



The background features a faint, stylized neural network diagram. It consists of several layers of circular nodes connected by lines. The diagram is divided into three main sections by vertical lines: an input layer on the left, a central 'Hidden layers' section, and an output layer on the right. Weights are labeled on the connections, such as  $w_{11}^{(1)}$ ,  $w_{11}^{(2)}$ ,  $w_{11}^{(3)}$ , and  $w_{11}^{(4)}$ . The text 'Hidden layers' is at the top of the central section, and 'Visible output layer' is on the right. The title 'Neural Network Programming Fundamentals' is centered over the diagram.

# Neural Network Programming Fundamentals

Learn the essentials of implementing neural networks effectively



# Objective

## 🎓 Goal

Learn the **fundamentals** of neural network programming

## 🔧 Important Techniques

Implement neural networks effectively with:

### 📌 Vectorization

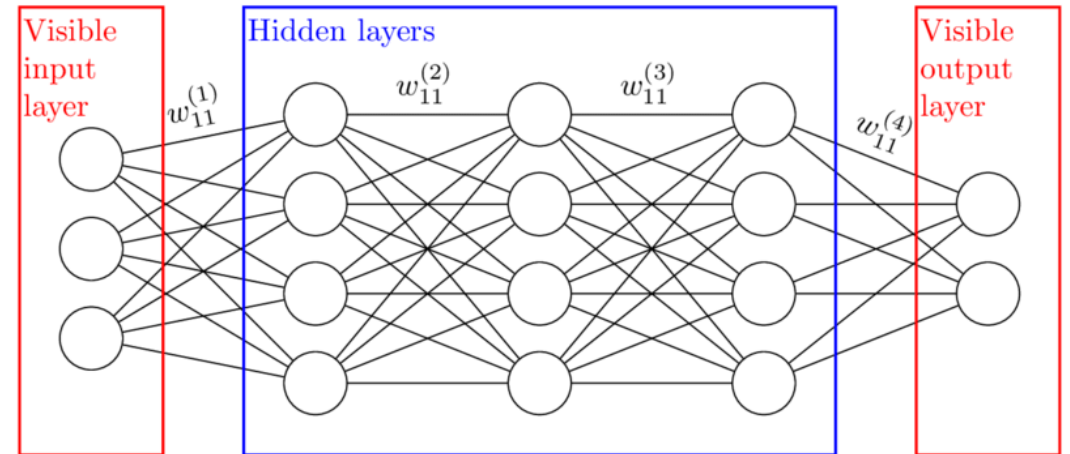
Avoid for-loops → Process entire training set at once

### ↔ Two-Phase Computation

- Forward Propagation
- Backward Propagation



Simplify concepts using **Logistic Regression** as an introductory model



# 🐾 Binary Classification Example

## 📋 Task

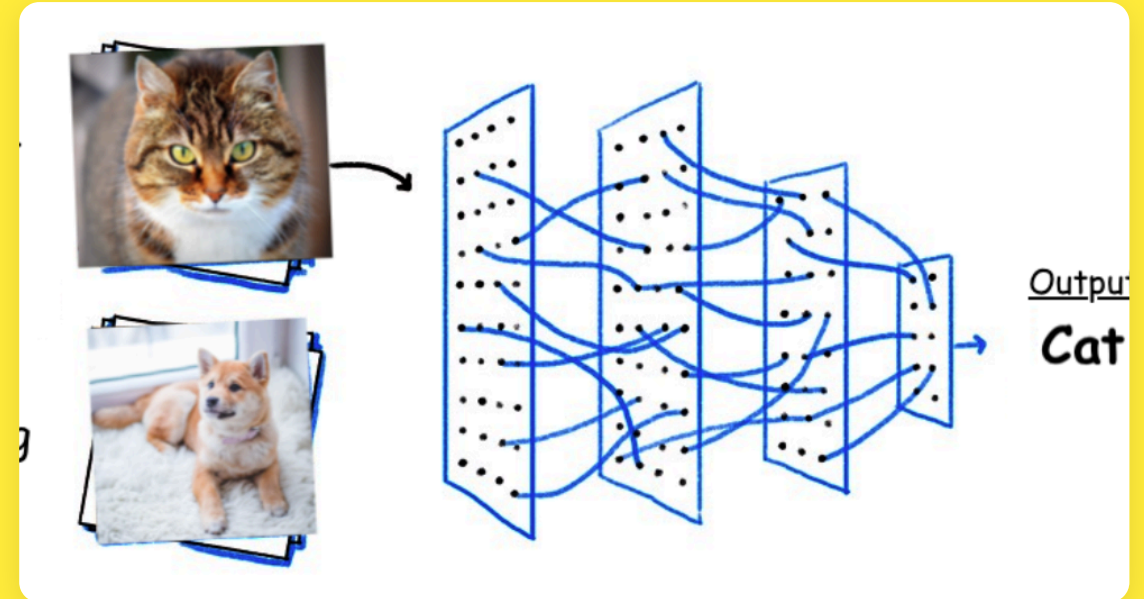
Classify an image as either **cat (1)** or **not cat (0)**

## ➡ Input (x)

A digital image represented as pixel values

## ➡ Output (y)

A value of **0** or **1** representing the class



**Cat**

**1**

**Not Cat**

**0**

# Representing an Image in a Computer

## Image Storage

An image is stored as **three matrices**, each representing a color channel:

### Red

R	R	R
R	R	R
R	R	R

### Green

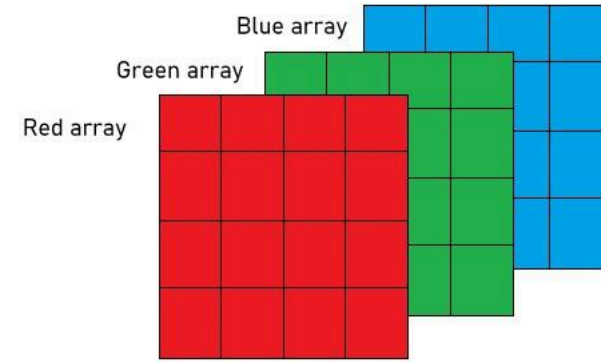
G	G	G
G	G	G
G	G	G

### Blue

B	B	B
B	B	B
B	B	B

## Feature Vector

The image is converted into a **long vector** by unrolling all pixel values



Arrays stacked over each other to form a Digital Image.

## Example: 64×64 Pixel Image

3 matrices of 64×64 pixels

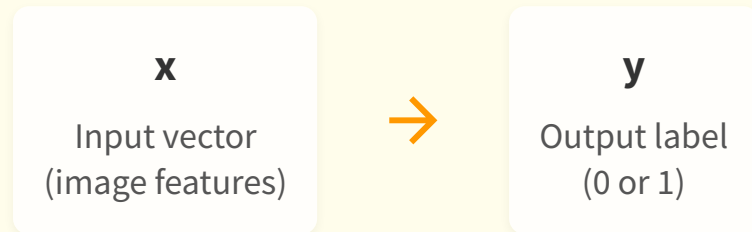
$$64 \times 64 \times 3 = 12,288$$

Number of input features (nx)

# 🏠 Preparing the Training Data

## 📋 Single Training Example

A training example consists of a pair  $(x, y)$ :



## 📁 Training Set

Contains **m examples**:

$(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)$

## Training Set Visualization

**Example 1**

$(x^1, y^1)$

**Example 2**

$(x^2, y^2)$

**Example 3**

$(x^3, y^3)$

...

**Example m**

$(x^m, y^m)$

## 📌 Key Point

Each example consists of an **input vector** and its corresponding **output label**

# Representing Data in Matrices

## Matrix Organization

To simplify programming and perform **efficient computations**:

### Input Matrix X

$x_1^1$	$x_1^2$	...	$x_1^m$
$x_2^1$	$x_2^2$	...	$x_2^m$
...	...	...	...
$x_{n \times 1}$	$x_{n \times 2}$	...	$x_{n \times m}$

Dimensions:  $(n \times m)$

Each column = one training example

### Output Matrix Y

$y^1$	$y^2$	...	$y^m$
-------	-------	-----	-------

Dimensions:  $(1 \times m)$

Each column = label corresponding to example in X

## Linear Algebra for Machine Learning

### Data Representation

#### Matrix and Vector Operations in Python

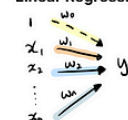
Scalar	Vector	Matrix	Tensor
scalar = 1 $\begin{bmatrix} 1 \end{bmatrix}$	np.array([1,2]) $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	np.array([[1,1],[2,2]]) $\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$	np.array([[[1,1],[2,2]], [[3,3],[4,4]]]) $\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 4 \end{bmatrix}$

Addition	Subtraction	Multiplication	Division	Dot Product
matrix1 + matrix2 $\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix}$	matrix2 - matrix1 $\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	matrix1 * matrix2 $\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 1 \times 2 & 1 \times 2 \\ 2 \times 2 & 2 \times 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 4 \end{bmatrix}$	matrix2 / matrix1 $\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} / \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 2/1 & 2/1 \\ 2/1 & 2/1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix}$	matrix1.dot(matrix2) $\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 1 \times 2 + 1 \times 2 & 1 \times 2 + 1 \times 2 \\ 2 \times 2 + 2 \times 2 & 2 \times 2 + 2 \times 2 \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 8 & 8 \end{bmatrix}$

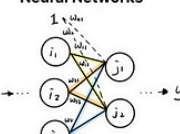
Reshape	Transpose	Inverse
matrix.reshape('row', 'col') $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix}$	matrix.T $\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$	np.linalg.inv(matrix) Identity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1/3 & 2/3 \\ 2/3 & -1/3 \end{bmatrix}$

#### Applications in Machine Learning

##### Linear Regression


$$\begin{bmatrix} 1 & x_1 & x_2 & \dots & x_n \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y \end{bmatrix}$$
$$y = 1 \cdot w_0 + x_1 w_1 + x_2 w_2 + \dots + x_n w_n$$

##### Neural Networks


$$\begin{bmatrix} 1 & i_1 & i_2 & i_3 \end{bmatrix} \cdot \begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} j_1 & j_2 \end{bmatrix}$$
$$j_1 = w_{01} + i_1 w_{11} + i_2 w_{21} + i_3 w_{31}$$
$$j_2 = w_{02} + i_1 w_{12} + i_2 w_{22} + i_3 w_{32}$$

## X Matrix

$n \times$  rows = features  
 $m$  columns = examples

## Y Matrix

1 row = labels  
 $m$  columns = examples

## Benefits



Faster computation



Clearer code



Vectorized operations



Extends to neural networks

# Standard Notation Used

**m**

## Number of Training Examples

Total examples in the dataset

**$n_x$**

## Number of Input Features

e.g., number of pixels in the image

**X**

## Input Matrix

Size: ( $n_x \times m$ )

**Y**

## Output Matrix

Size: ( $1 \times m$ )

### Notation Pattern

#### Single Example

$(x, y)$



#### Multiple Examples

$(X, Y)$

$(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m) \rightarrow (X, Y)$



### Extension to Neural Networks

This representation pattern is also used when building **multi-layer neural networks**, where:

- Each layer has its own matrices
- Dimensions follow consistent patterns
- Vectorized operations apply across layers



### Key Insight

Understanding this notation is **fundamental** to implementing neural networks efficiently