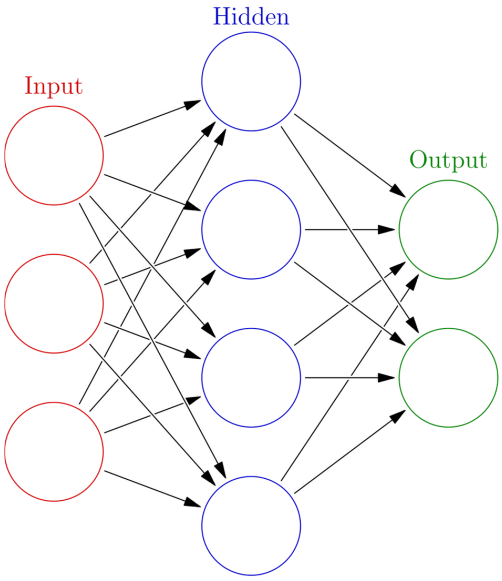
Machine Learning Mental Model

Machine Learning

- Table of Contents
  - General
  - Models
  - Training
- General:
  - AI [Machine Learning[Deep Learning]]
    - In practical terms the AI I outline here is centered around deep learning (neural networks with hidden layers) which is a subset of machine learning which is a subject of AI. This is where the bulk of the action is, so I will focus on machine learning and neural networks
  - (Problem → Model → Training → Testing) Development Loop
    - Problem
      - We are always trying to solve some problem in a better way. So before we begin utilizing any technique we need to know the problem that we want to solve.
      - In machine learning, there is the following set of problems types you will encounter and can solve with different techniques.
      - Classification Problems:
        - Deep learning is commonly used for tasks where the goal is to assign categories or labels to input data.
        - This includes:
          - image classification
          - text classification
          - speech recognition
      - Generative Problems:
        - Deep learning is commonly used for tasks where the goal is to output a media of communication (text, speech, video, etc) to input data.
        - This includes:
          - image generation
          - text generation
          - speech generation
      - Regression Problems (Prediction):
        - Deep learning is commonly used for tasks where the goal is predict what will happen next based on historical data.
      - Decision Making Problems (Agent):
        - Deep learning is commonly for decision making agents.
    - Model
      - The mathematical model that will take data inputs, do data transformations, and output data.
      - See Models section for in depth Breakdown.
    - Training
      - Train the Model on data.
      - See Training Section for in depth breakdown.
    - Testing
      - Once the neural network has been trained, you can use it for various purposes. For instance:
        - In classification tasks, you can input new data, and the trained network will predict which category it belongs to.

        - In regression tasks, you can input data, and the network will provide a continuous output.
        - In generation tasks, the network can create new data outputs based on the patterns it has learned.
        - In decision-making tasks, the network can take actions based on the learned policies.
    - Test with new data to see if your model has learned to generalize based on it's training step.
- Models
    - Table of Contents
        - General
        - NN Model Architecture
        - Model Types Analysis
    - General
        - A "Model" refers to a mathematical representation or an algorithm that learns patterns and relationships from data.
        - (Input → Data Transformation → Output) Machines = Model:
            - Models are primarily input-output systems. They take input data, process it through a set of mathematical operations (such as transformations, aggregations, and non-linear mappings), and produce an output or prediction based on the learned patterns from the input data.
            - The input data can take various forms depending on the problem and the type of model being used. For example:
                - In image classification, the input data could be an image represented as a matrix of pixel values.
                - In natural language processing, the input data could be a sequence of words or characters.
                - In speech recognition, the input data could be an audio waveform.
            - Similarly, the output data can vary based on the task and problem being addressed:
                - In classification tasks, the output may be a label or category that the model assigns to the input data.
                - In regression tasks, the output may be a continuous value, such as predicting a numeric quantity like house prices or temperature.
                - In generative tasks, the output may be a newly generated data point, such as a new image or a piece of text.
            - The key aspect of machine learning models is their ability to generalize from the training data to make predictions or produce outputs for new, unseen data. This generalization ability is crucial for their practical utility and success in real-world applications.
            - While models in machine learning are typically input-output systems, the internal workings of complex models like deep neural networks can be difficult to interpret and understand fully. Despite their black-box nature, these models can learn intricate patterns and relationships from data, enabling them to make accurate predictions and perform complex tasks across various domains.
        - Examples of Models:
            - Neural Network Models:
                - The most common is the Neural Network architecture. See Neural Network Architecture section for deeper dive into neural network models.
                - I focus on Neural Networks in this chapter, as they seem to be most useful thus far.
            - Linear Models:
                - These models assume a linear relationship between the input features and the target variable. Examples include linear regression for regression tasks and logistic regression for binary classification tasks.
            - Decision Trees:
                - Decision trees are tree-like structures where each node represents a feature, and branches represent possible feature values. These models are often used for classification and regression tasks.
            - Support Vector Machines (SVM):
                - SVM is a powerful classification algorithm that finds an optimal hyperplane to separate different classes in the data.
            - Ensemble Methods:

        - Ensemble methods combine multiple weak learners to create a stronger model. Examples include Random Forest (an ensemble of decision trees) and Gradient Boosting Machines (GBMs).
    - Naive Bayes:
        - Naive Bayes is a probabilistic model based on Bayes' theorem and is commonly used for text classification and spam filtering.
    - K-Nearest Neighbors (KNN):
        - KNN is a simple algorithm that classifies new data points based on the majority class of their k-nearest neighbors.
    - Clustering Models:
        - Clustering models group data points based on similarity, such as K-Means clustering and Hierarchical clustering.
    - Factorization Machines:
        - Factorization machines are used for tasks like recommendation systems and can handle interactions between features effectively.
    - Rule-based Models:
        - These models use explicit rules to make decisions, such as expert systems or rule-based classifiers.
- NN Model Architecture
    - Hyperparameters:
        - Hyperparameters are external settings that control the model's behavior during training.
        - Examples of hyperparameters include:
            - The learning rate (step size for weight updates).
            - The number of hidden layers and neurons in a neural network.
            - The type of activation functions used.
            - The regularization strength.
            - The batch size used during training.
        - Finding the optimal values for hyperparameters is often a manual or automated tuning process called hyperparameter tuning or hyperparameter optimization.
    - 

    - Input Layer:
        - The input layer is the initial part of the model where data is fed into the model. The number of nodes (neurons) in the input layer depends on the dimensionality of the input data.
        - Input Features:
            - Features refer to the individual input variables or characteristics of the data that are used as input to the model. These features are the building blocks of the input data and represent different attributes or properties of the data samples.

- Features can be numerical values, categorical variables, or even more complex representations like images, audio waveforms, or text sequences. The goal of a neural network is to learn patterns and relationships in the data by processing these features through its layers and adjusting its internal parameters (weights and biases) during training.
- For example:
  i. In an image classification task, features could be the pixel values of an image, and the neural network learns to recognize patterns in the pixels to classify the image into different categories (e.g., cat, dog, bird).
  ii. In natural language processing, features might be word embeddings or one-hot encoded vectors representing words in a sentence. The neural network learns to understand the context and relationships between words to perform tasks like sentiment analysis, language translation, or text generation.
  iii. In a time series prediction task, features could be historical data points over time, and the neural network learns to recognize patterns and trends in the time series to make predictions about future values.
  iv. In speech recognition, features might be audio spectrograms or Mel-frequency cepstral coefficients (MFCCs), and the neural network learns to process these representations to recognize spoken words or transcribe speech.
- The selection and engineering of appropriate features are essential for the success of a neural network in any given task. The quality and relevance of the features directly influence the model's ability to capture the underlying patterns and make accurate predictions. In some cases, neural networks can also learn to extract meaningful features directly from raw data through the use of convolutional layers (for images) or recurrent layers (for sequences).
- Feature engineering is a crucial step in the machine learning workflow, and it involves selecting, transforming, or creating features that best represent the data and are suitable for the specific problem at hand. Well-chosen and informative features can significantly improve the performance of a neural network and enable it to tackle complex tasks effectively.

- Hidden Layers:
  - In many AI models, there are one or more hidden layers between the input and output layers.
  - Each hidden layer contains a set of neurons that process the data and learn representations of the underlying patterns.
  - The hidden layer makes a model a deep learning model.
- Output Layer:
  - The output layer is the final layer of the model that produces the model's predictions or decisions. The number of nodes in the output layer depends on the nature of the problem, such as the number of classes for a classification task or the number of output values for a regression task.
- Activation Functions:
  - Definition:
    - Fire or Not: The activation function determines whether a neuron should "fire" (activate) or remain "silent" (inactive) based on the input it receives.
    - Activation functions are applied to the output of neurons in the hidden layers.
    - The activation function operates on the weighted sum of the inputs to a neuron (also known as the activation), producing the neuron's output.
    - Mathematically, the activation function takes the weighted sum of inputs (z) and applies a transformation, denoted as a function f(z), to produce the output of the neuron: Output = f(z)

      Typically, the weighted sum (z) is computed as follows:

      z = (w1 * input1) + (w2 * input2) + ... + (wn * inputn) + bias

      where w1, w2, ..., wn are the weights associated with each input, and the bias is an additional parameter that shifts the activation function's output.
  - Non Linearity:
    - Activation functions introduce non-linearity into the model, allowing it to learn and approximate complex relationships between the input data and the output predictions.
  - Commonly Used Activation Functions:

        - The choice of activation function is essential because it can significantly impact the model's learning and performance. Each activation function has its advantages and disadvantages, and the choice of the appropriate activation function depends on the specific problem and the characteristics of the data. It's common to use ReLU or its variants in most layers of modern deep neural networks due to their simplicity and better performance in many scenarios.
        - Sigmoid Function:
            - The sigmoid activation function maps the input to a range between 0 and 1. It was historically popular but has fallen out of favor in hidden layers of deep neural networks due to vanishing gradient issues.
        - Hyperbolic Tangent (tanh) Function:
            - The tanh activation function maps the input to a range between -1 and 1. Like the sigmoid, it can suffer from vanishing gradients in deep networks.
        - Rectified Linear Unit (ReLU):
            - The ReLU activation function sets all negative inputs to zero and leaves positive inputs unchanged. It is widely used in hidden layers of deep neural networks due to its simplicity and ability to alleviate vanishing gradient problems.
        - Leaky ReLU:
            - Leaky ReLU is a modification of ReLU that allows a small negative slope for negative inputs. This helps avoid the "dying ReLU" problem, where neurons can get stuck with zero gradients during training.
        - Parametric ReLU (PReLU):
            - PReLU extends Leaky ReLU by making the negative slope a learnable parameter during training.
        - Exponential Linear Unit (ELU):
            - ELU is a variant of ReLU that allows negative inputs to have a smoother mapping, which can help improve learning speed.
        - Swish:
            - Swish is a recently proposed activation function that offers a smooth curve with better performance compared to ReLU in some cases.
- Parameters
    - Parameters = Weights and Biases
        - Parameters are the learnable elements of a model that the algorithm tunes during the training process. Weights and Biases.
        - During training, the model adjusts these parameters to minimize the difference between its predictions and the true target values (as defined by the loss function).
        - The process of adjusting the parameters to optimize the model's performance is usually done through gradient descent and backpropagation.
    - Weights:
        - Connections between neurons in adjacent layers are defined by weights, which represent the strength of the connection. During training, the model updates these weights to learn from the input data and improve its performance.
    - Bias:
        - Bias is an additional parameter used in neural networks that allows the model to make predictions even when the input features are all zero.
        - A bias term is present in each neuron (except the input layer) of the neural network, and it acts as an offset that shifts the output of the neuron.
        - Like weights, biases are also adjusted during the training process to improve the model's accuracy.
        - Including bias in the model makes it more flexible and capable of learning complex relationships, as it allows the model to learn shifts and offsets in the data.
        - Output = Activation_Function(Weighted_Sum_of_Input + Bias)
- Loss Function:
    - Loss Function Minimization = Objective of a Model

        - The loss function quantifies the model's performance by measuring the difference between its predictions and the actual target values. The model's objective during training is to minimize the loss function.
        - A loss function, also known as a cost function or objective function, is a crucial component that measures how well the model's predictions match the true labels or target values. The loss function quantifies the difference between the predicted output of the neural network and the actual ground-truth values for a given set of input data. The goal during training is to minimize the value of the loss function, which indicates that the model's predictions are closer to the true labels.
    - Types of Loss Functions: The choice of the loss function depends on the type of task the neural network is performing. For example:
        - For regression tasks (predicting continuous values), the mean squared error (MSE) loss function is commonly used.
        - For binary classification tasks (two possible classes), the binary cross-entropy loss function is commonly used.
        - For multi-class classification tasks (more than two classes), the categorical cross-entropy loss function is commonly used.
- Optimization Algorithm:
    - Optimization Function = Search for Optimal Parameters:
        - The optimization algorithm is used to update the model's weights based on the gradients calculated from the loss function.
        - The optimization algorithms primary objective is to adjust the model's parameters (weights and biases) iteratively to minimize the loss function.
        - In other words, the optimization algorithm helps the neural network find the optimal set of parameters that result in the best possible predictions on the training data.
        - The optimization algorithm achieves this by updating the model's parameters in a way that reduces the difference between the predicted outputs and the true labels, as quantified by the loss function. The process involves taking small steps in the direction that decreases the loss, ultimately guiding the model toward better performance.
    - Common Optimization Algorithms:
        - Gradient-based optimization methods are commonly used in neural networks. These methods utilize the gradients of the loss function with respect to the model's parameters to determine the direction and magnitude of the parameter updates. The gradient represents the slope or rate of change of the loss function concerning each parameter, indicating the direction of steepest descent (where the loss decreases most rapidly).
        - These optimization algorithms help the neural network navigate the high-dimensional parameter space to find the optimal set of parameters that minimize the loss function, leading to a well-trained model that can make accurate predictions on new, unseen data.
        - The choice of the optimization algorithm can significantly affect the training speed and final performance of the neural network.
        - The most widely used optimization algorithms in neural networks include:
            i. Stochastic Gradient Descent (SGD): SGD is the simplest optimization algorithm and updates the model's parameters after evaluating the loss on each individual data point (or a small batch of data points). It is computationally efficient but can be noisy and slow to converge to the optimal solution.
            ii. Mini-batch Gradient Descent: Mini-batch Gradient Descent is a compromise between SGD and full-batch gradient descent. It updates the parameters after evaluating the loss on a small batch of data points, typically ranging from 32 to 512 examples. Mini-batch gradient descent is widely used in practice as it provides a good balance between computational efficiency and convergence speed.
            iii. Adam (Adaptive Moment Estimation): Adam is a popular adaptive optimization algorithm that combines the benefits of both AdaGrad and RMSprop. It dynamically adjusts the learning rate for each parameter based on the historical gradients, making it well-suited for a wide range of neural network architectures and problems.
            iv. RMSprop (Root Mean Square Propagation): RMSprop modifies the learning rate for each parameter based on the moving average of the squared gradients. It adapts the learning rate to each parameter's update history, which can improve convergence on different scales of the loss landscape.
            v. Adagrad (Adaptive Gradient Algorithm): Adagrad adapts the learning rate for each parameter based on the historical gradients. It performs larger updates for infrequent parameters and smaller updates for frequent parameters, making it

well-suited for sparse data.

vi. Adadelta: Adadelta is an extension of Adagrad that addresses the issue of aggressive learning rate decay. It uses a moving average of the squared parameter updates to adapt the learning rate.

vii. Nadam (Nesterov-accelerated Adaptive Moment Estimation): Nadam combines the benefits of Adam and Nesterov Accelerated Gradient (NAG). It incorporates NAG's momentum technique along with the adaptive learning rate of Adam.

- Regularization:
  - Regularization techniques are used to prevent overfitting, where the model performs well on the training data but poorly on new data. Regularization helps to generalize the model's learning to unseen data.
  - Regularization in neural networks (NN) is a technique used to prevent overfitting and improve the generalization performance of the model. Overfitting occurs when the neural network becomes too specialized in the training data, memorizing noise and outliers rather than learning the underlying patterns. As a result, the model's performance on new, unseen data (validation or test data) may be poor.
  - Regularization methods introduce additional constraints or penalties to the neural network's loss function during training. These constraints discourage the model from becoming overly complex and help it focus on the most important features in the data. The regularization techniques aim to strike a balance between fitting the training data well while maintaining generalization to new data.
  - There are several common regularization techniques used in neural networks:
    a. L1 Regularization (Lasso Regression): L1 regularization adds a penalty to the loss function proportional to the absolute values of the model's weights. This encourages some of the weights to become exactly zero, effectively leading to feature selection and simplifying the model.
    b. L2 Regularization (Ridge Regression): L2 regularization adds a penalty to the loss function proportional to the square of the model's weights. This discourages large weight values and results in a more smooth, distributed weight configuration, making the model more robust to variations in the data.
    c. Dropout: Dropout is a popular regularization technique that involves randomly "dropping out" a fraction of neurons during each training iteration. This prevents specific neurons from relying too heavily on certain features and forces the network to learn more robust and general representations.
    d. Batch Normalization: Batch normalization is a technique that normalizes the activations of neurons within a layer, helping to stabilize and accelerate training. It acts as a form of regularization by reducing internal covariate shift, making the network more resistant to overfitting.
    e. Early Stopping: Early stopping is not a direct regularization method, but it is a technique used to prevent overfitting. It involves monitoring the performance of the model on a validation set during training and stopping the training process when the performance starts to degrade. This is done to avoid training the model for too long and potentially overfitting the training data.
    f. Data Augmentation: Data augmentation is another indirect regularization technique. It involves applying random transformations to the training data, such as rotation, translation, or flipping, to create additional samples. Data augmentation increases the diversity of the training set, helping the model generalize better to unseen data.
  - By applying regularization techniques, neural networks can achieve better generalization, reduce overfitting, and improve performance on unseen data. The choice of the appropriate regularization method depends on the specific problem, the model architecture, and the characteristics of the data.

- Model Types Analysis:
  - Neural Networks:
    - Pros:
      - Powerful at learning complex patterns from large datasets.
      - State-of-the-art performance in tasks like image recognition, NLP, and game playing.
      - Can handle unstructured data like images, audio, and text effectively.
    - Cons:
      - Requires large amounts of labeled training data.

        - Computationally intensive, especially for deep architectures.
        - Interpretability can be a challenge, making them appear as black-box models.
    - Tradeoffs:
        - High performance but may lack transparency and require significant resources.
    - Applications:
        - Image and speech recognition, natural language processing, game playing.
- Decision Trees:
    - Pros:
        - Easy to understand and interpret.
        - Handle both numerical and categorical data.
        - Can capture non-linear relationships.
    - Cons:
        - Prone to overfitting, especially for deep trees.
        - May not capture complex interactions well.
    - Tradeoffs:
        - Simple and interpretable but may not be as accurate as neural networks for some tasks.
    - Applications:
        - Classification and regression tasks, data mining.
- Support Vector Machines (SVM):
    - Pros:
        - Effective in high-dimensional spaces.
        - Robust against overfitting with proper kernel selection.
        - Versatile for both linear and non-linear problems.
    - Cons:
        - Can be sensitive to the choice of kernel function and parameters.
        - Computationally expensive for large datasets.
    - Tradeoffs:
        - Good performance for small to medium-sized datasets but may be slower on large datasets.
    - Applications:
        - Image and text classification, regression tasks.
- K-Nearest Neighbors (KNN):
    - Pros:
        - Simple and intuitive.
        - No training phase; the model is memory-based.
        - Can handle multi-class problems naturally.
    - Cons:
        - Computationally expensive during inference, especially for large datasets.
        - Sensitive to irrelevant features and outliers.
    - Tradeoffs:
        - Easy to implement but might not scale well to large datasets.
    - Applications:
        - Pattern recognition, collaborative filtering, recommendation systems.
- Naive Bayes:
    - Pros:

        - Simple and fast to train and classify.
        - Performs well with high-dimensional and sparse data.
        - Good for text classification tasks.
    - Cons:
        - Makes the assumption of independence between features (naive).
        - May not work well for more complex relationships in the data.
    - Tradeoffs:
        - Fast and efficient but relies on the naive independence assumption.
    - Applications:
        - Text classification, spam filtering.
- Random Forest:
    - Pros:
        - Ensemble of decision trees, providing improved accuracy and reduced overfitting.
        - Handles high-dimensional data well.
        - Robust to outliers and noise.
    - Cons:
        - Can be computationally expensive during training and inference.
        - Difficult to interpret compared to individual decision trees.
    - Tradeoffs:
        - Higher accuracy compared to individual decision trees but may lack interpretability.
    - Applications:
        - Classification and regression tasks, anomaly detection.
- Genetic Algorithms:
    - Pros:
        - Can find near-optimal solutions for complex optimization problems.
        - Suitable for problems with a large search space.
    - Cons:
        - Computationally expensive for complex problems.
        - Convergence can be slow for certain tasks.
    - Tradeoffs:
        - Powerful optimization technique but can be time-consuming.
    - Applications:
        - Optimization problems in engineering, finance, and game playing.
- Hidden Markov Models (HMM):
    - Pros:
        - Effective for modeling sequential data with probabilistic dependencies.
        - Used for speech recognition, language modeling, and bioinformatics.
    - Cons:
        - Assumes a fixed number of states and Markovian transitions.
        - Sensitive to the choice of initial parameters.
    - Tradeoffs:
        - Suitable for sequential data but may require parameter tuning.
    - Applications:
        - Speech recognition, part-of-speech tagging, time series prediction.

        - Rule-Based Systems:
            - Pros:
                - Transparent and interpretable.
                - Easy to create and understand rules for specific domains.
            - Cons:
                - May not handle complex interactions well.
                - Manual rule creation can be time-consuming and error-prone.
            - Tradeoffs:
                - Interpretable but may lack the ability to learn from data like neural networks.
            - Applications:
                - Expert systems, decision support systems.
- Training
    - Training = Teaching Model to Learn and Generalize Patterns in Data
        - Training a neural network (NN) model refers to the process of teaching the model to learn patterns and relationships from data. During training, the model adjusts its internal parameters (weights and biases) based on the provided data and corresponding labels to make accurate predictions. The goal is to minimize the error between the model's predictions and the true labels, allowing the model to generalize well to new, unseen data.
    - How Training Works under the hood: Here's a simplified mental model of how the training process works:
        - Initialize the Neural Network:
            - At the beginning of training, the neural network is like a blank slate with randomly initialized weights and biases. It doesn't know anything about handwritten digits yet.
        - Feed Forward:
            - To train the network, you show it a large set of labeled images (training data) of handwritten digits. The neural network takes an image as input and performs a series of computations through its layers (input, hidden, and output layers) to generate a prediction for the digit it thinks is present in the image.
        - Calculate Error:
            - The network's prediction may not be accurate initially because the weights are random. You compare the predicted digit to the true label (the correct digit) and calculate the error (also known as the loss or cost) between the prediction and the actual label. The error represents how far off the network's guess is from the correct answer.
        - Backpropagation:
            - The key to training a neural network is to adjust its weights and biases to reduce the error. This is done through a process called backpropagation. Backpropagation involves computing the gradients (derivatives) of the error with respect to the weights and biases in the network.
        - Update Weights: (via Optimization Algorithms most commonly Gradient Descent)
            - Using the gradients, you update the network's weights and biases slightly to minimize the error. This is like nudging the model in the right direction to make better predictions.
        - Repeat:
            - Steps 2 to 5 are repeated for many iterations (epochs) with different training examples. Each pass through the entire training dataset is one epoch. As training progresses, the model's predictions become more accurate as it fine-tunes its weights to recognize patterns in the data.
        - Validation:
            - During training, you also evaluate the model on a separate validation dataset to monitor its performance. This helps you detect overfitting (when the model becomes too specialized in the training data) and make adjustments if needed.
        - Testing:
            - Once training is complete, you can evaluate the final model on a test dataset to assess its performance on unseen data.

- Ways to Train a Neural Network: Training a neural network can be broadly categorized into three main approaches based on the type of learning:
  i. Supervised Learning:
    1. In supervised learning, the neural network is provided with input-output pairs (labeled data) during training.
    2. The input data represents the features, and the output data represents the corresponding labels or target values.
    3. The network adjusts its internal parameters (weights and biases) iteratively using an optimization algorithm like gradient descent to minimize the prediction errors between its outputs and the true labels.
    4. The ultimate goal is to enable the model to make accurate predictions on new, unseen data.
  ii. Unsupervised Learning:
    1. In unsupervised learning, the neural network learns from data without explicit labels or target values. The primary objective is to uncover the underlying patterns and structure in the data.
    2. There are various approaches in unsupervised learning, including clustering, dimensionality reduction, and generative modeling. For example, clustering algorithms group similar data points together, while dimensionality reduction techniques aim to find a lower-dimensional representation of the data while preserving its important characteristics.
  iii. Reinforcement Learning:
    1. In reinforcement learning, the neural network learns from interacting with an environment and receiving feedback in the form of rewards or penalties.
    2. The network, known as an agent, takes actions in the environment and observes the resulting state and reward.
    3. The agent's goal is to learn a policy that maximizes the cumulative reward over time. Reinforcement learning is well-suited for sequential decision-making tasks, such as game playing, robotics, and autonomous vehicles.