deep Neural Networks: Training DNNs can be computationally expensive and time-consuming, often requiring specialized hardware like GPUs or TPUs.

Interpretability:

- Machine Learning: Many traditional machine learning models are relatively interpretable, meaning it's easier to understand the factors influencing their predictions.
- Deep Neural Networks: DNNs are often considered as "black boxes" because their inner workings can be challenging to interpret.

In summary, deep neural networks are a subset of machine learning techniques that focus on automatically learning hierarchical representations from data, especially in cases where traditional machine learning methods may not perform as well. The choice between machine

learning and deep learning depends on the specific problem, the available data, and the desired level of model complexity and interpretability.

4. Convolutional Neural Networks (CNNs)

Within the expansive realm of deep learning models, the Convolutional Neural Network (CNN) stands out as the beacon that brings a dazzling brilliance to the field. Convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. A CNN typically has three layers: a convolutional layer, pooling layer, and fully connected layer.

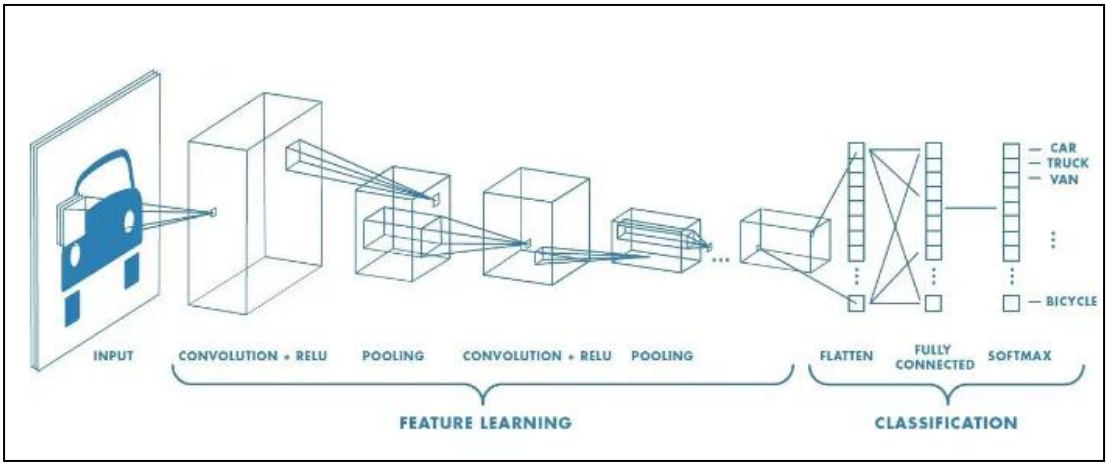


Figure: convolutional neural network architecture

4.1. Convolutional Layers

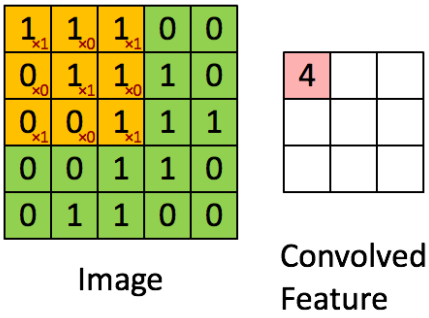
Convolutional layer is the core building block of CNN. The main objective of convolution is to extract features such as edges, colors, corners from the input. As we go deeper inside the network, the network starts identifying more complex features such as: shapes, digits, face parts as well. This layer applies a set of learnable filters to the input data to extract local patterns. The output of a convolutional layer is obtained by convolving the input with multiple filters, producing a set of feature maps. Mathematically, the output of a convolutional layer can be expressed as:

$$H_i = f(W_i * X + b_i)$$

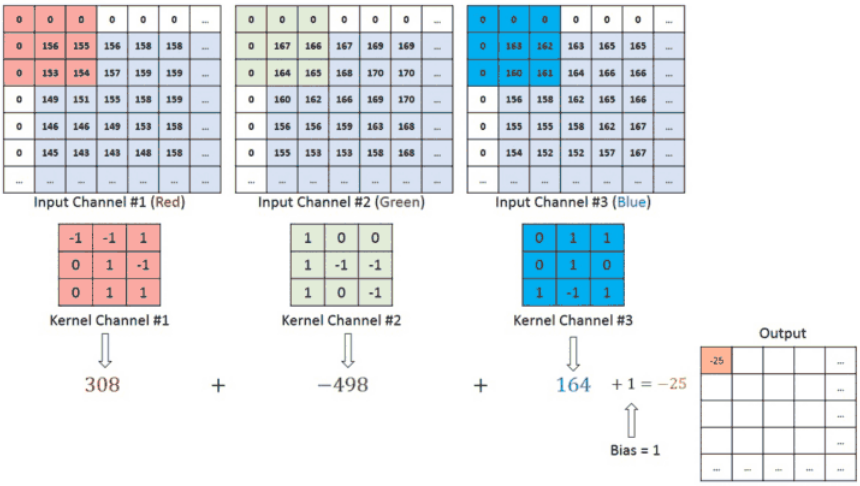
Where:

- H_i represents the feature map produced by the i^{th} filter.
- W_i denotes the weights associated with the i^{th} filter.

- X is the input data.
- b_i is the bias term for the i^{th} filter.



If the image is RGB then the filter will have smaller height and width compared to the image but it will have the same depth (height x width x **3**) as of the image. For RGB images, the convolving part can be visualized as follows:



At the end of the convolution process, we have a **featured matrix** which has lesser parameters (dimensions) than the actual image as well as more clear features than the actual one. So, now we will work with our featured matrix from now on.

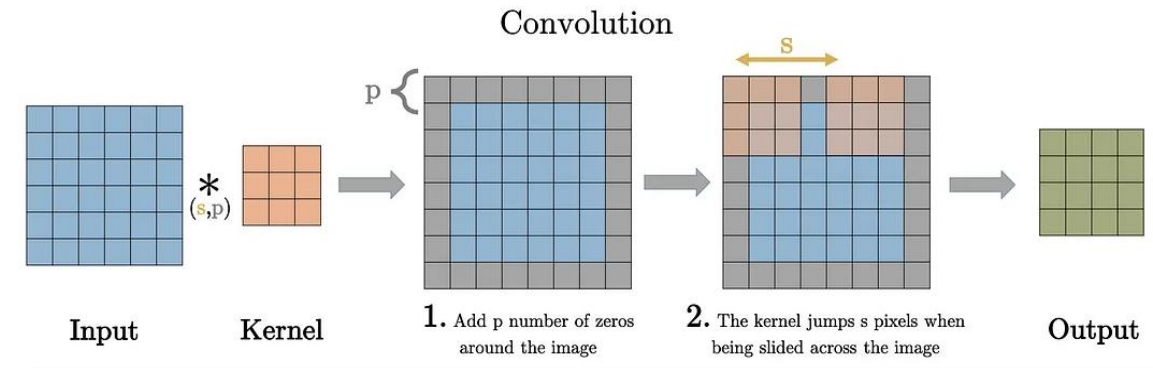
❖ **Convolutional layer parameters**

First we need to agree on a few parameters that define a convolutional layer.

Kernel Size: defines the field of view of the convolution

Padding (p): defines how the border of a sample is handled. The number of zeros padded around the original input increasing the size to (i+2*p)x(i+2*p).

Stride: defines the step size of the kernel when traversing the input image.



In the first step, the input image is padded with zeros, while in the second step the kernel is placed on the padded input and slid across generating the output pixels as dot products of the kernel and the overlapped input region. The kernel is slid across the padded input by taking jumps of size defined by the stride. The convolutional layer usually does a down-sampling i.e. the spatial dimensions of the output are less than that of the input.

4.1.1. Different Types of Convolutions in Deep Learning

Dilated convolutions (atrous convolutions): introduce another parameter to convolutional layers called the **dilation rate**. This defines a spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 will have the same field of view as a 5x5 kernel, while only using 9 parameters. Imagine taking a 5x5 kernel and deleting every second column and row.

This delivers a wider field of view at the same computational cost.

Dilated convolutions are particularly popular in the field of real-time segmentation. Use them if you need a wide field of view and cannot afford multiple convolutions or larger kernels.

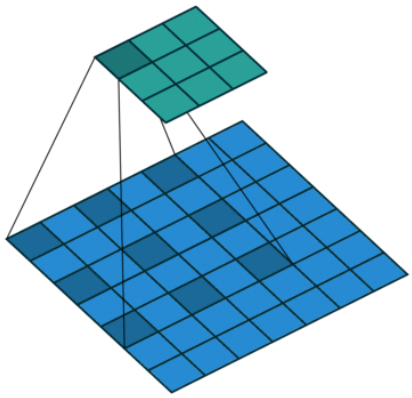
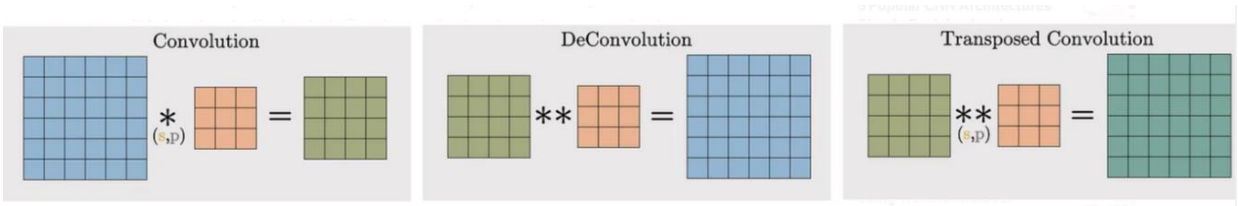


Figure: 2D convolution using a 3 kernel with a dilation rate of 2 and no padding

Transposed Convolutions: Some sources use the name deconvolution, which is inappropriate because it's not a deconvolution. A deconvolutional layer reverses the operation of a standard convolutional layer i.e. if the output generated through a standard convolutional layer is deconvolved, you get back the original input.

The transposed convolutional layer is similar to the deconvolutional layer in the sense that the spatial dimension generated by both is the same. A transposed convolutional layer, on the other hand, is usually carried out for upsampling i.e. to generate an output feature map that has a spatial dimension greater than that of the input feature map.

The transposed convolutional layer, like the regular convolutional layer, uses padding and stride values. These values are used to theoretically reverse the process that was applied to the output in order to produce the input. In other words, if you take the output and apply a regular convolution with the specified stride and padding, it will result in the same spatial dimensions as the input.



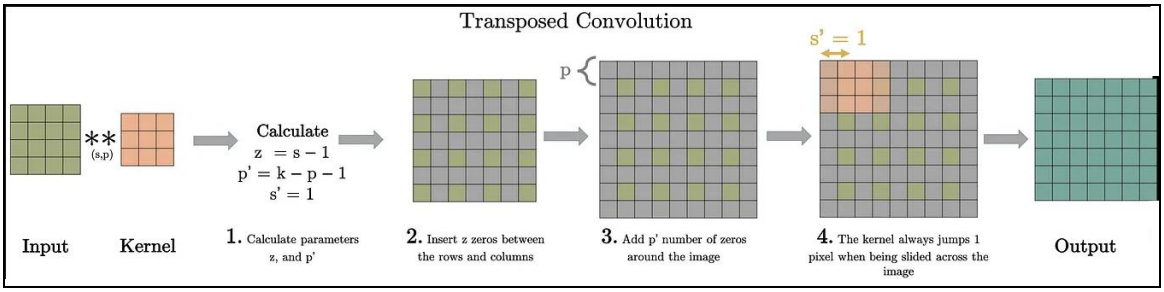
Implementing a transposed convolutional layer can be better explained as a 4 step process

Step 1: Calculate new parameters z and p'

Step 2: Between each row and columns of the input, insert z number of zeros. This increases the size of the input to ***(2*i-1)x(2*i-1)***

Step 3: Pad the modified input image with p' number of zeros

Step 4: Carry out standard convolution on the image generated from step 3 with a stride length of 1. The complete steps can be seen in the figure below.



Separable Convolutions

In a separable convolution, we can split the kernel operation into multiple steps. Let's express a convolution as **y = conv(x, k)** where **y** is the output image, **x** is the input image, and **k** is the kernel. Next, let's assume k can be calculated by: **k = k1.dot(k2)**. This would make it a separable convolution because instead of doing a 2D convolution with k, we could get to the same result by doing two 1D convolutions with k1 and k2.

For instance, consider the Sobel kernel, which is commonly used in image processing. Instead of using 9 values, you can create the same kernel by multiplying two smaller sets of numbers: [1, 0, -1] and $[1, 2, 1]^T$. This way, you only need 6 parameters instead of 9 to achieve the same result. This method is known as spatial separable convolution, although it's not typically used in deep learning as far as I know.

-1	0	+1
-2	0	+2
-1	0	+1

x filter

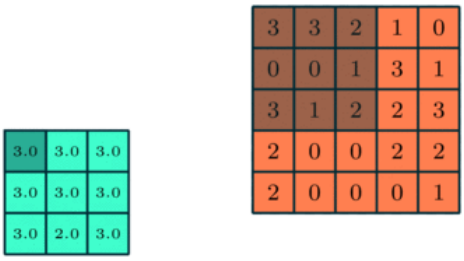
+1	+2	+1
0	0	0
-1	-2	-1

y filter

Sobel X and Y filters

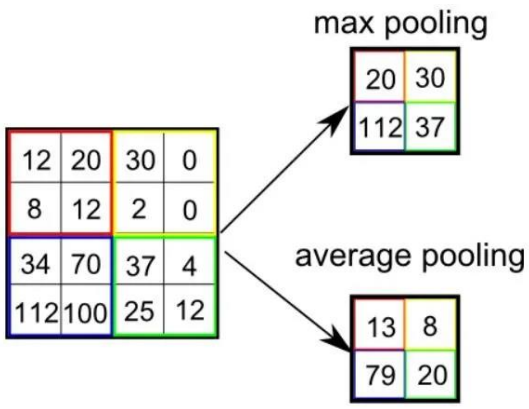
4.2. Pooling Layers

Pooling layers, also called sub-sampling layers, are used to reduce the amount of computer work needed to handle the data. They shrink the size of the featured matrix. In this layer, we focus on finding the most important features in a small area. Let's make it easier to understand with an example



The orange matrix is our featured matrix, the brown one is a pooling kernel and we get our blue matrix as output after pooling is done. So, here what we are doing is taking the maximum amongst all the numbers which are in the pooling region and shifting the pooling region each time to process another neighborhood of the matrix.

There are two types of pooling techniques: *AVERAGE-pooling* and *MAX- pooling.* The difference between these two is, in *AVERAGE-pooling*, we take the average of all the values of pooling region and in *MAX-pooling*, we just take the maximum amongst all the values lying inside the pooling region.

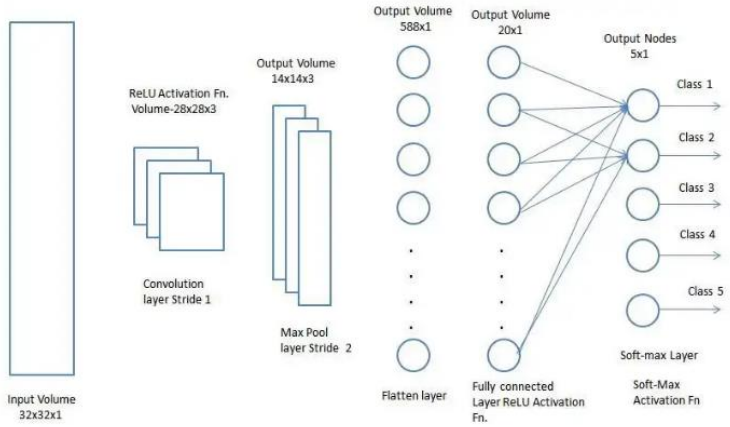


So, after pooling layer, we have a matrix containing main features of the image and this matrix has even lesser dimensions, which will help a lot in the next step.

4.3. Fully Connected Layers

So far, we've been focusing on finding and simplifying features in images, but we haven't started the classification process yet. Now, we're ready to classify images. After converting the input image into a suitable format for our Multi-Level fully connected architecture, we flatten the image into a single column of numbers. This flattened data is then sent through a feed-forward neural network, and we use back-propagation during training to make the network learn. As we

go through multiple training cycles (epochs), the model becomes capable of recognizing and classifying important features in images.



Important note: Common Set-up

In order to implement CNN, most successful architecture uses one or more stacks of convolution + pool layers with Relu activation, followed by a fatten layer then one or two dense layers.

4.4. Various architectures of CNNs

Convolutional Neural Networks (CNNs) have become the cornerstone of many computer vision tasks, such as image classification, object detection, and image segmentation. Over the years, several CNN architectures have been developed, each with its own unique design and characteristics. Here are some of the most prominent CNN architectures:

- LeNet-5: An early CNN used for handwritten digit recognition.
- AlexNet: A pioneering deep CNN that popularized the use of deep learning for image classification.
- VGGNet: Known for its uniform architecture and various depth configurations.
- GoogLeNet (Inception): Introduced "Inception modules" for efficient computation.
- ResNet: Introduced residual connections to train very deep networks.
- MobileNet: Designed for mobile and embedded vision applications with efficient depth-wise separable convolutions.
- DenseNet: Features densely connected layers for better feature reuse.
- Xception: Utilizes depthwise separable convolutions for efficiency.
- EfficientNet: Balances model size and accuracy through compound scaling.

- SqueezeNet: A lightweight and efficient model suitable for resource-constrained applications.

4.5. Key Differences between MLP and CNN

- CNN is mostly used for Image Data, whereas it is better to use MLP on structural data
- CNN has less parameter and tries to reduce the dimensions of image whereas in case of MLP number of parameters depends on the data
- CNN is complex in nature whereas MLP is relatively simple compared to CNN
- CNN uses special Convolution and Pooling Layers whereas MLP is just a network of Neurons
- CNN is generally used for huge or bulky data as compared to MLP

5. Key terminologies and techniques in DL

In this section, we will present fundamental concepts and techniques in the field of deep learning.

5.1. Terminologies in DL

In the following, we will introduce the fundamental concepts that play a pivotal role in deep learning:

5.1.1. What Is a Sample?

A sample is a single row of data. It contains inputs that are fed into the algorithm and an output that is used to compare to the prediction and calculate an error. A training dataset is comprised of many rows of data (many samples). A sample may also be called an instance, an observation, an input vector, or a feature vector.

For example, in the context of image classification, a sample might be a single image along with its corresponding label (e.g., a picture of a cat with a label "cat"). In natural language processing, a sample could be a single text document along with its associated category or sentiment label. Essentially, a sample is a unit of data used to train or test a deep learning model.

5.1.2. What Is a Batch?

In the context of deep learning and neural network training, a "batch" refers to a subset of the dataset that is used during a single iteration of the training process. Instead of updating the

model's parameters after processing each individual sample (a process known as stochastic gradient descent), batches are used to update the model's parameters in a more efficient manner.

Batch Size in Deep Learning: A Critical Component of Efficient Gradient Descent

The batch size is a hyperparameter that determines the number of samples in each batch. It can be set to various values, depending on the specific deep learning task and available computational resources. Common batch sizes are 32, 64, 128, or other powers of 2.

Batch size and gradient descent are closely related in the context of training deep neural networks. The choice of batch size can have a significant impact on the training process and the behavior of gradient descent algorithms. Here's how they are connected:

a. **Batch Gradient Descent:** In batch gradient descent, the entire training dataset is used to compute the gradient of the loss function with respect to the model's parameters in each training iteration. The model's parameters are then updated based on this aggregate gradient. This approach provides stable updates but can be computationally expensive, especially for large datasets.
   - **Pros:** Accurate and stable updates, typically converges to a good solution.
   - **Cons:** Requires storing the entire dataset in memory, which may not be feasible for large datasets. Slow convergence when the dataset is large.

b. **Stochastic Gradient Descent (SGD):** In stochastic gradient descent, the model's parameters are updated after processing each individual sample in the training dataset. This approach introduces noise into the gradient updates and can lead to erratic progress, but it helps the model escape local minima and is computationally efficient.
   - **Pros:** Very fast updates, potential for escaping local minima.
   - **Cons**: Noisy updates, which can slow convergence and make training erratic.

c. **Mini-Batch Gradient Descent:** Mini-batch gradient descent strikes a balance between batch gradient descent and stochastic gradient descent. Instead of using the entire dataset or just one sample, it divides the dataset into small, equally sized batches. The model's parameters are updated after processing each batch. The batch size is a hyperparameter that you can control.
   - **Pros:** Efficient updates due to parallel processing, a balance between stability and noise, better convergence compared to pure SGD.
   - **Cons:** Still introduces some level of noise compared to batch gradient descent.

Choosing the appropriate batch size in mini-batch gradient descent is essential for effective training. The optimal batch size can vary depending on factors like the dataset size, model architecture, available memory, and computational resources. Common batch sizes used in practice range from small values like 32 or 64 to larger values like 128 or 256.

In general, larger batch sizes provide more stable updates and are more computationally efficient, while smaller batch sizes introduce more noise but can help the model escape local minima and may generalize better. Therefore, the choice of batch size is often a matter of experimentation and hyper-parameter tuning to find the setting that works best for your specific deep learning task

Note: What if the dataset does not divide evenly by the batch size?
You can remove some samples from the dataset or change the batch size such that the number of samples in the dataset does divide evenly by the batch size

5.1.3. What is epoch and iteration

In the context of machine learning and deep learning, "epoch" and "iteration" are related but distinct terms:

Epoch:

An epoch is one complete pass through the entire training dataset. In other words, it's when the model has seen and been trained on every example in the dataset once. During an epoch, the model's parameters are updated multiple times based on the training data to learn patterns and improve its performance.

The number of epochs is typically a hyperparameter that you can specify when training a machine learning model. You decide how many times the model should go through the entire dataset during training. The right number of epochs can vary depending on the complexity of the problem and the size of the dataset. It's often determined through experimentation and validation.

Iteration (Time Step):

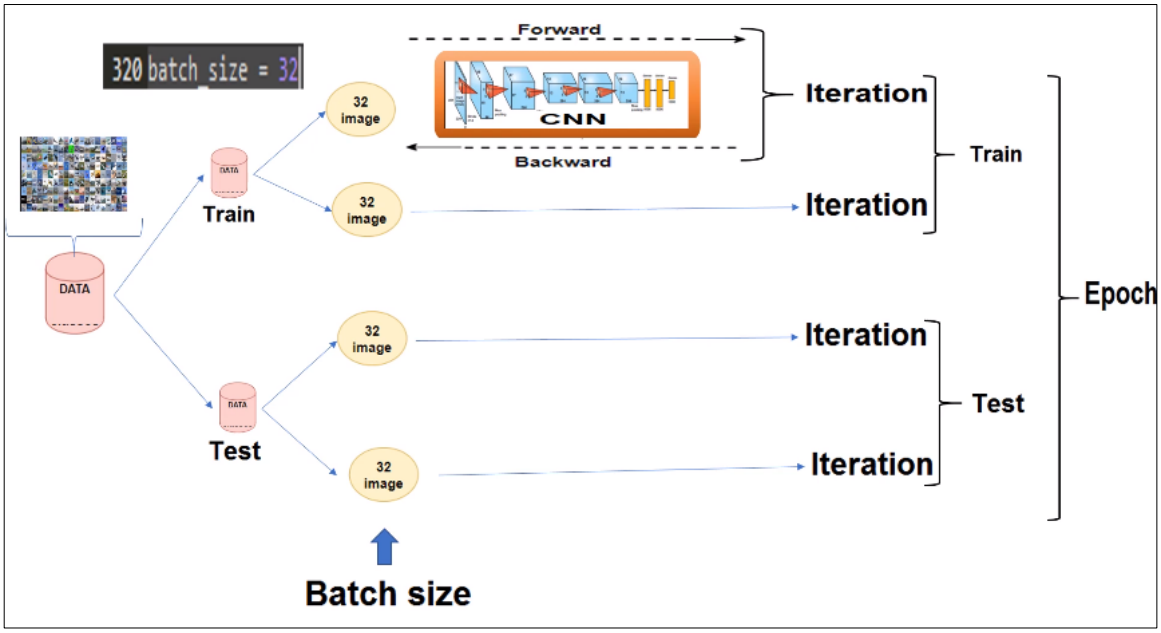
An iteration (or time step) refers to a single update of the model's parameters. It usually occurs for each mini-batch of data. In other words, during an epoch, there are multiple iterations where the model sees a subset of the data (a mini-batch), computes the gradients, and updates its parameters.

The process of training a machine learning model typically involves running multiple epochs, and within each epoch, you have multiple iterations as the model learns from different subsets of the data. The learning rate, batch size, and other hyperparameters influence how often the model's parameters are updated during each iteration.

Example

Let’s make this concrete with a small example.

Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs. This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples. This also means that one epoch will involve 40 batches or 40 updates to the model. With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process



4.1.4. Transfer learning

Transfer learning is a technique where a pre-trained neural network, often trained on a large dataset, is used as a starting point for a different but related task. Instead of training a new network from scratch, you fine-tune the pre-trained model to adapt to your specific problem by adjusting its final layers

4.1.5. Softmax in DL

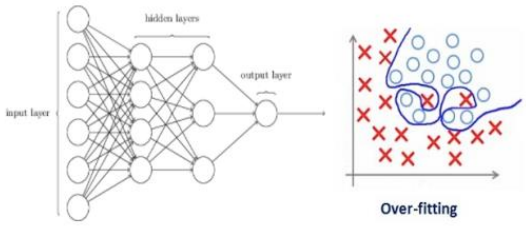
Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would

5.2.Techniques in DL

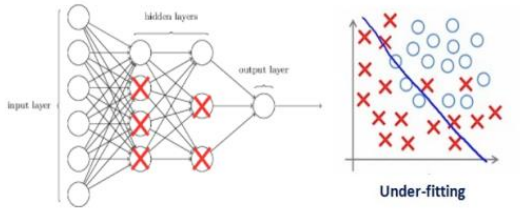
In the following, we will introduce the fundamental techniques that play a pivotal role in deep learning:

5.2.1. Regularization in Deep Learning

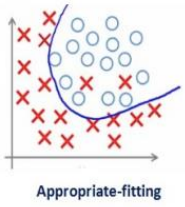
In deep learning, regularization is a technique used to prevent overfitting. It is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the model's performance on the unseen data as well. In deep learning, it actually penalizes the weight matrices of the nodes.



Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero.



Such a large value of the regularization coefficient is not that useful. We need to optimize the value of regularization coefficient in order to obtain a well-fitted model as shown in the image below.



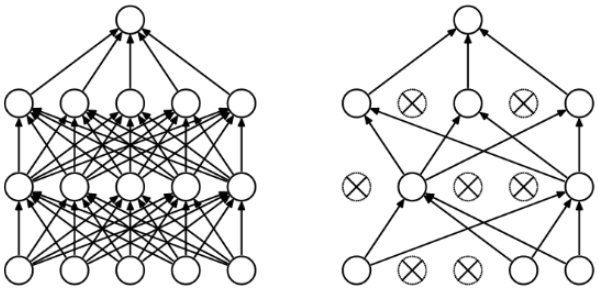
Different Regularization Techniques in Deep Learning

Here, we will introduce several techniques for implementing regularization in the context of deep learning.

Dropout:

Dropout is a regularization technique unique to deep neural networks. During training, dropout randomly deactivates a fraction of neurons in each layer for each input example. This prevents the network from becoming overly reliant on specific neurons.

The primary hyperparameter for dropout is the dropout rate, which determines the probability that a neuron will be deactivated during training. Common values range from 0.2 to 0.5, but the choice depends on the problem and the degree of regularization required



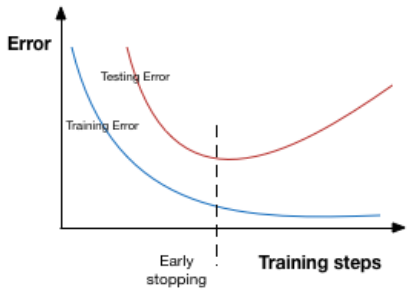
(a) Standard Neural Net (b) After applying dropout.

Data Augmentation:

The simplest way to reduce over-fitting is to increase the size of the training data. In the case we are dealing with images, there are a few ways of increasing the size of the training data – rotating the image, flipping, scaling, shifting, etc. This technique usually provides a big leap in improving the accuracy of the model.

Early stopping:

In Early stopping, we keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model. In the image below, we will stop training at the dotted line since after that our model will start overfitting on the training data.



5.2.2. Batch Normalization in Deep Learning:

In deep learning, batch normalization is a technique that normalizes the input of each layer within a mini-batch during training. It helps stabilize the training process by reducing the internal covariate shift, which is the change in the distribution of layer inputs during training.

Batch normalization is important in deep learning because it accelerates training, makes convergence more stable, and reduces the sensitivity to weight initialization. It enables the use of higher learning rates and often results in better generalization.

Batch normalization is implemented as a layer within a neural network. It computes the mean and standard deviation of the inputs within a mini-batch and normalizes them using these statistics.

6. Conclusion

In this chapter, we explored the fundamentals of deep learning, with a specific focus on Convolutional Neural Networks (CNNs). Deep learning, empowered by CNNs, has not only revolutionized the field of computer vision but also made significant strides in other domains. As deep learning continues to evolve, we can anticipate even more exciting developments that will reshape the landscape of artificial intelligence and its practical applications.

Chapter 4:

Bayesian Network

1. Introduction

Bayesian networks [7], also known as belief networks or probabilistic graphical models, are powerful tools for representing and reasoning about uncertainty and probabilistic dependencies in complex systems. In this chapter, we will delve into the fundamentals of Bayesian networks, their graphical representation, and how to perform probabilistic inference using them. We'll also delve into important concepts like probability and conditional probability, aiming to provide a solid understanding of how Bayesian networks function and their usefulness in various situations.

1.1. Learning Objectives

- Understand the fundamental concepts of Bayesian networks.
- Learn how to represent and manipulate Bayesian networks.
- Explore probabilistic reasoning and inference in Bayesian networks.
- Gain insight into applications of Bayesian networks in various fields.

2. Probabilistic reasoning in Artificial intelligence

Before we delve into understanding Bayesian Networks, which rely on probabilistic reasoning and related terms, let's first familiarize ourselves with some basic concepts:

2.1. Uncertainty

Certainty relates to a strong confidence in the truth or falseness of a statement, often expressed as a probability close to 1 or 0. Uncertainty, however, recognizes a lack of definitiveness, with probabilities spread across a range between 0 and 1, indicating differing degrees of belief in the accuracy of a statement. So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

2.2. Probabilistic reasoning

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as: "It will rain today," "behavior of someone for some situations," "A match between two

teams or two players". These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

2.3. Probability Basics

Probability is a fundamental concept in statistics and mathematics that quantifies uncertainty. It allows us to make informed decisions in situations where outcomes are uncertain. Here's a more detailed explanation:

2.3.1. Probability

Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

We can find the probability of an uncertain event by using the below formula:

$$\text{probability of occurrence} = \frac{\text{Number of favorable outcomes}}{\text{Total number of possible outcomes}}$$

Example

Let's consider the rolling of a fair six-sided die. The event A is rolling a 4. The formula for probability is:

$$\mathrm{P(A)} = \frac{\text{Number of favorable outcomes for rolling a 4}}{\text{Total number of possible outcomes}}$$

In this case, there is only one way to roll a 4 (favorable outcome), and there are six possible outcomes (rolling a 1, 2, 3, 4, 5, or 6).

$$P(rolling\ \ 4) = \frac{1}{6}$$

So, the probability of rolling a 4 on a fair six-sided die is: 1/6, or approximately 0.167.

2.3.2. Probability Concepts

- **Sample Space (Ω):** The sample space is the set of all possible outcomes of a random experiment. For example, when rolling a fair six-sided die, the sample space is Ω = {1, 2, 3, 4, 5, 6}.
- **Event (A):** An event is a subset of the sample space, representing a specific outcome or a collection of outcomes. For example, "rolling an even number" can be represented as A = {2, 4, 6}.

- **Complement of an Event (A'):** The complement of an event A, denoted as A', represents all outcomes that are not in A. In the example of rolling an even number, A' would represent rolling an odd number.
- **Random Experiments:** A random experiment is a process or procedure that leads to one of several possible outcomes. These outcomes are not predictable with certainty, and the result of the experiment is determined by chance. Examples of random experiments include rolling a die, flipping a coin, or conducting a scientific experiment with uncertain outcomes.
- **Random variables:** Random variables are used to represent the possible outcomes of a random event. It can take on different values with associated probabilities. Random variables can be discrete (e.g., the outcome of a dice roll) or continuous (e.g., the temperature in a given location).

2.3.3. Marginal Probability

Marginal probability, denoted as P(A), refer to the probability of a single event A (or random variable) occurring on its own, without considering any other variables. For example, if you're rolling a fair six-sided die, the margin probability of rolling a 4 would be 1/6 (P (4) = 1/6), as there is one favorable outcome (rolling a 4) out of six possible outcomes.

2.3.4. Joint Probability

The joint probability, denoted as P(A and B) or P(A∩B), is the probability of both events A and B occurring simultaneously. In the context of two events, A and B, the formula for joint probability is given by the product of the individual probabilities of each event:

$$P(A \text{ and } B) = P(A \cap B) = P(A) \times P(B)$$

For instance, if you want to find the joint probability of flipping a coin and rolling a die and getting both a "heads" (H) and a "4", it would be: P(H and 4) = P(H) * P(4) = (1/2) * (1/6) = 1/12.

The general formula for the joint probability of more than two events occurring simultaneously is an extension of the formula for two events. For three events A, B, and C, the joint probability is given by:

$$P(A \cap B \cap C) = P(A) \times P(B|A) \times P(C|A \cap B)$$

In this formula:

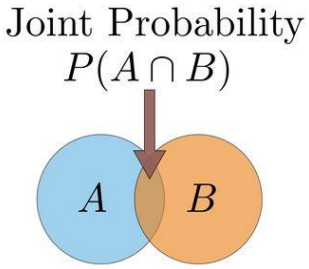
- P(A) is the probability of event A.
- P(B|A) is the conditional probability of event B given that event A has occurred.
- P(C|A∩B) is the conditional probability of event C given that events A and B have occurred.

For more than three events, the pattern continues:

$$P(A1 \cap A2 \cap \ldots \cap An) = P(A1) \times P(A2|A1) \times P(A3|A1 \cap A2) \times \ldots \times P(An|A1 \cap A2 \cap \ldots \cap An-1)$$

Each term in the product represents the **conditional probability** of the current event given that all the previous events have occurred.

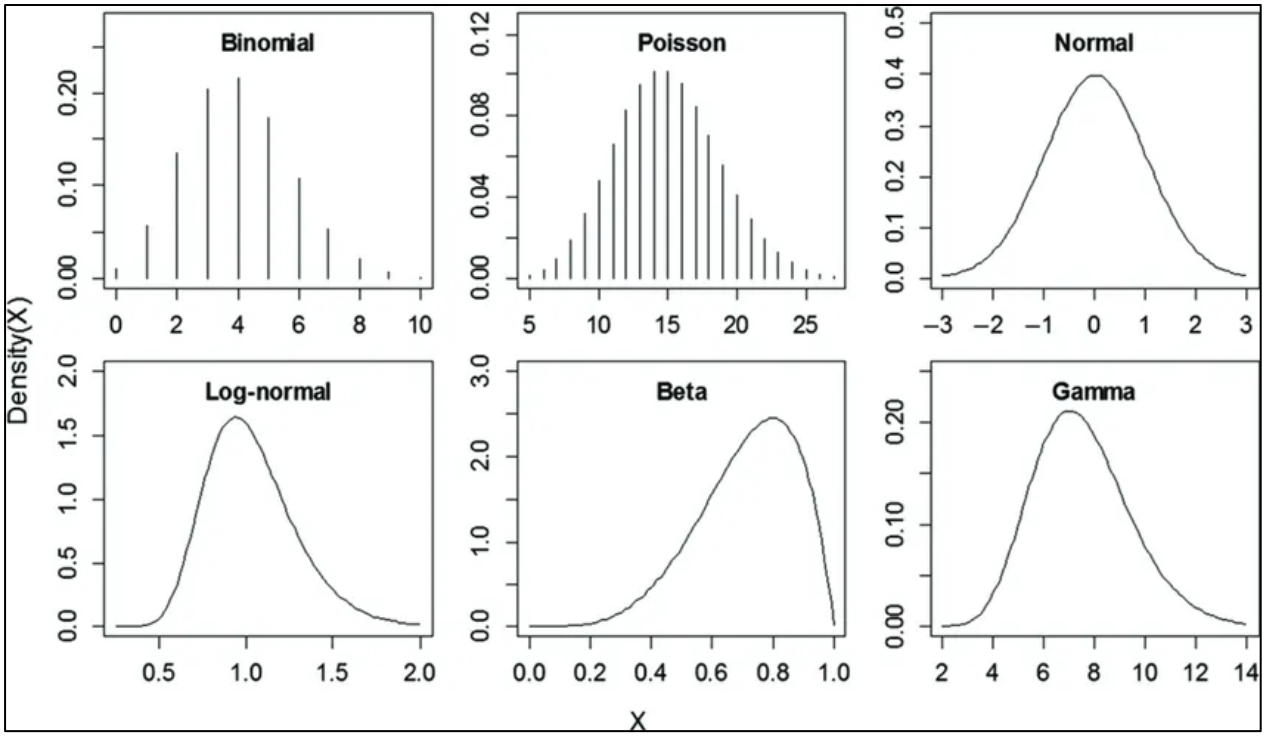
It's important to note that the events must be mutually exclusive for this formula to hold. If the events are not independent, the formula may involve more complex conditional probabilities.



2.3.5. Probability Distribution

A probability distribution describes how the probabilities of possible outcomes are spread or distributed over a set of events or random variables. Probability distributions can be continuous (like the probability density function of a normally distributed random variable) or discrete (like the probability mass function of a coin flip).

Common probability distributions include the uniform, binomial, normal (Gaussian), and Poisson distributions. Each distribution serves to model different types of random phenomena in various fields. The figure below illustrates some common probability distributions:



To illustrate, let's examine a discrete probability distribution representing the number of heads when flipping a fair coin twice. The probability mass function (PMF) for this scenario is denoted as f(x)=P(X=x), where X represents the number of heads in two flips. The specific probabilities for obtaining 0, 1, or 2 heads are as follows:

- For X=0, there is only one way to get 0 heads (both flips are tails): $\frac{1}{2}\times\frac{1}{2}=\frac{1}{4}$
- For X=1, there are two ways to get 1 head (HT or TH): $2\times\frac{1}{2}\times\frac{1}{2}=\frac{1}{2}$
- For X=2, there is only one way to get 2 heads (both flips are heads): $\frac{1}{2}\times\frac{1}{2}=\frac{1}{4}$.

So, the probability distribution is: $(0)=\frac{1}{4}$ $f(1)=\frac{1}{2}$ $(2)=\frac{1}{4}$. As expected, the sum of these probabilities is $\frac{1}{4}+\frac{1}{2}+\frac{1}{4}=1$, which ensures that all possible outcomes are covered.

2.3.6. Total Probability Law

The Law of Total Probability is a theorem that allows you to find the probability of an event by considering all possible ways that the event can occur. It is often expressed as follows: If B_1, B_2,…, B_n form a partition of the sample space S, then for any event A, the law of total probability states that:

$$P(A) = \sum_{i=1}^{n} P(A|B_i) * P(B_i)$$

In this formula:

- P(A) is the marginal probability of event A.
- P(A|Bi) is the conditional probability of event A given that event Bi has occurred.
- P(Bi) is the probability of event Bi occurring.
- n is the number of different scenarios or conditions.

2.3.7. Chain rule

In probability, the chain rule helps find the joint probability of two variables, X and Y. It's expressed as P(X, Y) = P(X|Y) * P(Y).

$$P(X, Y) = P(X|Y) * P(Y).$$

Extending this to three random variables, say X, Y, and Z, the joint probability P(X, Y, Z) can be calculated as the product of the conditional probability of X given both Y and Z (P(X|Y, Z)), the conditional probability of Y given Z (P(Y|Z)), and the probability of Z (P(Z)), as depicted in the equation:

$$P(X, Y, Z) = P(X|Y, Z) * P(Y|Z) * P(Z).$$

In general, for n random variables, you can use the chain rule to express the joint probability as a product of conditional probabilities and marginal probabilities:

$$P(X_1, X_2, ..., X_n) = P(X1|X_2, X_3, ..., X_n) * P(X_2|X_3, ..., X_n) * ... * P(X_{n-1}|X_n) * P(X_n).$$

2.4.Conditional Probability

Conditional probability is the probability of an event occurring given that another event has already occurred. It is denoted as P(A|B), where A is the event of interest, and B is the conditioning event. The formula for conditional probability is:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

- **P(A|B):** Conditional probability of event A given event B.
- **P(A ∩ B):** Probability of both events A and B occurring (joint probability).
- **P(B):** Probability of event B occurring.

Example:

Let's imagine a bag with 4 red marbles and 2 green marbles. If we want to find the conditional probability of picking a green marble on the second draw given that the first marble drawn was red, we can use the formula:

$$P(\text{Green on 2nd draw} \mid \text{Red on 1st draw}) = \frac{P(\text{Red on 1st draw and Green on 2nd draw})}{P(\text{Red on 1st draw})}$$

Here:

- P(Red on 1st draw) is the probability of drawing a red marble first, which is $\frac{4}{6}$ since there are 4 red marbles out of a total of 6 marbles initially.
- P(Red on 1st draw and Green on 2nd draw) is the probability of drawing a red marble first and a green marble second. After drawing a red marble, there are now 5 marbles left, and 2 of them are green. So, this probability is $\frac{4}{6} \times \frac{2}{5}$.

Now, we can calculate the conditional probability:

$$P(\text{Green on 2nd draw} \mid \text{Red on 1st draw}) = \frac{\frac{4}{6} \times \frac{2}{5}}{\frac{4}{6}} = \frac{\frac{4}{15}}{\frac{4}{6}} = \frac{6}{15} = 0.4$$

The conditional probability of picking a green marble on the second draw given that the first marble drawn was red = 40%.

2.4.1. Independent Events

Independence means that the occurrence of one event does not affect the likelihood or outcome of another. When two events are independent, their joint probability is the product of their individual probabilities (P(A and B) = P(A) * P(B)).

For example, when rolling a die, the outcome of one roll is independent of the outcome of any other roll. The probability of getting a 4 on one roll doesn't change based on the results of previous or subsequent rolls. If you flip a coin twice and roll a die, these events are typically independent. The probability of getting "heads" twice and rolling a "4" would be:

P(H and H and 4) = P(H) * P(H) * P(4) = (1/2) * (1/2) * (1/6) = 1/24.

2.4.2. Dependence Events

In probability, dependence events signify the connection between two or more events. When the occurrence of one event influences the likelihood or outcome of another, these events are

classified as dependent. Mathematically, if A and B are dependent events, the conditional probability of A given B is expressed as:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

In the previously discussed example of drawing marbles, when a red marble is drawn on the first draw, the likelihood of drawing a green marble on the second draw is influenced by the fact that one red marble has already been drawn.

These events exhibit dependence because the outcome of the initial draw, involving a red marble, alters the composition of the remaining marbles in the bag. This alteration in composition subsequently impacts the probability of the second event, specifically the probability of drawing a green marble.

This example serves as a demonstration of dependence in probability, where the occurrence of one event significantly influences the probability of another event due to the non-replacement of marbles after each draw.

2.4.3. Conditional Probability of Specific Event Given a Multiple Events

The conditional probability of specific event given multiple events can be expressed using the concept of conditional probability. Consider events A_1, A_2, …, A_n where n is the number of events. The conditional probability of event A_k given that all the previous events A_1, A_2,…, A_{k-1} have occurred is given by:

$$\mathrm{P(Ak \mid A1 \cap A2 \cap \ldots \cap Ak - 1)} = \frac{\mathrm{P(A1 \cap A2 \cap \ldots \cap Ak)}}{\mathrm{P(A1 \cap A2 \cap \ldots \cap Ak - 1)}}$$

In this formula:

- $P(A_1 \cap A_2 \cap \ldots \cap A_k)$ is the joint probability of all events $A_1, A_2, \ldots, A_k$ occurring together.
- $P(A_1 \cap A_2 \cap \ldots \cap A_{k-1})$ is the joint probability of events $A_1, A_2, \ldots, A_{k-1}$ occurring together.
- $P(A_k \mid A_1 \cap A_2 \cap \ldots \cap A_{k-1})$ is the conditional probability of event A_k given that all the previous events $A_1, A_2, \ldots, A_{k-1}$ have occurred.

This formula can be extended for any number of events. The general pattern is to express the conditional probability of each event as the ratio of the joint probability of all events up to that point to the joint probability of all events up to the previous event.

2.4.4. Conditional Probability of Multiple Events Given a Specific Event

The conditional probability of many events given an event can be expressed using the concept of conditional probability. If you have events A_1, A_2, …, A_n and you want to find the conditional probability of these events given another event B, you can use the formula:

$$P(A1 \cap A2 \cap \ldots \cap An \mid B) = \frac{P(A1 \cap A2 \cap \ldots \cap An \cap B)}{P(B)}$$

In this formula:

- $P(A_1 \cap A_2 \cap \ldots \cap A_n|B)$ is the joint probability of events A_1, A_2, …, A_n and B occurring together.
- P(B) is the probability of event B occurring.

2.4.5. Manipulating Joint Probabilities with Dependent Events

When manipulating joint probabilities for dependent events, you may also encounter the formula for the joint probability of two events, which can be expressed as:

$$P(A \cap B) = P(A|B) \times P(B)$$

For more than two dependent events (A, B, and C):

$$P(A \cap B \cap C) = P(A|B \cap C) \times P(B|C) \times P(C)$$

In a general form for n events:

$$P(A1 \cap A2 \cap \ldots \cap An) = P(A1|A2 \cap A3 \cap \ldots \cap An) \times P(A2|A3 \cap A4 \cap \ldots \cap An) \times \ldots \times P(An-1|An) \times P(An)$$

Each term in the product represents the conditional probability of the current event given that all the previous events have occurred.

When manipulating these probabilities, it's essential to have information about the dependencies between events and to adjust the conditional probabilities accordingly based on the given conditions

3. Bayes' Theorem

3.1.Overview

Bayes' theorem is also known as *Bayes' rule*, *Bayes' law*, or *Bayesian reasoning*, which determines the probability of an event based on prior knowledge of the conditions that might be relevant to the event.

The theorem is named after the British mathematician Thomas Bayes. Bayesian inference is an application of this theorem, playing a fundamental role in Bayesian statistics. It enables us to calculate the value of P(B|A) based on what we know about P(A|B).

Bayes' theorem allows us to adjust our probability predictions when we acquire new real-world information. For instance, if we want to assess the likelihood of someone having cancer based on their age, we can use Bayes' theorem to make a more accurate estimation. Bayes' theorem can be derived using product rule and conditional probability of event A with known event B. As from product rule we can write:

$$P(A \wedge B) = P(A|B)\, P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$P(A \wedge B) = P(B|A)\, P(A)$$

Equating right hand side of both the equations, we will get:

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)}$$

The equation mentioned above is commonly referred to as **Bayes' rule** or **Bayes' theorem**. This equation forms the foundation of most modern AI systems for handling probabilistic reasoning. It illustrates the straightforward connection between joint and conditional probabilities. In this context:

- P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis (event) *A* when we have occurred an evidence *B*.
- P(B|A) is called the **likelihood**, in which we consider that hypothesis is true, then we calculate the probability of evidence.
- P(A) is called the **prior probability**, probability of hypothesis before considering the evidence
- P(B) is called **marginal probability**, pure probability of an evidence.

3.2. Illustrative Example: Standard 52-card deck

In a standard deck of 52 cards, there are 13 cards of each suit, namely clubs (♣), diamonds (♦), hearts (♥) and spades (♠). Each suit has cards that go like this: 2 through 10, Jack, Queen, King and Ace. Face cards are the cards which have pictures on them instead of numbers (Jack, Queen and King cards).

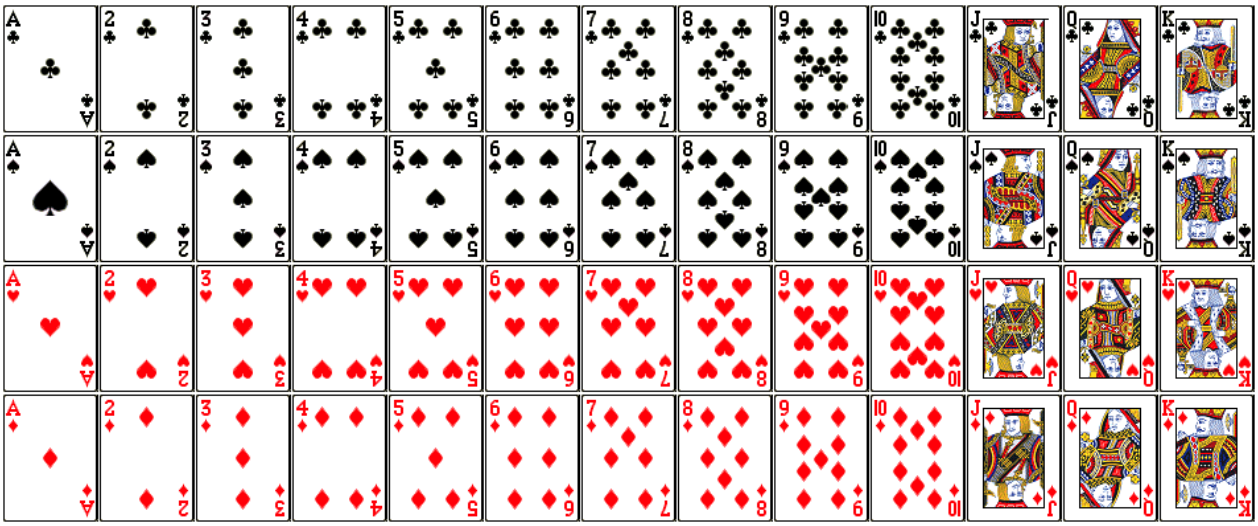


Figure: Standard 52-card deck

Questions:

1. Suppose you're holding a standard deck of cards and you're interested in finding the probability of drawing an Ace (event A) when you know that the card drawn is a red card (event B).
2. You draw a single card from a standard deck of playing cards. Now, calculate the posterior probability of P(King|Face), which represents the probability that the drawn face card is specifically a king.

Solution:

1. Here's how you can apply Bayes' theorem:

I. Define the events:

- Event A: Drawing an Ace.
- Event B: Drawing a red card (hearts or diamonds).

II. Find the individual probabilities:

- P(A): The probability of drawing an Ace. There are four Aces in a deck, so P(A) = 4/52 or 1/13.
- P(B): The probability of drawing a red card. There are 26 red cards in a deck (13 hearts and 13 diamonds), so P(B) = 26/52 or 1/2.

III. Determine the conditional probabilities:

- P(A|B): The probability of drawing an Ace given that it's a red card.
- P(B|A): The probability of drawing a red card given that it's an Ace.

Now, let's apply Bayes' theorem to find P(A|B), the probability of drawing an Ace given that it's a red card:

Bayes' theorem states: P(A|B) = [P(B|A) * P(A)] / P(B)

- P(B|A): The probability of drawing a red card given that it's an Ace is 2/4 because there are two red Aces (Ace of Hearts and Ace of Diamonds) among the four Aces.
- P(A): We previously calculated P(A) as 1/13.
- P(B): We also calculated P(B) as 1/2.

Now plug these values into Bayes' theorem: P(A|B) = [(2/4) * (1/13)] / (1/2) = 1/13

So, the probability of drawing an Ace given that it's a red card is **1/13**.

This example illustrates how Bayes' theorem allows you to update your probability estimate based on new information (in this case, the color of the card).

2. $P(king|face) = \frac{P(face|king)*P(king)}{P(face)}$

- P(king): probability that the card is King= 4/52= 1/13
- P(face): probability that a card is a face card= 3/13
- P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$P(king|face) = \frac{1*\left(\frac{1}{13}\right)}{\left(\frac{3}{13}\right)} = 1/3$, it is a probability that a face card is a king card

4. Bayesian Networks

4.1.Overview

A Bayesian network (BN) or belief networks, is a graphical model that represents probabilistic relationships among a set of variables. It is named after the Reverend Thomas Bayes, who introduced Bayes' theorem. The structure of a Bayesian network is represented by a directed acyclic graph (DAG), where nodes represent variables and edges represent probabilistic dependencies between variables[8].

The key components of a Bayesian network are:

- **Nodes (Random Variables):**

  Nodes represent random variables, which can be discrete or continuous. Each node in the network corresponds to a unique aspect of the modeled problem.

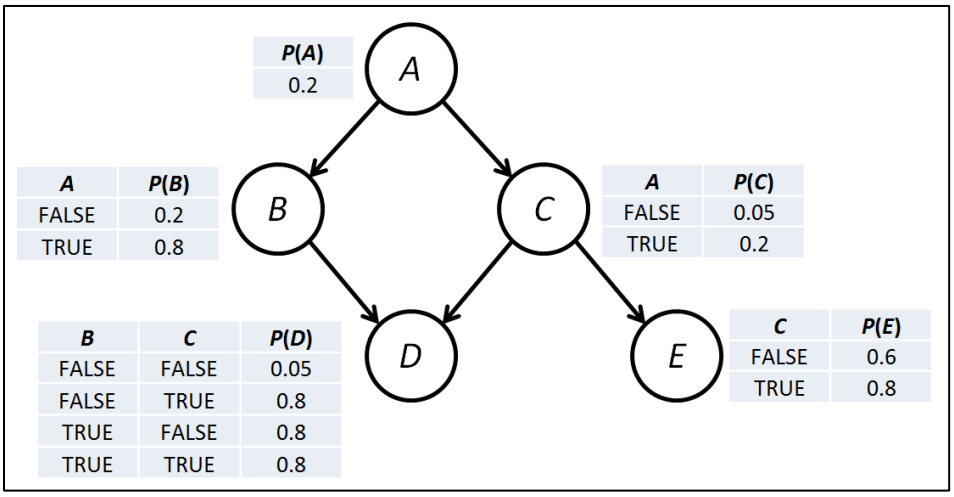
- **Directed Edges (Dependencies):**

Directed edges connect nodes and represent probabilistic dependencies. An edge from node A to node B indicates that B depends on A. These dependencies can be causal or simply associative.

- **Conditional Probability Tables (CPTs):**

  Each node has an associated CPT that specifies the conditional probability distribution of that node given the values of its parent nodes. CPTs quantify how a variable depends on its parents.

  The conditional distributions for each node are given as conditional probabilities table or CPT. Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.



Bayesian networks are widely used in various fields, including artificial intelligence, machine learning, medical diagnosis, and expert systems. They provide a flexible framework for modeling uncertainty and reasoning under uncertainty.

4.2.Inference in Bayesian Networks

Inference in Bayesian networks refers to the process of making predictions or drawing conclusions about the values of certain variables in the network given observed evidence or data. There are two main types of inference in Bayesian networks:

- **Marginal Inference:** Involves computing the marginal probability distribution of one or more variables in the network. This is often done using the sum-product algorithm (also known as variable elimination).

- **Conditional Inference:** Involves computing the conditional probability distribution of one or more variables given the observed evidence. This is often done using Bayes' theorem and updating the probabilities based on the observed data.

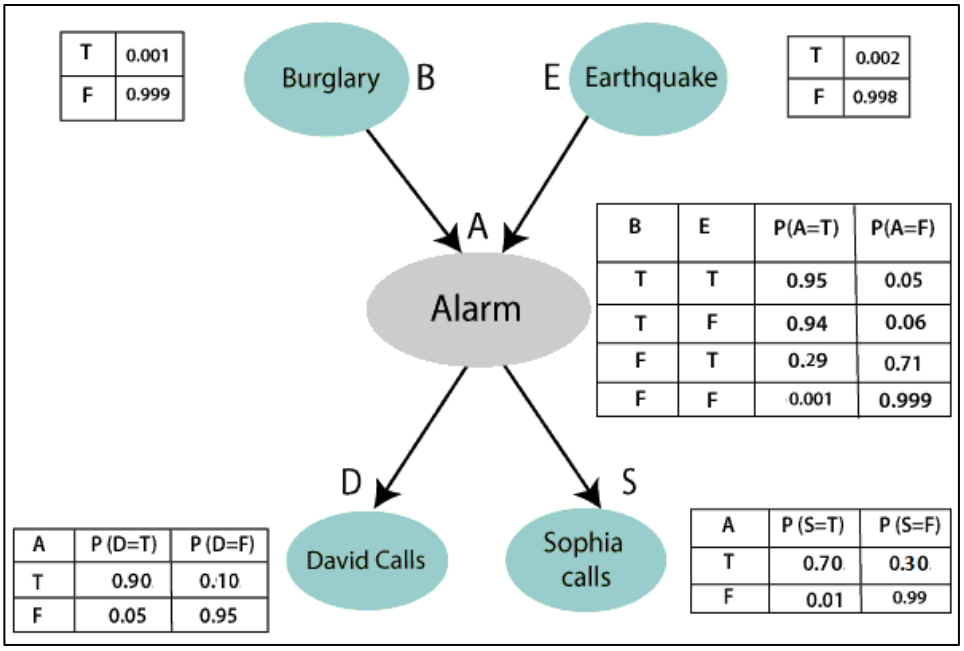
Bayesian Network Inference involves the following steps:

- **Observations or Evidence:** Specify the values of observed variables.
- **Propagation:** Use the probabilistic dependencies in the Bayesian network to update the probabilities of other variables.
- **Inference:** Compute the desired probabilities or predictions based on the observed evidence and the network structure.

4.3. Illustrative example: burglar alarm

Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm.

The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.



Question:

1. What is the probability that the alarm has sounded but neither a burglary nor earthquake has occurred, and both David and Sophia call?
2. What is the probability that David call?
3. What is the probability David call Sophia don't call , the alarm is going off but there is no body in the house and there is no earthquake

5. Conclusion

In summary, Bayesian networks, including belief networks and probabilistic graphical models, are powerful tools for handling uncertainty in complex systems. This chapter delved into the fundamental principles, exploring their graphical representation and methods for probabilistic inference. By focusing on key concepts like probability and conditional probability, the goal was to build a strong understanding of how Bayesian networks work and their practical uses in different situations. As we wrap up our exploration, it's clear that these tools not only provide a structured way to model complex relationships but also offer valuable insights for making informed decisions in uncertain situations.

Chapter 5:

Hidden Markov model

1. Introduction

The chapter explores Hidden Markov Models (HMMs), building on the foundational concepts of stochastic processes and Markov chains. Stochastic processes lay the groundwork for understanding randomness in systems, while Markov chains focus on state transitions. HMMs add complexity by introducing hidden states influencing observable outcomes. The chapter delves into essential elements of HMMs, explaining their structure and some of their key algorithms. This concise exploration provides a fundamental understanding of HMMs and their applications in modelling dynamic systems.

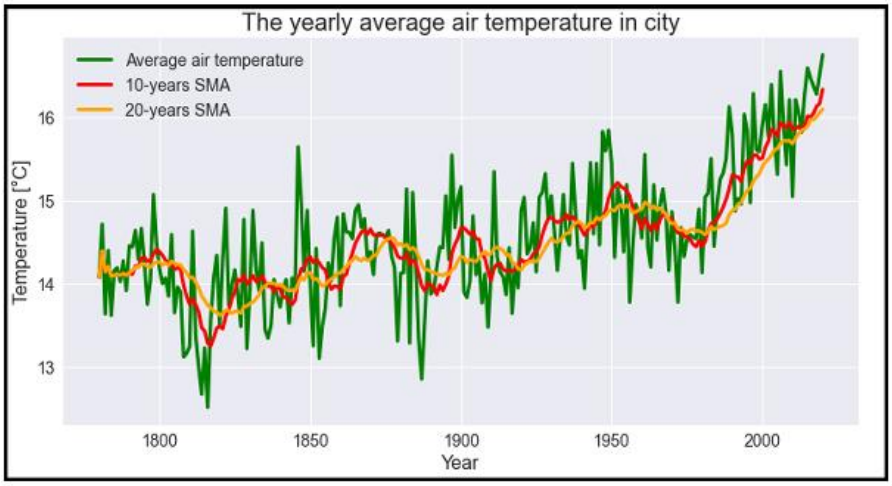
2. Stochastic Process

2.1. Definition

A stochastic process is a mathematical model that describes the evolution of random variables over time. These random variables can represent various phenomena such as stock prices, weather conditions, or the state of a system. In a stochastic process, the outcome at each time step is not deterministic but is governed by probability. . Mathematically, it's defined as $\{X(t), t \in T\}$, where $X(t)$ represents the state of the system at time t, and T is the index set, often representing discrete or continuous time.

2.2. Example of a Stochastic Process

The yearly average air temperature in a city is a stochastic process, denoted as $\{Xt:t \in T\}$, where Xt represents the random variable for the average temperature in year t. This stochastic modeling acknowledges the inherent randomness and variability in temperature, allowing for probabilistic predictions and the quantification of uncertainties associated with climate patterns over time.



Let's denote the temperature on year t as X(t). The joint distribution of temperatures over three consecutive years (t1, t2, t3) can be expressed as P(X(t1), X(t2), X(t3)). This distribution reflects the probabilities of observing specific temperature combinations over the three years, taking into account the historical data and climatic conditions.

2.3. Classification of Stochastic Processes

Stochastic processes can be classified into several categories based on their properties and characteristics:

- **Discrete-Time vs. Continuous-Time:** Stochastic processes can be defined for discrete time points (e.g., daily stock prices) or continuous time (e.g., continuous trajectories of particles in physics).
- **Stationary vs. Non-Stationary:** A stationary process has statistical properties that do not change over time, while a non-stationary process may exhibit changing statistics.
- **Markov vs. Non-Markov:** Markov processes have the Markov property, meaning the future state depends only on the current state, not the entire history. Non-Markov processes depend on past states.
- **Ergodic vs. Non-Ergodic:** An ergodic process is one where the time average and ensemble average of properties of the process are the same. In non-ergodic processes, this equivalence does not hold.
- **Homogeneous vs. Non-Homogeneous:** In a homogeneous process, the statistical properties remain constant over time, while in a non-homogeneous process, these properties change with time.

3. Markov Models

A Markov model, also known as a Markov process, represents a stochastic process for randomly changing systems where it is believed that future states do not depend on past states. These models show all possible states as well as the transitions, rate of transitions, and probabilities between them.

Markov models are frequently used to model the probabilities of various states and the rates of transitions among them. The method is generally used to model systems. Markov models can also be used to identify patterns, make predictions, and learn the statistics of sequential data.

A Markov process is a process that is capable of being in more than one state, can make transitions among those states, and in which the states available and transition probabilities depend only upon what state the system is currently in. In other words, there is no memory in a Markov process.

There are four types of Markov models that are used situationally:

- **Markov chain:** used by systems that are autonomous and have fully observable states
- **Hidden Markov model:** used by systems that are autonomous where the state is partially observable.
- **Markov decision processes:** used by controlled systems with a fully observable state.
- **Partially observable Markov decision processes:** used by controlled systems where the state is partially observable.

Markov models can be manifested in equations or in graphical models. Graphic Markov models typically use circles (each containing states) and directional arrows to show potential transitional changes between them. The directional arrows are labelled with the rate or the variable one for the rate.

Applications of Markov modelling include modelling languages, natural language processing (NLP), image processing, bioinformatics, speech recognition, and modelling computer hardware and software systems.

4. Markov chain

4.1. Overview

A Markov chain [9] is a mathematical model used to describe systems or processes that exhibit a sequence of events or transitions from one state to another, where the future state depends only on the current state and is independent of previous states.

A Markov chain is a stochastic process without memory, implying that the system's future behaviour is solely determined by its current state. This feature makes Markov chains especially valuable for modelling diverse real-world scenarios, including weather patterns, financial markets, and biological processes.

The diagram below illustrates a two-state Markov chain labeled E and A. Each number on the diagram represents the probability of the Markov process transitioning from one state to another, with the arrow indicating the direction. For instance, if the process is in state A, there's a 0.4 probability of transitioning to state E, and a 0.6 probability of staying in state A.

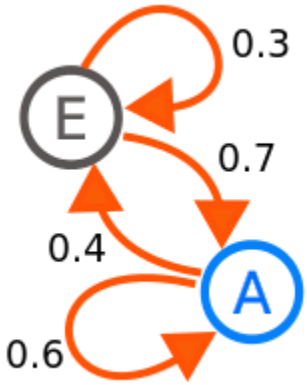


Figure: Markov chain

4.2. Key Concepts

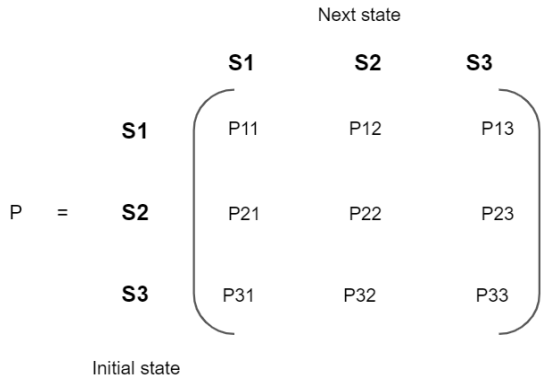
- **States & State Space:** a Markov chain consists of a finite or countable set of states X0, X1, X2,… . These states represent different conditions or configurations of the system. The **state** of a Markov chain is the value of **Xt** at time **t**. For example, if **Xt = 6**, we say the process is in state 6 at time t.

  The state space of a Markov chain, often denoted as S = {S_1, S_2, S_3, ...}, is the set of values that each **Xt** can take. For example, S = {1, 2, 3, 4, 5, 6, 7}.

- A **trajectory** of a Markov chain is a particular set of values for X0, X1, X2, ...
  For example, if X0 = 1, X1 = 5, and X2 = 6, then the trajectory up to time t = 2 is 1, 5, 6. More generally, if we refer to the trajectory s0, s1, s2, s3, . . . , we mean that X0 = s0, X1 = s1, X2 = s2, X3 = s3, . . . 'Trajectory' is just a word meaning 'path'.
  - The **transition probability** is the probability of moving from one state of a system into another state. If a Markov chain is in state i, the transition probability, pij, is the probability of going into state j at the next time step. Mathematically:

$$\mathbf{pij = P(next\ state\ Sj\ at\ t=1 \mid initial\ state\ Si\ at\ t=0)}$$

  Where i and j are initial state and next state, respectively.
- **Transition Probability Matrix** (State Transition Matrix): The transition probabilities are typically organized into a **Transition Probability Matrix**, denoted as P, where P(i, j) represents the probability of transitioning from state S_i to state S_j. The matrix P is defined as:



  Here, each row sums up to 1 since they are probabilities.
- **Markov Property:** The basic property of a Markov chain is that only the most recent point in the trajectory affects what happens next. This is called the Markov Property. It means that **Xt+1** depends only upon **Xt**, and it does not depend upon **Xt−1, . . . , X1, X0**. We formulate the Markov Property in mathematical notation as follows:

$$\mathbf{P(Xt+1 = s \mid Xt = st\ , Xt-1 = st-1, \ldots , X0 = s0) = P(Xt+1 = s \mid Xt = st),}$$

  For all t = 1, 2, 3, . . . and for all states s0, s1, . . . , st , s.

Explanation:

$$\mathbb{P}(X_{t+1} = s \mid X_t = s_t, \cancel{X_{t-1} = s_{t-1}, X_{t-2} = s_{t-2}, \ldots, X_1 = s_1, X_0 = s_0})$$

- $\mathbb{P}(X_{t+1} = s)$: *distribution of* X_{t+1}
- $X_t = s_t$: *depends on* X_t
- $\cancel{X_{t-1} = s_{t-1}, X_{t-2} = s_{t-2}, \ldots, X_1 = s_1, X_0 = s_0}$: *but whatever happened before time* t *doesn't matter.*

- **Time Homogeneity:** Markov chains often assume time homogeneity, meaning that the transition probabilities remain constant over time. This implies that the probability of transitioning from one state to another does not change as time progresses.

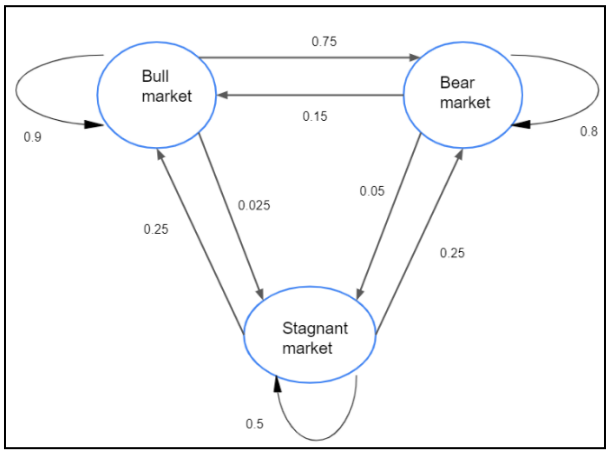
4.3. Illustrative Example: Markov chains to predict stock market trends

We want to model the stock markets trend. So first recall our assumption that a stocks market movement is random. Therefore there is a dynamic system we want to examine the stock markets trend. We can identify that this system transitions randomly**,** between a set numbers of states. All possible collections of all these states are called *state-space*. For our case, we can identify that a stock markets movement can only take on three states (the state-space):

- **Bull markets:** periods of time where prices generally are rising, due to the actors having optimistic hopes of the future.
- **Bear markets:** periods of time where prices generally are declining, due to the actors having a pessimistic view of the future.
- **Stagnant markets:** periods of time where the market is characterized by neither a decline nor a rise in general prices.

In fair markets, it is expected that the market information is distributed equally among its actors and that prices fluctuate randomly. This means that every actor has equal access to information such that no actor has an upper hand due to inside information. Through technical analysis of historical data, certain patterns can be found as well as their estimated probabilities.

After a week characterized by a bull market trend, there is a 90% chance that another bullish week will follow. Additionally, there is a 7.5% chance that the bull week instead will be followed by a bearish one or a 2.5% chance that it will be a stagnant one. After a bearish week, there's an 80% chance that the upcoming week also will be bearish, and so on as represented in the state transition diagram below:



By compiling these probabilities into a table, we get the following state transition matrix **M**:

To / *From*	Bull	Bear	Stagnant
Bull	0.9	0.075	0.025
Bear	0.15	0.8	0.05
Stagnant	0.25	0.25	0.5

$$= \begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix} = M$$

We then create a 1x3 vector C which contains information about which of the three different states any current week is in; where column 1 represents a bull week, column 2 a bear week, and column 3 a stagnant week. In this example we will choose to set the current week as bearish, resulting in the vector C = [0 1 0].

Given the state of the current week, we can then calculate the possibilities of a bull, bear, or stagnant week for any number of n weeks into the future. This is done by multiplying the vector C with the matrix, giving the following:

$$\text{One week from now: } C * M^1 = [0 \quad 1 \quad 0]\begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}^1 = [0.15 \quad 0.8 \quad 0.05]$$

$$\text{5 weeks from now: } C * M^5 = [0 \quad 1 \quad 0]\begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}^5 = [0.48 \quad 0.45 \quad 0.07]$$

$$\text{52 weeks from now: } C * M^{52} = [0 \quad 1 \quad 0]\begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}^{52} = [0.63 \quad 0.31 \quad 0.05]$$

$$\text{99 weeks from now: } C * M^{99} = [0 \quad 1 \quad 0]\begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}^{99} = [0.63 \quad 0.31 \quad 0.05]$$

A characteristic of what is called a regular Markov chain is that, over a large enough number of iterations, all transition probabilities will converge to a value and remain unchanged. This means that, after a sufficient number of iterations, the likelihood of ending up in any given state of the chain is the same, regardless of where you start.

So, when the chain reaches this point, we can say the transition probabilities reached a **steady-state**.

From this we can conclude that as n $\rightarrow \infty$, the probabilities will converge to a steady-state, meaning that 63% of all weeks will be bullish, 31% bearish, and 5% Stagnant. What we also see is that the steady-state probabilities of this Markov chain do not depend upon the initial state. The results can be used in various ways; some examples are calculating the average time it takes for a bearish period to end or the risk that a bullish market turns bearish or stagnant.

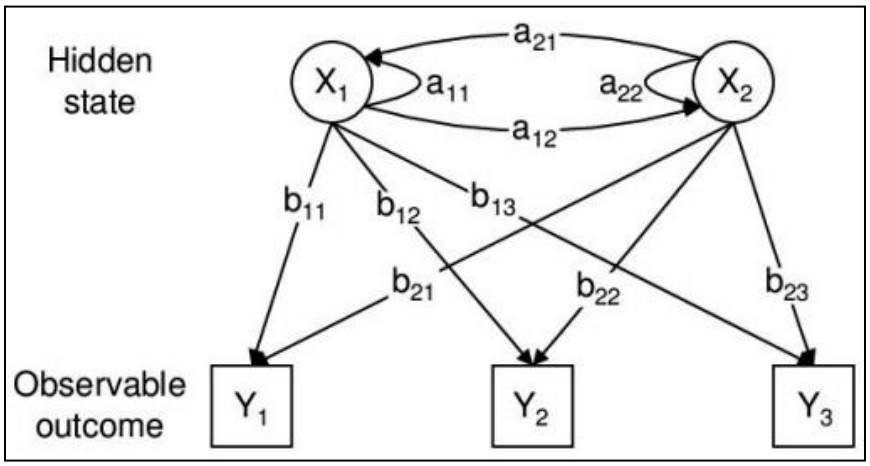
5. Hidden Markov Models

Hidden Markov Models (HMMs) are a dual stochastic system consisting of an unobservable stochastic process, which operates as a concealed Markov chain transitioning between states, and an observable process influenced by these hidden states. The primary aim of HMMs is to unveil concealed information within the observables [9].

HMMs comprise two key components: Hidden states (X) and Observable symbols (Y). These components are governed by essential probabilities:

- Transition probabilities (a): the probability of moving from one state to another state

- Emission probabilities (b): the probability of an observation being generated from a state.
- Initial probability distribution (X_i) for the starting state of the Markov chain.



5.3. Applications of HMCs

Hidden Markov Chains are used in various applications:

- **Speech Recognition:** Modelling phoneme transitions as hidden states and observed audio signals as observable states.
- **Natural Language Processing:** Part-of-speech tagging (hidden states) based on observed words (observable states).
- **Finance:** Modelling financial market regimes (e.g., bull and bear markets) as hidden states and observed stock prices.
- **Bioinformatics:** Predicting gene sequences (hidden states) from observable DNA sequences.

Incorporating Bayesian networks and Hidden Markov Chains in your analysis can help you model complex systems, infer relationships, and make informed decisions under uncertainty. These techniques are foundational in various fields, including machine learning and artificial intelligence

5.4. Fundamental HMM Algorithms

In this section we will cover the cornerstone algorithms for understanding and using HMMs. These algorithms include the forward algorithm for probability estimation, the Viterbi algorithm for decoding the most likely state sequence, and the Baum-Welch algorithm for training HMMs.

5.4.1. Computing Sequence Probabilities: The Forward Algorithm

The Forward Algorithm is a fundamental dynamic programming algorithm used in Hidden Markov Models (HMMs) to compute the probability of observing a particular sequence of observations given the model. It is often used to solve problems like sequence likelihood estimation, where you want to calculate the probability of observing a specific sequence of observations. Here's a step-by-step explanation of the Forward Algorithm:

Inputs:

1. Hidden Markov Model (HMM) parameters:
   - Hidden States (S): The set of possible hidden states.
   - Observations (O): The set of possible observations.
   - Initial State Probabilities (π): The probabilities of starting in each hidden state.
   - Transition Probabilities (A): The probabilities of transitioning from one hidden state to another.
   - Emission Probabilities (B): The probabilities of emitting observations from each hidden state.
2. Sequence of observations (O_1, O_2, ..., O_x): The sequence of observed data you want to calculate the likelihood for.

Steps:

1. Initialization:

Initialize an array of forward probabilities (α) for each hidden state and the first observation. For each hidden state S_i:

$$\alpha(1, S_i) = \pi(S_i) * B(S_i, O_1)$$

Where: $\pi(S_i)$ is the initial state probability and $B(S_i, O_1)$ is the emission probability for the first observation.

2. Recursion:

For each time step t from 2 to T (T is the length of the observation sequence):

- Calculate the forward probabilities for each hidden state S_i at time t:

$$\alpha(t, S_i) = \sum_{all\ Sx} [\alpha(t-1, Sx) * A(Sx, S_i) * B(S_i, O_t)]$$

This calculation involves considering the probability of being in state S_x at time t-1 ($\alpha(t-1, S_x)$), transitioning from S_x to S_i ($A(S_x, S_i)$), and emitting observation O_t ($B(S_i, O_t)$). The sum is taken

over all possible states S_x, representing the contribution of each state to the probability of being in state Si at time t.

3. Termination:

The final probability is obtained by summing the forward probabilities for all hidden states at the last time step T:

$$P(\text{Observations}) = \sum \alpha(T, S_i), \text{ for all } S_i.$$

The result, P(Observations), represents the probability of observing the given sequence of observations under the HMM with its specific parameters.

5.4.2. Finding the Most Likely State Sequence: The Viterbi Algorithm

The algorithm is named after its inventor, Andrew Viterbi. The Viterbi Algorithm is a dynamic programming algorithm used in Hidden Markov Models (HMMs) to find the most likely sequence of hidden states given a sequence of observations. It is widely used in various applications, including speech recognition, natural language processing, bioinformatics, and more. Here's a step-by-step explanation of the Viterbi Algorithm:

Inputs:

Hidden Markov Model (HMM) parameters:

- Hidden States (S): The set of possible hidden states.
- Observations (O): The set of possible observations.
- Initial State Probabilities (π): The probabilities of starting in each hidden state.
- Transition Probabilities (A): The probabilities of transitioning from one hidden state to another.
- Emission Probabilities (B): The probabilities of emitting observations from each hidden state.

Steps:

1. Initialization:

Initialize the Viterbi probabilities (δ) for the first observation (O[1]) and the backpointer (ψ) for each state:

$$\delta(1, i) = \pi(i) * B(i, O[1]) \text{ for all states } i.$$

$\psi(1, i)$ is undefined at this point.

Here, $\delta(1, i)$ represents the probability of the most likely path that ends in state i at time step 1, and $\pi(i)$ is the initial probability of being in state i.

2. Recursion:

Calculate the Viterbi probabilities and backpointers for the remaining observations (O[2] to O[T]) for each state and each time step:

For each time step t from 2 to T:

For each state j:

Calculate $\delta(t, j)$ and $\psi(t, j)$ as follows:

$$\delta(t, j) = \max[\delta(t-1, i) * A(i, j) * B(j, O[t])], \text{ where i is a state in S.}$$

$$\psi(t, j) = \text{argmax}[\delta(t-1, i) * A(i, j)], \text{ where i is a state in S.}$$

Here, $\delta(t, j)$ represents the probability of the most likely path that ends in state j at time step t, and $\psi(t, j)$ stores the state from the previous time step that maximizes the probability.

3. Termination:

Calculate the overall probability of the most likely path and the final state in that path:

$$P^*(O) = \max[\delta(T, i)], \text{ where i is a state in S.}$$

The most likely final state is the state j that maximizes $\delta(T, j)$.

4. Backtracking:

Recover the most likely sequence of hidden states by backtracking through the backpointers. Start from the final state j* found in Step 3 and trace back through the ψ values to find the most likely path of hidden states.

The sequence of states obtained through backtracking represents the most likely sequence of hidden states that best explains the given sequence of observations.

5.4.3. Training HMM Parameters: The Baum-Welch Algorithm

The Baum-Welch Algorithm or Forward-Backward algorithm is an Expectation-Maximization (EM) algorithm used for training an HMM when the model parameters are unknown.

5.5. Illustrative example: Weather Prediction with Hidden Markov Models

In this example, we'll use a basic two-state HMM for weather prediction. Suppose you want to predict the weather on a given day, but you can't directly observe the weather. Instead, you can only see whether your friend is carrying an umbrella (which is your observation). You want to use an HMM to make predictions.

HMM Setup:

- Hidden States: Sunny (S), Rainy (R)

- Observations: Umbrella (U), No Umbrella (N)

Model Parameters:

Initial State Probabilities (π):

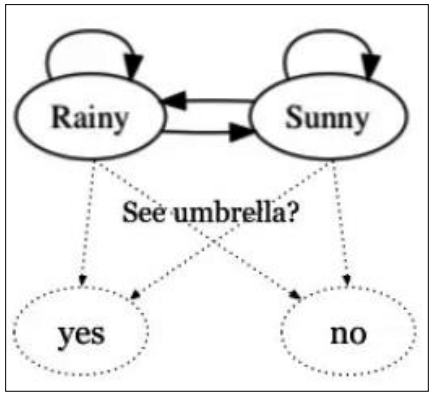
- P(S) = 0.7 (Initial state is sunny with a 70% chance).
- P(R) = 0.3 (Initial state is rainy with a 30% chance).

Transition Probabilities (A):

- P(S | S) = 0.8 (80% chance of staying sunny).
- P(R | S) = 0.2 (20% chance of transitioning from sunny to rainy).
- P(S | R) = 0.4 (40% chance of transitioning from rainy to sunny).
- P(R | R) = 0.6 (60% chance of staying rainy).

Emission Probabilities (B):

- P(U | S) = 0.2 (20% chance of carrying an umbrella when it's sunny).
- P(N | S) = 0.8 (80% chance of not carrying an umbrella when it's sunny).
- P(U | R) = 0.6 (60% chance of carrying an umbrella when it's rainy).
- P(N | R) = 0.4 (40% chance of not carrying an umbrella when it's rainy).



Questions:

1. What is the probability of observing the sequence U, U, N, U using the Forward Algorithm?
2. What is the most likely sequence of hidden states for the observations U, U, N, U using the Viterbi Algorithm

Solutions:

1. Using the Forward Algorithm:

Step 1: Initialization: Initialize the forward probabilities (α) for the first observation (U) for both the Sunny and Rainy states.

- $\alpha(1, S) = P(U \mid S) * P(S) = 0.2 * 0.7 = 0.14$
- $\alpha(1, R) = P(U \mid R) * P(R) = 0.6 * 0.3 = 0.18$

Step 2: Recursion: Calculate the forward probabilities for the remaining observations (U, N, U) for both states by considering the previous probabilities and the transition probabilities.

- $\alpha(2, S) = [\alpha(1, S) * P(S \mid S) * P(U \mid S)] + [\alpha(1, R) * P(S \mid R) * P(U \mid S)] = [0.14 * 0.8 * 0.2] + [0.18 * 0.4 * 0.2] = 0.0224 + 0.0144 = 0.0368$
- $\alpha(2, R) = [\alpha(1, S) * P(R \mid S) * P(U \mid R)] + [\alpha(1, R) * P(R \mid R) * P(U \mid R)] = [0.14 * 0.2 * 0.6] + [0.18 * 0.6 * 0.6] = 0.0168 + 0.0648 = 0.0816$
- $\alpha(3, S) = [\alpha(2, S) * P(S \mid S) * P(N \mid S)] + [\alpha(2, R) * P(S \mid R) * P(N \mid S)] = [0.0368 * 0.8 * 0.8] + [0.0816 * 0.4 * 0.8] = 0.023552 + 0.025984 = 0.049536$
- $\alpha(3, R) = [\alpha(2, S) * P(R \mid S) * P(N \mid R)] + [\alpha(2, R) * P(R \mid R) * P(N \mid R)] = [0.0368 * 0.2 * 0.4] + [0.0816 * 0.6 * 0.4] = 0.002944 + 0.019584 = 0.022528$
- $\alpha(4, S) = [\alpha(3, S) * P(S \mid S) * P(U \mid S)] + [\alpha(3, R) * P(S \mid R) * P(U \mid S)] = [0.049536 * 0.8 * 0.2] + [0.022528 * 0.4 * 0.2] = 0.007936 + 0.00180576 = 0.00974176$
- $\alpha(4, R) = [\alpha(3, S) * P(R \mid S) * P(U \mid R)] + [\alpha(3, R) * P(R \mid R) * P(U \mid R)] = [0.049536 * 0.2 * 0.6] + [0.022528 * 0.6 * 0.6] = 0.00593632 + 0.00814464 = 0.01408196$

Step 3: Termination: Calculate the overall probability of observing the entire sequence U, U, N, U by summing up the forward probabilities at the last time step (T).

$$P(\text{Observations}) = \alpha(4, S) + \alpha(4, R) = 0.00974176 + 0.01408196 = 0.02382372$$

So, the probability of the sequence U, U, N, U is approximately 0.0238, given the HMM and its parameters.

2. Using the Viterbi Algorithm

The most likely sequence of hidden states (Sunny or Rainy) for the observations U, U, N, U using the Viterbi Algorithm

Step 1: Initialization: Initialize the Viterbi probabilities (δ) for the first observation (U) and the backpointer (ψ) for both states.

- $\delta(1, S) = P(U \mid S) * P(S) = 0.2 * 0.7 = 0.14$

- $\delta(1, R) = P(U \mid R) * P(R) = 0.6 * 0.3 = 0.18$

No need for a backpointer at this step.

Step 2: Recursion: Calculate the Viterbi probabilities and backpointers for the remaining observations (U, N, U) for both states.

For observation U at time step 2:

- $\delta(2, S) = \max[\delta(1, S) * P(S \mid S) * P(U \mid S), \delta(1, R) * P(S \mid R) * P(U \mid S)] = \max[0.14 * 0.8 * 0.2, 0.18 * 0.4 * 0.2] = \max[0.0224, 0.0144] = 0.0224$ (maximum from S)
- $\psi(2, S) = \text{argmax}[\delta(1, S) * P(S \mid S) * P(U \mid S), \delta(1, R) * P(S \mid R) * P(U \mid S)] = S$
- $\delta(2, R) = \max[\delta(1, S) * P(R \mid S) * P(U \mid R), \delta(1, R) * P(R \mid R) * P(U \mid R)] = \max[0.14 * 0.2 * 0.6, 0.18 * 0.6 * 0.6] = \max[0.0168, 0.0648] = 0.0648$ (maximum from R)
- $\psi(2, R) = \text{argmax}[\delta(1, S) * P(R \mid S) * P(U \mid R), \delta(1, R) * P(R \mid R) * P(U \mid R)] = R$

For observation N at time step 3:

- $\delta(3, S) = \max[\delta(2, S) * P(S \mid S) * P(N \mid S), \delta(2, R) * P(S \mid R) * P(N \mid S)] = \max[0.0224 * 0.8 * 0.8, 0.0648 * 0.4 * 0.8] = \max[0.014336, 0.020736] = 0.020736$ (maximum from R)
- $\psi(3, S) = \text{argmax}[\delta(2, S) * P(S \mid S) * P(N \mid S), \delta(2, R) * P(S \mid R) * P(N \mid S)] = R$
- $\delta(3, R) = \max[\delta(2, S) * P(R \mid S) * P(N \mid R), \delta(2, R) * P(R \mid R) * P(N \mid R)] = \max[0.0224 * 0.2 * 0.4, 0.0648 * 0.6 * 0.4] = \max[0.001792, 0.019584] = 0.019584$ (maximum from R)
- $\psi(3, R) = \text{argmax}[\delta(2, S) * P(R \mid S) * P(N \mid R), \delta(2, R) * P(R \mid R) * P(N \mid R)] = R$

For observation U at time step 4:

- $\delta(4, S) = \max[\delta(3, S) * P(S \mid S) * P(U \mid S), \delta(3, R) * P(S \mid R) * P(U \mid S)] = \max[0.020736 * 0.8 * 0.2, 0.019584 * 0.4 * 0.2] = \max[0.00331776, 0.00156672] = 0.00331776$ (maximum from S)
- $\psi(4, S) = \text{argmax}[\delta(3, S) * P(S \mid S) * P(U \mid S), \delta(3, R) * P(S \mid R) * P(U \mid S)] = S$
- $\delta(4, R) = \max[\delta(3, S) * P(R \mid S) * P(U \mid R), \delta(3, R) * P(R \mid R) * P(U \mid R)] = \max[0.020736 * 0.2 * 0.6, 0.019584 * 0.6 * 0.6] = \max[0.002496, 0.00669312] = 0.00669312$ (maximum from R)
- $\psi(4, R) = \text{argmax}[\delta(3, S) * P(R \mid S) * P(U \mid R), \delta(3, R) * P(R \mid R) * P(U \mid R)] = R$

Step 3: Termination: Calculate the overall maximum probability for the last time step (T) and determine the most likely final state (j*):

  - $\delta(4, S) = 0.00331776$
  - $\delta(4, R) = 0.00669312$

The maximum probability at time step 4 is obtained from state R (Rainy):

- $j^* = \text{argmax}[\delta(4, j)] = R$

Step 4: Backtracking:

Now, we can proceed with the backtracking step to find the most likely sequence of hidden states, starting with the most likely final state j^* (R) and following the backpointers backward to time step 1. Here's the complete most likely state sequence:

1. Time Step 4: $j^* = R$ (Rainy)
2. Time Step 3: $\psi(3, R) = R$ (Rainy)
3. Time Step 2: $\psi(2, R) = R$ (Rainy)
4. Time Step 1: $\psi(1, R) = R$ (Rainy)

So, the most likely sequence of hidden states for the observations U, U, N, U, obtained using the Viterbi Algorithm, is R, R, R, R, corresponding to Rainy weather throughout the observed sequence

6. Conclusion

In conclusion, this course introduced Hidden Markov Models (HMMs), taking us through the basics of random processes and Markov chains. We've dived into the workings of HMMs, learning about their structure and important algorithms. Now, armed with this knowledge, you're ready to use HMMs to understand and model dynamic systems with hidden parts and observable results.

Chapter 6:

Applications of ML in communication networks

1. Introduction

Machine Learning (ML) finds diverse applications in communication networks, significantly enhancing efficiency, performance, and security. This chapter will explore key applications that collectively contribute to the evolution of more intelligent, adaptive, and efficient communication networks, addressing the challenges posed by the increasing complexity and dynamic nature of modern telecommunications.

2. Network Optimization

Machine Learning (ML) plays a crucial role in optimizing communication networks by employing algorithms and models to enhance performance, efficiency, and overall functionality.

2.1. Traffic engineering

One fundamental aspect is traffic engineering, where ML algorithms analyze historical and real-time data to discern traffic patterns. This enables predictive models to anticipate peak usage times and dynamically adjust routing, preventing congestion and ultimately reducing latency for improved network performance.

2.2. Resource allocation

Resource allocation is another key area where ML proves beneficial. Predictive models forecast resource demands by considering historical usage patterns and current network conditions. Through dynamic resource allocation, ML ensures that network resources such as bandwidth and processing power are efficiently distributed to meet demand. This adaptability enhances network responsiveness, minimizing underutilization or overloading of resources.

2.3. Quality of Service (QoS) Optimization

Quality of Service (QoS) optimization is crucial for ensuring a consistent and high-quality user experience. ML algorithms continually monitor and analyze QoS parameters, dynamically adjusting them to maintain optimal performance. This is particularly important for real-time applications like video conferencing and online gaming, where a seamless user experience is paramount.

2.4. Anomaly Detection and Fault Prevention

Anomaly detection and fault prevention are enhanced through ML. By identifying abnormal patterns in network behavior, ML models can detect potential faults or security threats early on. This proactive approach allows for timely interventions, preventing network failures and improving overall reliability. ML can also learn from historical data to recognize patterns associated with network issues, triggering alerts or automated responses.

2.5. Dynamic Spectrum Management (DSM)

In wireless communication networks, ML aids in dynamic spectrum management (DSM). By analyzing factors like interference, signal strength, and user demand, ML algorithms dynamically adjust spectrum allocation to maximize efficiency. This is particularly relevant in the context of emerging technologies like 5G, where spectrum management is intricate.

2.6. Autonomous Network Management

Autonomous network management is facilitated by ML, enabling networks to self-configure, self-heal, and self-optimize. Automated decision-making based on ML analysis reduces the need for manual intervention, making networks more adaptive to changing conditions.

2.7. Capacity Planning and Scalability

Capacity planning and scalability benefit from ML as well. ML models analyze historical data to predict future capacity requirements, allowing for proactive planning for network upgrades and expansions. This ensures that networks remain scalable to accommodate growing demands.

2.8. Energy Efficiency

Lastly, ML contributes to energy efficiency in communication networks. Algorithms optimize the power consumption of network components by predicting and adjusting energy usage based on demand and traffic patterns. This leads to more sustainable and cost-effective network operations. In summary, ML in network optimization provides a data-driven and adaptive approach, contributing to the creation of more resilient, efficient, and intelligent communication networks

3. 5G and Beyond

The "5G and Beyond" section represents a pivotal domain where Machine Learning (ML) is instrumental in shaping the future of communication networks. As the telecommunications industry transitions to 5G and explores beyond, ML plays a multifaceted role in optimizing various aspects of these advanced networks.

3.1. Network Slicing Optimization:

One of the distinctive features of 5G is network slicing, allowing the creation of virtualized, isolated network segments tailored for specific applications. ML contributes by optimizing the allocation of resources for each slice based on real-time demand, ensuring efficient and dynamic network utilization.

3.2. Beamforming Optimization:

Beamforming is a key technology in 5G for directing signals to specific users, enhancing network efficiency and throughput. ML algorithms analyze environmental conditions, user locations, and historical data to optimize beamforming parameters dynamically, adapting to changing network conditions for improved signal quality.

3.3. Predictive Maintenance for Millimeter Wave (mmWave) Networks:

5G relies heavily on high-frequency mmWave bands, which are susceptible to environmental conditions. ML models can predict potential issues in mmWave networks by analyzing historical data, enabling proactive maintenance to address challenges such as signal blockage due to obstacles or weather conditions.

3.4. Dynamic Spectrum Sharing:

ML facilitates dynamic spectrum sharing, a crucial aspect of 5G networks. By continuously analyzing spectrum usage patterns and interference levels, ML algorithms optimize the sharing of available spectrum among different services and users, ensuring efficient and interference-free communication.

3.5. Ultra-Reliable Low Latency Communication (URLLC):

5G aims to support URLLC, requiring ultra-low latency and high reliability for applications like autonomous vehicles and critical infrastructure. ML contributes by dynamically managing

network resources to meet stringent latency requirements, predicting and preventing potential bottlenecks.

3.6. Massive Machine Type Communication (mMTC):

For the massive deployment of IoT devices in 5G networks, ML aids in optimizing communication between a vast number of devices. Predictive models can anticipate communication patterns, adapt to varying device densities, and efficiently manage the network to accommodate the diverse requirements of IoT applications.

3.7. Security and Threat Detection:

With the increased complexity of 5G networks, ML is crucial for security. ML models can analyze network traffic patterns to detect anomalies indicative of cyber threats. Additionally, ML contributes to securing the various components of 5G networks, including the core network, edge computing, and device authentication.

3.8. Energy-Efficient 5G Networks:

ML plays a role in optimizing the energy consumption of 5G networks, particularly in scenarios where massive MIMO (Multiple Input, Multiple Output) and other energy-intensive technologies are employed. ML models can predict and adjust energy usage based on traffic patterns, contributing to sustainable and eco-friendly network operations.

In essence, as communication networks evolve into the realm of 5G and beyond, the integration of Machine Learning becomes indispensable. ML not only optimizes specific technological aspects but also ensures that these advanced networks are adaptive, secure, energy-efficient, and capable of supporting a diverse range of applications with varying requirements. The symbiotic relationship between ML and advanced communication technologies is poised to redefine the possibilities of connectivity and communication in the coming years.

4. Conclusion

In summary, Machine Learning plays a crucial role in improving the efficiency, performance, and security of communication networks. This chapter has explored key applications contributing to the development of more intelligent and adaptive networks, effectively addressing the challenges posed by the dynamic nature of modern telecommunications. Looking ahead, the integration of

Machine Learning principles promises continued advancements and enhanced capabilities in communication technolog

Bibliography

[1] J. Finlay, *An introduction to artificial intelligence*: Crc Press, 2020.

[2] K. Chowdhary, *Fundamentals of artificial intelligence*: Springer, 2020.

[3] J. P. Mueller and L. Massaron, *Machine learning for dummies*: John Wiley & Sons, 2021.

[4] L. B. Almeida, "Multilayer perceptrons," in *Handbook of Neural Computation*, ed: CRC Press, 2020, pp. C1. 2: 1-C1. 2: 30.

[5] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, pp. 579-588, 2009.

[6] J. P. Mueller and L. Massaron, *Deep Learning for dummies*: John Wiley & Sons, 2019.

[7] F. V. Jensen and T. D. Nielsen, *Bayesian networks and decision graphs* vol. 2: Springer, 2007.

[8] T. Koski and J. Noble, *Bayesian networks: an introduction* vol. 924: John Wiley & Sons, 2011.

[9] P. Dymarski, *Hidden Markov models: theory and applications*: BoD–Books on Demand, 2011.