# Deep Learning

Vikas Thammanna Gowda

02/13/2025

## 1 Intro to Deep Learning

Deep learning is a subfield of machine learning that uses deep neural networks—networks with multiple layers—to learn complex patterns from data. Each layer can be thought of as a stage of feature extraction, going from simple to complex representations.

**Neural Networks as Building Blocks:** Deep learning relies on artificial neural networks, which are computational structures loosely inspired by the human brain. Each "layer" (or set of neurons) in the network extracts increasingly abstract features from the data.

**Representation Learning:** In traditional machine learning, one might manually design features for the model. Deep learning automates this feature extraction by letting the network learn which patterns are important.

**Data Requirements and Computation:** Training deep networks typically requires large datasets and substantial computational resources (often involving GPUs). Deep networks can process images, text, audio, and more, achieving state-of-the-art results in many domains.

### Example 1: Image Recognition

**Task:** Classify images into categories (e.g., detecting whether a picture contains a cat or a dog).
**Deep Learning Approach:**

- Use a Convolutional Neural Network (CNN) with multiple layers.
- Early layers learn simple patterns such as edges or corners; deeper layers combine these into higher-level concepts like a "cat's ear" or a "dog's muzzle."
- You don't manually define these features; the network learns them through training on thousands or millions of labeled images.

**Outcome:** By "stacking" multiple layers, the model can learn to classify images with very high accuracy.

### Example 2: Language Translation

**Task:** Translate text from English to Spanish (or any other language pair).
**Deep Learning Approach:**

- Use a Recurrent Neural Network (RNN) or a Transformer architecture (like those used in large language models).
- The model reads words (or tokens) in one language and uses learned parameters to generate a correct sequence of words in the target language.
- Again, the model automatically learns linguistic features—such as vocabulary, grammar rules, and idiomatic expressions—without you explicitly programming them.

In both cases, the hallmark of deep learning is that the system automatically learns features from large datasets through multiple layers of transformations.

## 1.1 Difference Between Deep Learning and Machine Learning

Machine learning is the broader field encompassing many algorithms and approaches (e.g., linear regression, decision trees, support vector machines, and shallow neural networks). Deep learning is a specific subset of machine learning focused on "deep" (multi-layered) neural network architectures.

### 1. Feature Engineering vs. Feature Learning

**Traditional Machine Learning (ML):**

- Often requires manual feature engineering. For instance, if you are building a spam filter, you might create features such as "number of times certain words appear" or "presence of suspicious links."
- You then feed these engineered features into an algorithm like a Random Forest or SVM.

**Deep Learning (DL):**

- Learns features automatically. For the same spam filter, you might feed raw email text into a deep network (like an LSTM or Transformer) and let it figure out which words, phrases, or sentence structures are indicative of spam.

### 2. Data and Compute Requirements

- **ML:** Works well with smaller datasets, especially if you have carefully designed features.
- **DL:** Tends to require large amounts of labeled data and powerful hardware (GPUs/TPUs) to train effectively. Performance often scales with more data.

### 3. Complexity and Performance

- **ML:** Traditional methods can be easier to interpret and tune; they may reach a performance plateau if you keep adding more data without a major change in feature engineering or model design.
- **DL:** Neural networks with many layers can capture extremely complex relationships. They typically improve as you add more data and more layers, often achieving state-of-the-art results in tasks like image recognition, natural language processing, and voice recognition.

**Example 1: Spam Detection**
**Traditional Machine Learning (ML):**

- Gather emails labeled "spam" or "not spam."
- Manually decide on features (e.g., "number of exclamation marks," "contains 'free money' phrase").
- Use a classification algorithm like Logistic Regression or Naive Bayes.
- Model performance heavily depends on how good your chosen features are.

**Deep Learning (DL):**

- Gather large amounts of emails (again, labeled "spam" or "not spam").
- Feed the raw text or tokenized text directly into a deep neural network (e.g., an LSTM or Transformer model).
- The network automatically learns which words, phrases, and context patterns best distinguish spam from legitimate email.

**Example 2: Customer Churn Prediction**

**Traditional ML:**

- Business analysts define critical indicators (e.g., last login date, average purchase value), and feed them into a Random Forest to predict whether a customer will leave.

**Deep Learning:**

- If there's a massive volume of customer behavioral data (clickstream, purchase logs, support tickets), a deep network could ingest this raw or minimally processed data and learn complex signals (e.g., user navigation patterns, text from support conversations) that correlate with churn risk.

## 2 Perceptrons

In biology, a neuron is a cell that receives signals from other neurons, processes them, and then (if it crosses a certain threshold) sends an output signal to other neurons. Artificial neurons are a simplified mathematical abstraction of this process, used in neural networks.

The perceptron is one of the earliest forms of an artificial neuron, introduced by Frank Rosenblatt in 1958. It is essentially a binary classifier that maps input features to either 0 or 1 (e.g., "spam" vs. "not spam," "cat" vs. "dog," etc.), based on a linear combination of the inputs.

### 2.1 How a Perceptron Works

**Inputs and Weights:**
Suppose we have inputs $x_1, x_2, \ldots, x_n$ with corresponding weights $w_1, w_2, \ldots, w_n$.
**Summation:** Compute the weighted sum (plus bias $b$):

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

**Step Activation:** In the classic perceptron, the activation is a step function:

$$\text{output} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

**Decision:** If the output is 1, the perceptron decides "true" (or "positive" class). If the output is 0, it decides "false" (or "negative" class).

### 2.2 Perceptron Learning Rule (Training)

The perceptron learning rule adjusts the weights $w_i$ and bias $b$ whenever the perceptron misclassifies an example. Given a training dataset with known labels (e.g., cat/dog), the algorithm is:

1. Initialize the weights and bias (often to small random values or zeros).
2. For each training example $(x, \text{label})$:
    (a) Compute the perceptron's prediction.
    (b) If the prediction is correct, do nothing.
    (c) If the prediction is incorrect, update the weights and bias:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{where} \quad \Delta w_i = \eta \times (\text{label} - \text{prediction}) \times x_i$$

$$b \leftarrow b + \eta \times (\text{label} - \text{prediction})$$

Here, $\eta$ is the learning rate, which controls how big an update you make each time you correct an error.

This rule effectively nudges the decision boundary toward correctly classifying misclassified examples.

## 2.3 What is an Activation Function?

In a neural network, each neuron computes a weighted sum of its inputs plus a bias term:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

The activation function $f(\cdot)$ then transforms $z$ into the neuron's output:

$$\text{output} = f(z)$$

Without an activation function, the neuron would simply perform a linear transformation of the input. Nonlinear activation functions allow neural networks to approximate complex, nonlinear relationships.

### 1. Step Function

**Definition** The step function (or Heaviside function) outputs a 1 if the input is above a threshold (often 0) and 0 otherwise:

$$\text{Step}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

**Key Characteristics**

- **Binary Output:** The neuron's output is either 0 or 1, making it suitable for very simple binary classification (e.g., early perceptron models).
- **Non-differentiable:** Because it jumps abruptly at $z = 0$, its gradient is 0 everywhere except at the threshold, which complicates gradient-based training (backpropagation).
- **Rarely Used in Modern Deep Learning:** Modern networks usually need smooth, continuous activations.

**Typical Use** A single-layer perceptron (the original neural network) uses the step function to decide between two classes (e.g., "positive" vs. "negative").

### 2. Sigmoid (Logistic) Function

**Definition** The sigmoid function squashes any real-valued input to a $(0, 1)$ range:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

**Key Characteristics**

- **Continuous and Differentiable:** Allows for gradient-based training.
- **Range:** Outputs a value between 0 and 1, often interpreted as a probability in binary classification.
- **Saturation and Vanishing Gradients:** For large positive $z$, $\sigma(z) \approx 1$, and for large negative $z$, $\sigma(z) \approx 0$. Gradients become very small (close to zero) in these regions, slowing down learning (a problem known as the vanishing gradient).
- **Centered at 0.5:** Sigmoid outputs 0.5 when $z = 0$.

**Typical Use**

- **Binary Classification (output layer):** Often used as the final activation in a network predicting two classes (e.g., "spam" vs. "not spam").
- **Hidden Layers (less common nowadays):** Historically used, but ReLU or other activations are usually preferred to reduce vanishing gradient issues.

**3. Hyperbolic Tangent (tanh) Function**

**Definition** The hyperbolic tangent function is closely related to the sigmoid function but maps inputs to $(-1, 1)$:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

**Key Characteristics**

- **Range:** Outputs values from $-1$ to 1, which can help with centering data around 0.
- **Similar Gradient Saturation Issue:** Like sigmoid, tanh saturates for large magnitudes of $z$, leading to small gradients.
- **Zero-Centered:** Because outputs can be negative or positive, the average output of a batch can be closer to zero, often helping optimization in some cases more than sigmoid.

**Typical Use**

- **Hidden Layers:** Sometimes used as the activation for recurrent neural networks (LSTM, GRU) or in some feed-forward networks.
- **Still Susceptible to Vanishing Gradients:** However, it often outperforms sigmoid in practice because it's zero-centered.

**4. Rectified Linear Unit (ReLU) Function**

**Definition** The ReLU function outputs 0 if the input is negative and the raw input otherwise:

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

**Key Characteristics**

- **Simplicity:** Very simple definition and easy to compute.
- **Nonlinearity:** If $z \leq 0$, output $= 0$; if $z > 0$, output $= z$.
- **Sparsity:** Neurons can become "inactive" (output 0) for negative inputs.
- **Mitigates Vanishing Gradient:** For $z > 0$, the gradient is 1, so it avoids some saturation problems of sigmoid/tanh.
- **Dead Neurons:** If weights update in such a way that a neuron's input is always negative, that neuron may never "activate" again, becoming effectively dead.

**Typical Use**

- **Most Common Activation in Hidden Layers:** Especially in computer vision tasks (CNNs) and many feed-forward networks.
- **Variants:** Leaky ReLU, Parametric ReLU, ELU, GELU, etc., which address the "dead neuron" problem by allowing a small slope for negative $z$.

## 5. Softmax Function

**Definition** Softmax is used primarily for multi-class classification. Given a vector of $K$ inputs $z = (z_1, z_2, \ldots, z_K)$, the softmax output for class $i$ is:

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

**Key Characteristics**

- **Converts Outputs to Probabilities:** The outputs of the softmax for all $K$ classes sum to 1 and lie between 0 and 1.
- **Used in Output Layer:** Almost always used in the final layer for multi-class problems (e.g., classifying images into 10 categories).
- **Differentiable:** Allows for gradient-based optimization with cross-entropy loss.

**Typical Use**

- **Multi-Class Classification:** E.g., classifying an image as cat, dog, or bird (the output layer would have 3 units with a softmax activation).