# Basic ML/DL

KUCSEPotato
a23493307@gmail.com

June 2025

*This paper is written for a project that develops ML/DL models. Since I am a learner, not an expert, if there are any errors, please kindly let me know.*

## 1 Introduction

Machine Learning (ML) and Deep Learning (DL) have become foundational technologies across various domains such as image recognition, natural language processing, autonomous driving, and scientific discovery. However, for many newcomers, understanding how these models work and how to get started can be a daunting task.

This paper is written with the goal of providing a clear and accessible introduction to the fundamental concepts of ML and DL. Rather than diving straight into advanced topics or code-heavy implementations, we begin by answering basic but essential questions: What is machine learning? How does it differ from traditional programming? What components make up a machine learning model? Why are loss functions, optimization, and regularization important?

I also introduce core ideas such as data representation, model training, evaluation metrics, and overfitting, all in a way that assumes no prior background in mathematics or computer science beyond high school level. Furthermore, we offer simplified explanations of widely-used techniques like gradient descent, activation functions, and neural networks.

By the end of this paper, the reader should have a solid grasp of key ML/DL principles and be equipped with the foundational knowledge needed to take the next steps — whether that be implementing models in PyTorch, reading academic papers, or exploring more advanced topics like convolutional or transformer-based architectures.

This document is intended as a first step toward deeper learning, written by a learner for learners.

# 2 Big Picture of Deep Learning: From Input to Learning

Deep learning can often feel overwhelming for beginners due to the many components involved. However, most elements in deep learning serve a single unified goal:

**To train a model that minimizes a loss function and accurately predicts outputs from inputs.**

This section provides a high-level overview of the deep learning process, highlighting each stage from input to training, and explaining the role of components such as models, loss functions, optimizers, normalization, and activation functions.

## 1. Problem and Data Definition

Every deep learning task starts with defining a problem (e.g., image classification). This involves preparing:

- **Input data** $x$: e.g., images of animals
- **Target label** $y$: e.g., 0 for cat, 1 for dog
- A dataset of pairs $(x, y)$ for training

## 2. Model Design

The model defines how inputs are transformed into predictions. It consists of layers and parameters (weights $W$, biases $b$). For image tasks, Convolutional Neural Networks (CNNs) are commonly used.

$$\hat{y} = f(x; \theta)$$

where $\theta$ represents all trainable parameters.

## 3. Forward Propagation

Input data passes through the model:

- Convolutional Layers (extract features)
- Activation Functions (e.g., ReLU for non-linearity)
- Pooling Layers (downsampling)
- Fully Connected Layers (decision-making)
- Output Layer (e.g., Softmax for classification)

The output is a prediction $\hat{y}$.

## 4. Loss Computation

A **loss function** compares the prediction $\hat{y}$ with the true label $y$ to compute the error.

$$\mathcal{L} = \text{Loss}(\hat{y}, y)$$

Common choices include Cross-Entropy for classification or Mean Squared Error (MSE) for regression.

## 5. Backward Propagation

Using the computed loss, gradients of the loss with respect to each parameter are calculated through **backpropagation**.

## 6. Optimization (Weight Update)

An optimizer (e.g., SGD, Adam) uses the gradients to update parameters in order to reduce the loss.

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}$$

Here, $\eta$ is the learning rate.

## 7. Regularization and Stabilization Techniques

To improve training stability and generalization:

- **Normalization (e.g., BatchNorm)**: Stabilizes intermediate outputs
- **Dropout / Stochastic Depth**: Prevents overfitting
- **Activation Functions (e.g., ReLU, GELU)**: Adds non-linearity

## 8. Evaluation and Validation

After training, the model is tested on unseen data to evaluate its performance and generalization.

## 9. Summary Diagram

Input $\rightarrow$ Model (CNN etc.) $\rightarrow \hat{y} \rightarrow$ Loss $\rightarrow$ Backpropagation $\rightarrow$ Optimizer $\rightarrow$ Updated Model

Each part exists to help one goal: **reduce the loss and improve predictions.**

# 3 Stage-by-Stage Deep Dive

While the previous section offered a high-level overview, this section delves into the mathematical formulations and intuitive understanding behind each component in the deep learning training pipeline.

## 3.1 Model as a Function: $f(x;\theta)$

At its core, a deep learning model is a parametric function $f(x;\theta)$ that transforms an input $x$ into an output $\hat{y}$ through a composition of layers. Each layer performs a mathematical operation:

$$\text{Layer}_i: \quad z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}, \quad a^{[i]} = \sigma(z^{[i]})$$

Where:

- $W^{[i]}$, $b^{[i]}$ are weights and biases of layer $i$
- $a^{[i-1]}$ is the input (activation) from the previous layer
- $\sigma(\cdot)$ is an activation function (e.g., ReLU)

This composition is repeated over multiple layers to learn complex hierarchical representations.

## 3.2 Forward Propagation

During forward pass, input data $x$ is passed through the model layers to compute the output $\hat{y}$:

$$\hat{y} = f(x;\theta)$$

For classification, $\hat{y}$ typically represents class probabilities using a softmax function:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } k = 1, \ldots, K$$

### 3.3 Loss Function

The loss quantifies how much the prediction $\hat{y}$ differs from the ground truth $y$. For a classification task:

$$\mathcal{L}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

This is the categorical cross-entropy loss, where $y_k$ is a one-hot encoded target vector.

For regression tasks, Mean Squared Error is used:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|^2$$

### 3.4 Backpropagation: Computing Gradients

To improve the model, we compute the gradient of the loss with respect to each parameter using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial W^{[i]}} = \frac{\partial \mathcal{L}}{\partial a^{[L]}} \cdot \cdots \cdot \frac{\partial a^{[i]}}{\partial z^{[i]}} \cdot \frac{\partial z^{[i]}}{\partial W^{[i]}}$$

This process flows backward from the output to the input — hence the name "backpropagation".

### 3.5 Optimization: Gradient Descent

Using the gradients, we update each parameter in the direction that reduces the loss:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}$$

Where:

- $\eta$ is the learning rate,
- $\nabla_\theta \mathcal{L}$ is the gradient of the loss with respect to the parameters.

Variants like Momentum, RMSProp, and Adam enhance this update by adapting the learning rate or smoothing gradients over time.

### 3.6 Role of Activation Functions

Activation functions introduce non-linearity into the model, enabling it to learn complex patterns. Without them, the model would behave like a linear function.

Common activations:

- **ReLU:** $\max(0, x)$ — fast and widely used
- **Sigmoid:** $\frac{1}{1+e^{-x}}$ — squashes values to (0,1)
- **GELU:** smoother version of ReLU, used in Transformer-based models

## 3.7 Normalization and Regularization

**Batch Normalization** standardizes layer inputs to have mean 0 and variance 1 per mini-batch:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad y_i = \gamma \hat{x}_i + \beta$$

**Dropout** randomly sets neuron outputs to 0 during training to prevent overfitting.

## 3.8 Final Output and Evaluation

After training, the model is evaluated on unseen test data using metrics such as accuracy, precision, or F1-score. Proper evaluation ensures that the model generalizes beyond the training set.

# 4 Case Study: Image Classification with Linear Models

This section provides a concrete example of how deep learning principles apply to the task of image classification. Starting from a basic K-Nearest Neighbors (K-NN) approach, we gradually move toward linear classifiers and modern loss functions such as Softmax and Cross Entropy.

## 4.1 Problem Setup and Data Representation

The image classification task aims to assign a label $y \in \{1, ..., K\}$ to a given input image $x \in R^D$. For example, CIFAR-10 images of size $32 \times 32 \times 3$ are flattened into 3072-dimensional vectors.

Unlike humans, computers perceive images as numerical tensors. This discrepancy between human perception and raw data representation is known as the **semantic gap**. For example, an image may appear as a $32 \times 32 \times 3$ tensor of integers from 0 to 255, but no meaning is attached to these numbers unless learned through a model.

## 4.2 K-Nearest Neighbor: A Simple Baseline

K-Nearest Neighbor (K-NN) is a simple, non-parametric method. It memorizes all training data and predicts the label of a new sample by majority vote among the $K$ most similar examples (based on a distance metric such as L1 or L2).

While intuitive and easy to implement, K-NN suffers from high memory and inference costs. It does not actually learn a model and does not scale well for high-dimensional inputs like images.

## 4.3 Linear Classifier: A Parametric Mapping

To make learning scalable, we define a linear model:

$$f(x; W, b) = Wx + b$$

Here:

- $x \in R^D$: input vector (e.g., flattened image)
- $W \in R^{K \times D}$: weight matrix
- $b \in R^K$: bias vector
- $f(x)$: output score vector (one score per class)

The matrix $W$ and vector $b$ are the **parameters** we learn. This linear transformation maps the input $x$ to a new space where classification is easier.

## 4.4 Loss Functions: Softmax and Cross-Entropy

The output scores are not yet probabilities. We apply the Softmax function:

$$P(y = k \mid x) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where } s_k = [Wx + b]_k$$

Then we compute the Cross-Entropy loss:

$$\mathcal{L}(x, y) = -\log P(y = y_{\text{true}} \mid x)$$

This measures how well the predicted distribution $P$ matches the true distribution $Q$ (often a one-hot vector).

Softmax + Cross-Entropy encourages the model to give high confidence to the correct class and low scores to others.

## 4.5 Relationship to KL Divergence

The cross-entropy loss can be decomposed as:

$$H(P, Q) = H(P) + D_{\text{KL}}(P \| Q)$$

In practice, $H(P)$ is constant, so minimizing cross-entropy is equivalent to minimizing the KL divergence between the true and predicted distributions.

## 4.6 SVM vs Softmax: A Conceptual Comparison

| **Aspect** | **SVM** | **Softmax Classifier** |
|---|---|---|
| Output | Margin scores | Probability scores |
| Learning Goal | Maximize margin | Maximize probability for true label |
| Interpretability | Less interpretable | Intuitive probabilistic meaning |
| Focus | Hard cases only | All predictions matter |

Softmax is often preferred when confidence estimation and probabilistic outputs are important.

## 4.7 Training and Hyperparameter Tuning

Training involves finding $W$ and $b$ that minimize the average loss over the dataset:

$$\mathcal{L}_{\text{total}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(x_i, y_i)$$

Important hyperparameters:

- Learning rate: step size in optimization
- Batch size: number of samples per update

- Regularization strength: penalizes large weights
- Number of epochs: how many passes over the dataset

Validation and test sets help measure generalization and prevent overfitting.

## 4.8 Summary

This case study has demonstrated how linear models and softmax-based classifiers can be used in image classification. Despite their simplicity, these methods form the basis of more advanced deep learning systems. Concepts like score functions, loss minimization, and probabilistic modeling generalize to complex architectures like CNNs and Transformers.