A Beginner's Survey of Deep Neural Networks: Foundations and Architectures

Harsh Gupta
Independent Researcher
harshmail281199@gmail.com

May 25, 2025

Abstract

This survey explores the fundamental concepts of neural networks, progressing from shallow architectures like the perceptron to deep neural networks (DNNs). It provides an intuitive and structured understanding of how neural networks evolve in complexity and capability, particularly in supervised learning settings such as classification problems. The paper covers the mathematical foundations, architecture design, training methodologies, including backpropagation and optimization techniques, and illustrates their use in practical applications. This stepwise approach is intended to bridge the conceptual gap between simple and deep models for beginners and practitioners.

Keywords: Perceptron, Shallow Neural Networks, Deep Neural Networks, Classification, Backpropagation, Supervised Learning, Neural Network Training

Contents

Acknowledgements

I would like to express my sincere gratitude to the open-source learning community and the educators behind online platforms such as Coursera, particularly Dr. Andrew Ng and the DeepLearning.AI team, whose Deep Learning Specialization has been instrumental in shaping my foundational understanding of neural networks. I also acknowledge the countless authors, researchers, and contributors of freely available tutorials, articles, and documentation that have enriched my learning journey through the web.

1 Introduction

Deep neural networks have revolutionized modern artificial intelligence by enabling machines to learn and model complex patterns from data. Their impact spans diverse fields such as image classification, natural language understanding, and speech recognition, where they have led to remarkable breakthroughs. This paper aims to provide an intuitive and accessible introduction to the foundations of neural networks, targeting readers who are new to the field. We explore the evolution from simple perceptrons to sophisticated deep learning architectures, covering essential mathematical principles, architectural designs, and training methodologies. This survey is structured to progressively build the reader's understanding, requiring no prior deep learning expertise.

1.1 What is a Neural Network?

A neural network is a computational model inspired by the structure and functioning of the biological brain. It comprises interconnected units called neurons, organized into layers: an input layer, one or more hidden layers, and an output layer. Each neuron receives numerical inputs, computes a weighted sum, applies a non-linear activation function, and passes the output to subsequent layers. Neural networks are highly effective at learning complex patterns from data, making them suitable for a wide range of tasks such as classification, regression, and pattern recognition.

1.2 Learning Outcomes of this Survey Paper

Upon completing this survey, readers will be able to:

- Understand the fundamental structure and operation of neural networks.
- Describe the historical development and motivations underlying deep learning.
- Differentiate between shallow and deep neural network architectures.
- Explain key training mechanisms, including loss functions, backpropagation, and optimization techniques.
- Recognize challenges in training deep networks, such as overfitting and vanishing gradients.
- Identify common applications of deep neural networks across various domains.
- Build a conceptual bridge from simple models like the perceptron to more advanced deep learning systems.

1.3 Motivation

The rapid advancements in artificial intelligence and machine learning are largely driven by the success of deep neural networks. These models have transformed fields such as computer vision, natural language processing, and speech recognition, enabling breakthroughs previously considered decades away. However, the underlying principles and training techniques of deep neural networks can be complex and intimidating for beginners.
This survey seeks to demystify these concepts by providing a clear, structured introduction to neural networks, bridging the gap between simple models and their deep counterparts. By gaining a foundational understanding through this paper, readers will be better prepared to appreciate current

innovations, apply deep learning techniques in practical scenarios, and contribute to future developments. Whether you are a student, practitioner, or researcher, this paper offers essential knowledge to embark on your deep learning journey with confidence.

2 Background and History

Neural networks have their roots in early attempts to mimic the human brain's architecture through computational models. The first conceptual model, the McCulloch-Pitts neuron (1943), laid the groundwork for later developments. The perceptron, introduced by Rosenblatt in 1958, was a pioneering learning algorithm capable of classifying input patterns; however, its inability to solve non-linearly separable problems limited its potential, leading to a decline in interest known as the AI Winter. The field revived in the 1980s with the development of the backpropagation algorithm, which enabled the training of multi-layer networks and opened the door for deeper architectures. Over the past two decades, improvements in hardware, algorithms, and data availability have propelled deep learning into the forefront of artificial intelligence, driving significant advances across numerous fields.

3 Intuition of a Neural Network

Neural networks are inspired by the human brain and are built using layers of interconnected units called neurons. Each neuron receives inputs, processes them with weights and biases, applies a non-linear activation function, and passes the output to the next layer.

To understand how neural networks work at a conceptual level, it is useful to examine their key components and mechanisms, as outlined below.

3.1 Structure of Layers

A neural network typically consists of:

- **Input Layer:** Accepts the input features. In our example image, there are 3 input neurons corresponding to 3 features.
- **Hidden Layers:** Perform intermediate computations. Each neuron here transforms the inputs it receives using a weighted sum and an activation function. The example shows two hidden layers with 4 neurons each.
- **Output Layer:** Produces the final prediction or classification result. The number of output neurons depends on the task. In binary classification, it's often 1.

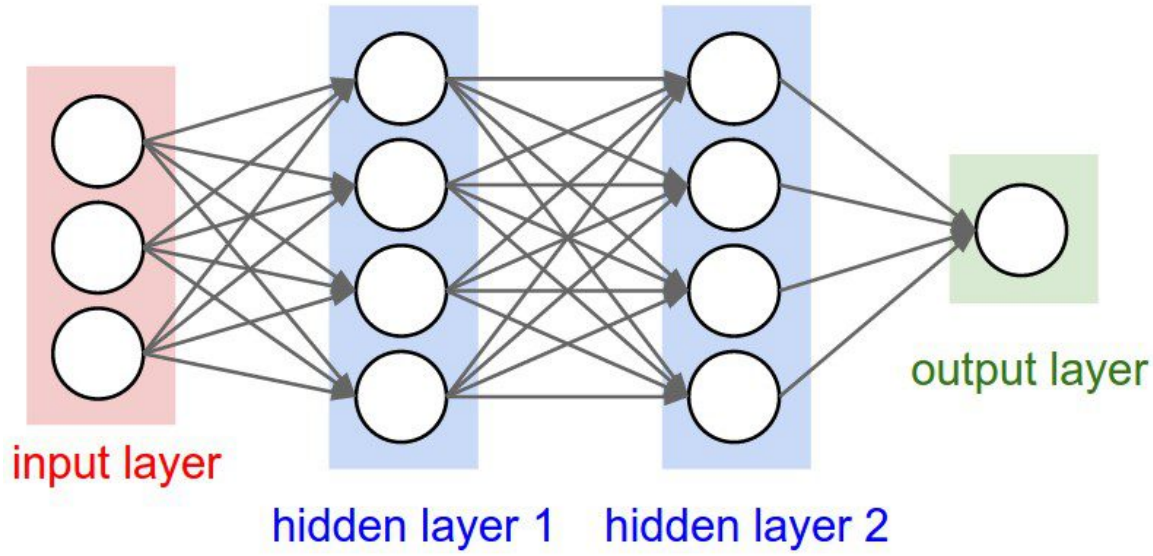

Figure 1: A Simple Neural Network

3.2 Activation and Non-Linearity

After calculating the weighted sum of inputs, each neuron applies an **activation function** to introduce non-linearity. Without this, a neural network would behave like a simple linear model.

Common activation functions include:

- **Sigmoid:** Maps input to range (0,1)
- **Tanh:** Maps input to range (-1,1)
- **ReLU (Rectified Linear Unit):** Converts negative values to zero

3.3 Shallow vs. Deep Networks

- **Shallow Neural Network:** Contains only one hidden layer. Suitable for simple problems.
- **Deep Neural Network (DNN):** Contains multiple hidden layers. These can model more complex patterns and hierarchies of features.

As the number of layers increases, the network can capture more abstract representations of data but also becomes harder to train.

3.4 Forward and Backward Propagation (Overview)

Forward Propagation is the process of computing the output from the input by moving data forward through the network layer by layer.

Backward Propagation (Backprop) is the training mechanism. After forward propagation,

the network compares its output to the actual target using a loss function. It then computes gradients of the loss with respect to each weight and updates the weights using optimization techniques like gradient descent.

This learning loop continues until the network performance stabilizes.

4 Perceptron and Feedforward Networks

4.1 Perceptron: The Simplest Neural Unit

The perceptron is the most fundamental building block of neural networks. It is a binary classifier that decides whether an input belongs to one class or another using a linear decision boundary. Mathematically, a perceptron takes a weighted sum of its inputs, adds a bias, and passes the result through an activation function.

The output y of a perceptron is given by:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

Where:

- x_i: Input feature i
- w_i: Weight associated with x_i
- b: Bias term
- f: Activation function (commonly the step function)

The step function is defined as:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

4.2 Limitations of Perceptron

Although simple and intuitive, a single-layer perceptron can only solve linearly separable problems. For instance, it cannot solve the XOR problem, which necessitates more complex architectures involving multiple layers and nonlinear activation functions.

4.3 Feedforward Neural Networks

Feedforward Neural Networks (FNNs) are a type of neural network where connections between nodes do not form cycles. They are organized in layers:

- **Input Layer:** Takes the raw input features $(x_1, x_2, \ldots, x_n)$.
- **Hidden Layers:** Intermediate layers that perform transformations using weights, biases, and activation functions.
- **Output Layer:** Produces the final output based on the task (e.g., binary class, multi-class, regression).

Each neuron in a hidden or output layer computes the following:

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j \tag{3}$$

$$a_j = f(z_j) \tag{4}$$

Where:

- w_{ji}: Weight from neuron i in the previous layer to neuron j
- x_i: Output from neuron i in the previous layer
- b_j: Bias for neuron j
- f: Activation function (e.g., sigmoid, ReLU)

4.4 Loss Function

The loss function quantifies the difference between the predicted output $\hat{y}$ and the true label y. For binary classification, a common choice is the Binary Cross-Entropy loss:

$$\mathcal{L}(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right] \tag{5}$$

The goal of training is to minimize this loss across all training examples.

4.5 Backpropagation: Learning in Neural Networks

Backpropagation is the core algorithm for training feedforward networks. It involves:

1. **Forward Pass:** Compute activations layer-by-layer from input to output.
2. **Loss Calculation:** Evaluate the loss using the predicted and true values.
3. **Backward Pass:** Propagate the error backward using the chain rule to compute gradients of the loss w.r.t. each weight.
4. **Weight Update:** Update the weights using an optimizer like gradient descent.

Weight update rule using Gradient Descent:

$$w := w - \eta \frac{\partial \mathcal{L}}{\partial w} \tag{6}$$

Where:

- η: Learning rate (step size)
- $\frac{\partial \mathcal{L}}{\partial w}$: Gradient of loss with respect to the weight

4.6 Illustrative Diagram

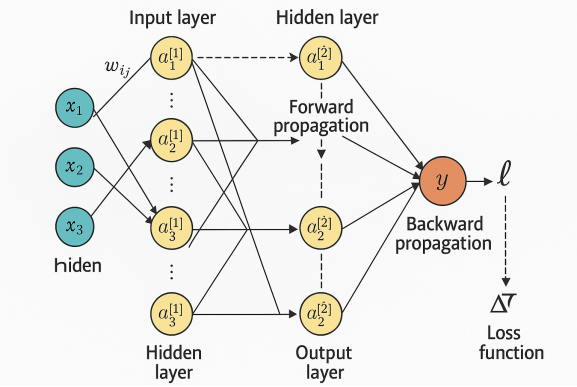

Figure 2: Forward and Backward Propagation in a Neural Network

4.7 Summary

Feedforward networks are foundational to deep learning. They allow complex functions to be modeled through layers of neurons and are trained using backpropagation to minimize loss functions. Although simple in architecture compared to modern models, understanding them is essential for grasping more advanced deep networks.

5 Backpropagation Algorithm

Backpropagation is the cornerstone of training deep neural networks. It is a supervised learning technique used to compute gradients for updating the weights in the network using gradient descent. It involves propagating the error from the output layer backward through the network, layer by layer, using the chain rule of calculus.

5.1 Overview

Backpropagation involves the following steps:

1. Forward Pass – compute the output of the network.
2. Compute the Loss – compare predicted and actual outputs.
3. Backward Pass – compute gradients using derivatives.
4. Weight Update – update weights using gradient descent.

5.2 Notations

Let:

- a^l : activation of layer l
- z^l : weighted input to layer l (before activation)
- W^l : weight matrix for layer l
- b^l : bias vector for layer l
- σ : activation function (e.g., sigmoid)
- L : total number of layers
- δ^l : error at layer l
- $\mathcal{L}$: loss function

5.3 Forward Propagation Equations

For each layer l, from input to output:

$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$

5.4 Loss Function

For simplicity, assume Mean Squared Error (MSE):

$$\mathcal{L} = \frac{1}{2} \left\| a^L - y \right\|^2$$

where a^L is the output of the network and y is the target label.

5.5 Step-by-Step Derivatives (Backpropagation)

Step 1: Output Error (Layer L):

$$\delta^L = \nabla_{a^L} \mathcal{L} \circ \sigma'(z^L) = (a^L - y) \circ \sigma'(z^L)$$

Step 2: Backpropagate the Error
For layers $l = L-1, L-2, \ldots, 2$:

$$\delta^l = \left((W^{l+1})^T \delta^{l+1}\right) \circ \sigma'(z^l)$$

Step 3: Compute Gradients

$$\frac{\partial \mathcal{L}}{\partial W^l} = \delta^l (a^{l-1})^T$$
$$\frac{\partial \mathcal{L}}{\partial b^l} = \delta^l$$

Step 4: Update Weights and Biases
Using Gradient Descent:

$$W^l := W^l - \eta \frac{\partial \mathcal{L}}{\partial W^l}$$
$$b^l := b^l - \eta \frac{\partial \mathcal{L}}{\partial b^l}$$

where η is the learning rate.

5.6 Activation Function Derivatives (Example)

For sigmoid:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad \Rightarrow \quad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

For ReLU:

$$\text{ReLU}(x) = \max(0, x) \quad \Rightarrow \quad \text{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

5.7 Chain Rule Intuition

Backpropagation heavily relies on the chain rule:

$$\frac{d\mathcal{L}}{dW} = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dW}$$

This chaining allows gradient information to be passed backward through layers.

5.8 Why Backpropagation Works

The key idea behind backpropagation is to efficiently compute how each weight in the network affects the final output error. By storing intermediate values from the forward pass (e.g., z^l, a^l), we can avoid redundant calculations during the backward pass.

5.9 Summary

Backpropagation is the algorithmic backbone for training neural networks. It enables efficient computation of gradients, making it possible to scale learning to deep architectures. Each training step gradually tunes the weights to minimize the error, making the model better at prediction.

6 Activation Functions in Neural Networks

6.1 Purpose of Activation Functions

Explain why neural networks need non-linearity — without activation functions, no matter how many layers, the network behaves like a linear model.

6.2 Common Activation Functions

6.2.1 Sigmoid:

Used in binary classification tasks, especially output layers.
Disadvantage: vanishing gradients.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \text{Range: } (0, 1)$$

6.2.2 Tanh:

Zero-centered and often preferred over sigmoid.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \text{Range: } (-1, 1)$$

6.2.3 ReLU (Rectified Linear Unit):

Popular in hidden layers of deep networks. Fast and avoids vanishing gradient for $x > 0$.

$$\text{ReLU}(x) = \max(0, x)$$

6.2.4 Leaky ReLU:

Fixes ReLU's issue of dying neurons.

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases}$$

6.2.5 Softmax:

Used in the output layer of multi-class classifiers:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

6.3 When to Use Which Function

The choice of activation function depends on the network architecture, the nature of the data, and the task being solved. Below is a guideline for when to use each commonly used activation function:

- **Sigmoid:** Best used in the output layer for binary classification tasks. Due to its saturation and vanishing gradient issues, it is rarely used in hidden layers.
- **Tanh:** Similar to sigmoid but zero-centered. Preferable over sigmoid for hidden layers when outputs are expected to have both negative and positive values.
- **ReLU:** Widely used in hidden layers of deep networks due to its simplicity and computational efficiency. Helps avoid vanishing gradients when $x > 0$.
- **Leaky ReLU / Parametric ReLU:** Preferred when there's concern about dead neurons in ReLU (i.e., neurons that output zero and never activate). These provide a small, non-zero gradient when $x < 0$.
- **Softmax:** Used in the output layer of multi-class classification problems to produce a probability distribution over classes.

Choosing the right activation function can significantly impact the network's ability to converge and generalize well on unseen data.

6.4 Impact on Backpropagation

Activation functions not only define the output of a neuron but also influence how gradients are propagated during training. During backpropagation, the derivative of the activation function is multiplied with the gradient of the loss to compute weight updates.

- **Sigmoid:** Its derivative is small for large or small input values, which can lead to the vanishing gradient problem:
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$
- **Tanh:** Also suffers from vanishing gradients but performs better than sigmoid in zero-centered tasks:
$$\tanh'(x) = 1 - \tanh^2(x)$$
- **ReLU:** Has a derivative of 1 for positive inputs, which helps preserve gradients and accelerates training:
$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$
However, for $x \leq 0$, gradients become zero, potentially causing "dead" neurons.

- **Leaky ReLU:** Provides a small gradient for $x < 0$, thus mitigating the dead neuron problem:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases}$$

- **Softmax:** The derivative of softmax is more complex, as it involves a Jacobian matrix. It is typically used in combination with cross-entropy loss, where the derivative simplifies significantly, making backpropagation efficient in practice.

The effectiveness of backpropagation strongly depends on choosing activation functions with non-zero and stable gradients. Poor choices can result in slow or failed training due to exploding or vanishing gradients.

7 Deep Neural Networks

A neural network is considered **deep** when it contains two or more hidden layers between the input and output layers. While shallow networks (with only one hidden layer) can approximate many functions, deep networks are more powerful in practice for capturing complex patterns and hierarchical relationships in data.

7.1 What Makes a Network Deep?

The depth of a neural network refers to the number of layers through which data passes. Specifically:

- **Shallow Network:** Typically contains a single hidden layer.
- **Deep Neural Network (DNN):** Contains two or more hidden layers, possibly dozens or hundreds in modern architectures.

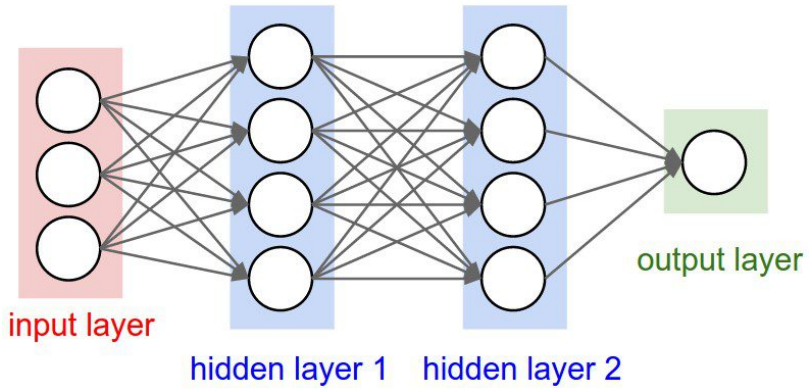

Figure 3: Illustration of a Deep Neural Network with 2 Hidden Layers

7.2 Why Depth Matters

Each additional layer in a deep network allows the model to learn increasingly abstract representations:

- Lower layers detect basic features (e.g., edges in images).
- Intermediate layers combine these into patterns (e.g., shapes).
- Higher layers recognize complex objects or concepts (e.g., faces or letters).

This hierarchical feature learning is one of the key reasons deep networks outperform shallow ones in many real-world tasks.

7.3 Theoretical Foundations

The **Universal Approximation Theorem** states that a single hidden-layer neural network can approximate any continuous function on a compact domain. However, achieving this may require exponentially many neurons. Deep networks, on the other hand, can approximate the same functions more efficiently, with fewer neurons and better generalization, by reusing features across layers.

8 Challenges and Limitations

Despite their remarkable capabilities, deep neural networks (DNNs) come with a range of challenges and limitations that affect their design, training, and deployment. Understanding these issues is critical for building robust and efficient models.

8.1 Vanishing and Exploding Gradients

As gradients are propagated backward through many layers during training, they may become extremely small (vanish) or large (explode). This leads to:

- **Vanishing gradients:** Weight updates become negligible, especially in earlier layers, slowing or stalling learning.
- **Exploding gradients:** Weight updates become excessively large, causing instability or divergence in training.

This is particularly problematic when using activation functions like sigmoid or tanh and has led to the development of ReLU, better weight initialization schemes, and techniques like gradient clipping.

8.2 Overfitting

Deep networks have a high capacity to learn patterns, but this also makes them prone to overfitting, where the model performs well on training data but fails to generalize to unseen data.

- **Symptoms:** Low training error but high validation/test error.
- **Solutions:** Regularization (L1, L2), dropout, early stopping, and data augmentation.

8.3 Computational Cost

Training deep networks is computationally expensive, often requiring specialized hardware like GPUs or TPUs, especially for large datasets or complex models. This also increases energy consumption and development time.

8.4 Data Dependency

DNNs typically require large volumes of labeled data to perform effectively. In domains where labeled data is scarce or expensive to obtain (e.g., medical or legal), training a deep model becomes challenging.

8.5 Interpretability

Unlike traditional models like decision trees or linear regression, DNNs operate as black-box models. It is often unclear how or why a network arrives at a particular prediction, which poses problems in critical areas like healthcare, finance, and law.

8.6 Hyperparameter Sensitivity

Deep networks are sensitive to many design choices and training parameters, such as learning rate, batch size, activation functions, and network depth. Poorly chosen hyperparameters can result in suboptimal performance or training failure.

8.7 Adversarial Vulnerability

Deep models can be fooled by adversarial examples — small, carefully crafted perturbations to the input that cause incorrect outputs. This raises security concerns in real-world applications like autonomous driving and biometric systems.

8.8 Deployment and Maintenance

Deploying deep models in production involves challenges such as model compression, inference speed, updating with new data, and monitoring performance over time to avoid concept drift.

9 Applications

Deep neural networks (DNNs) have become the foundation for solving many complex real-world problems. Their ability to learn hierarchical and abstract representations from data has enabled significant breakthroughs across various fields.

9.1 Computer Vision

DNNs, particularly Convolutional Neural Networks (CNNs), have revolutionized the field of computer vision.

- **Image Classification:** Assigning a label to an entire image (e.g., identifying cats vs. dogs).
- **Object Detection:** Identifying and localizing multiple objects within an image.
- **Image Segmentation:** Classifying each pixel in an image into categories.
- **Facial Recognition:** Matching or verifying human faces in photos or videos.

9.2 Natural Language Processing (NLP)

DNNs, especially Recurrent Neural Networks (RNNs) and Transformers, have drastically improved how machines understand and generate human language.

- **Machine Translation:** Translating text between languages (e.g., English to French).
- **Text Classification:** Categorizing documents, emails, or reviews based on sentiment or topic.
- **Question Answering and Chatbots:** Building intelligent assistants like ChatGPT or Siri.
- **Summarization:** Condensing large documents into concise summaries.

9.3 Speech and Audio Processing

Neural networks are integral to systems that convert between speech and text or interpret audio signals.

- **Speech Recognition:** Converting spoken language into text.
- **Speech Synthesis (Text-to-Speech):** Generating human-like speech from text.
- **Voice Authentication:** Verifying identities through vocal patterns.

9.4 Autonomous Systems and Robotics

DNNs are key to enabling perception, control, and decision-making in intelligent agents.

- **Self-Driving Vehicles:** Interpreting visual surroundings and making navigation decisions.
- **Industrial Robots:** Performing tasks like sorting, assembly, and inspection with minimal human input.
- **Drones:** Navigating and avoiding obstacles using onboard sensors and vision systems.

9.5 Recommendation Systems

Deep learning powers many personalized content platforms:

- **Media Recommendations:** Suggesting videos (e.g., YouTube, Netflix) or songs (e.g., Spotify).
- **E-Commerce:** Recommending products based on browsing and purchase history.
- **Social Media Feeds:** Personalizing posts and ads shown to users.

10 Conclusion

This survey paper has provided an introductory yet comprehensive overview of deep neural networks (DNNs), starting from the fundamental concept of perceptrons and progressing through feedforward architectures, activation functions, and the training process via forward and backward propagation. We have explored how deep networks differ from shallow models, the mathematical basis of their learning process, and their practical applications across a wide range of domains.

Despite their powerful capabilities, DNNs are not without limitations. Issues such as vanishing gradients, overfitting, interpretability, and computational demands remain active areas of research. However, through the development of new architectures, training techniques, and regularization strategies, many of these challenges are being addressed.

Deep neural networks continue to drive advances in fields like computer vision, natural language processing, speech recognition, healthcare, finance, and autonomous systems. Their ability to learn complex representations from data has transformed modern AI.

As this field evolves, future work will focus on improving interpretability, reducing data dependency, increasing training efficiency, and developing models that are more robust and generalizable. This paper serves as a foundational guide for learners and practitioners looking to understand the essential components and significance of deep neural networks.

References

[1] Andrew Ng's Machine Learning Specialization - offered by DeepLearning.AI.

[2] Andrew Ng's Deep Learning Specialization - offered by DeepLearning.AI.

[3] Andrew Ng and Kian Katanforoosh. CS230 - Stanford Class.

[4] Deep Learning tutorials and articles - Medium, Towards Data Science.

[5] Wikipedia - Artificial Neural Network.