AI decoded: Demystifying the Buzzwords

from magic to math, for High School students

Guilherme Marcelo, University of Lisbon

December 2025

Meta AI: Tech

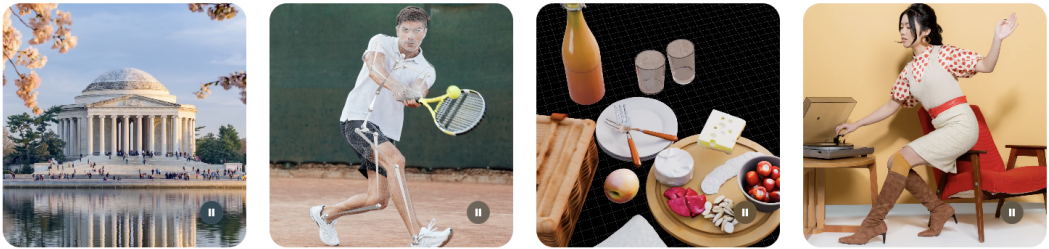


Figure: Meta AI: 3D SAM

Meta AI: Products

Figure: Meta AI: Hyperscape

The "Black Box" Problem

When we talk about AI, people imagine robots with consciousness.

But as an engineer, I see it differently:

One Definition

AI is simply a function $f(x;\theta)$ that maps an input to an output.

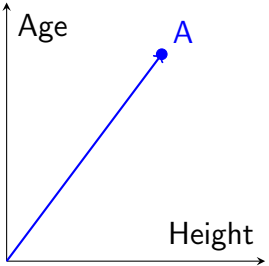
$$\text{Input (Image)} \xrightarrow{f(x;\theta)} \text{Output ("Cat")}$$

The "Learning" part? Finding the specific mathematical numbers inside θ that make $f(x;\theta)$ right most of the time.

Everything is a Vector

To a computer, the world is just a list of numbers. We call these **Vectors**.

- **You** are a vector:
  $\mathbf{v} = [\text{Height}, \text{Age}, \text{Grade}]$
- **An Image** is a vector:
  $\mathbf{x} = [\text{Pixel}_1, \text{Pixel}_2, \ldots, \text{Pixel}_N]$



Sparse vs. Dense Representations

How do we represent concepts like words?

1. Sparse (Old Way)

One dimension per word.

- "Cat": $[0, 1, 0, 0, \dots]$
- "Dog": $[0, 0, 1, 0, \dots]$

Problem: No relationship between words. Orthogonal and huge.

2. Dense (AI Way)

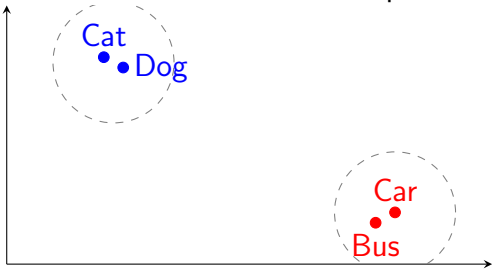
Learned coordinates in space.

- "Cat": $[0.9, 0.1, -0.5]$
- "Dog": $[0.8, 0.2, -0.4]$

Benefit: Encodes meaning. Similar concepts have similar numbers.

Semantic Properties in Vector Space

When AI learns dense representations, **meaning** becomes **geometry**.



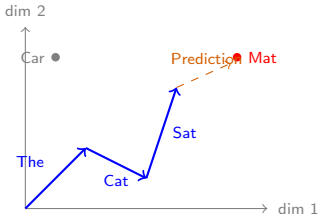
Ideally, minimizing loss pushes "Dog" closer to "Cat" and far from "Car".

The "Next Token": Demystifying ChatGPT

We saw that "Cat" is a vector.

What is ChatGPT? It is just a "Probability Machine" operating in this vector space.

$$P(\text{word}|\text{context})$$



Insight: It doesn't "know" facts; it predicts the most probable next step.

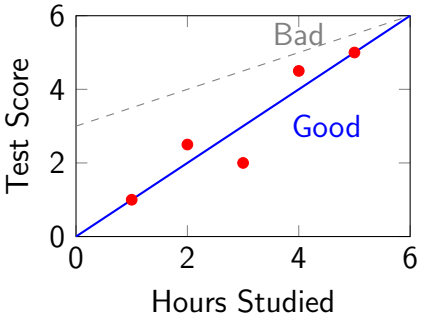
The Simplest Model: Linear Regression

Before we talk about massive brains, let's talk about a single line.

We want to find a line:

$$\hat{y} = wx + b$$

- w: Weight (Slope)
- b: Bias (Intercept)
- $\theta = \{w, b\}$ (Two params)



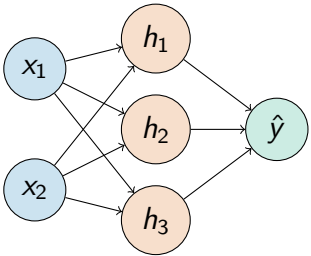
The Neural Network

If a linear regression is $\hat{y} = wx + b$, what is a Neural Network?

It is just **Layers** of linear regressions wrapped by an activation function, fed into each other.

- **1 Neuron:** A simple line.
- **ChatGPT:** Billions of lines interacting.

The math doesn't change. We just have more w's to find.



How do we know if the AI is "Smart"?

We measure how **wrong** it is.

- If the AI guesses $\hat{y}$ and the real answer is y, the error is the **distance** between them.
- In math, we love the "Squared Euclidean Distance":

$$\text{Loss} = \mathcal{L}(\theta) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$$

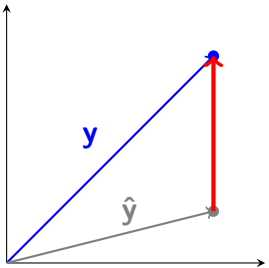
Goal: Minimize Loss to 0.

Visualizing the Error: Why Squared Norm?

What does $\|\hat{\mathbf{y}} - \mathbf{y}\|^2$ look like?
Let's say we are predicting a 1D point (e.g., exam score).

- **True Vector (y):** The actual scores stacked.
- **Prediction ($\hat{\mathbf{y}}$):** What our AI thinks the scores are.
- **Error vector:** The red arrow connecting them.

Minimizing the squared norm means shrinking that red arrow to zero length.



Intuition Check

Let's play a game. I give you points on a line: $2, 4, 9$.

I want you to pick **one single number** (θ) that is "closest" to all of them at once.

The Optimization Problem

Find θ that minimizes the total squared error:

$$\min_{\theta} \sum_{i=1}^{N} (\theta - x_i)^2$$

The Math Proof (Convex Functions)

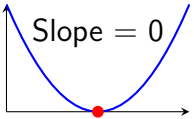
For simple shapes (Convex), we can just find the bottom analytically.

Loss Function: $\mathcal{L}(\theta) = \sum_{i=1}^{N} (\theta - x_i)^2$

Derivative: $\frac{d\mathcal{L}}{d\theta} = 2 \sum_{i=1}^{N} (\theta - x_i)$

Set to 0: $0 = \sum \theta - \sum x_i$

$$\implies N\theta = \sum x_i \implies \theta = \frac{1}{\mathbf{N}} \sum \mathbf{x_i}$$



Result: The parameter θ becomes the **Mean**.

But what if it's hard? (Non-Convex)

Deep Learning models are NOT simple bowls. They are complex landscapes.

How do we find the bottom if we can't just solve $\mathcal{L}'(x) = 0$?

- **Idea 1: Random Guessing?** Try a billion random numbers and keep the best one.
- *Why it fails:* In high dimensions (billion parameters), the chance of hitting the right spot is effectively zero.
- **Idea 2: Follow the Gradient (Slope).**

Gradient Descent: Going Downhill

Imagine you are blindfolded on a mountain. You want to reach the valley. You feel the ground with your foot.

- If the ground slopes **UP**, you step **BACK**.
- If the ground slopes **DOWN**, you step **FORWARD**.

Rule: Go against the slope.

$$w_{new} = w_{old} - \text{StepSize} \cdot \text{Slope}$$

$$w_{new} = w_{old} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_{old}}$$

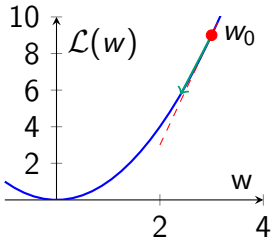
1D Example

Let's minimize $\mathcal{L}(w) = w^2$. Suppose we start at a bad guess: $w_0 = 3$.

1. Current position: $w_0 = 3$.
2. Calculate Slope $(2w)$: $2(3) = 6$.
3. Slope is positive $(+)$. We must go negative $(-)$.
4. Update:

$$w_{new} = 3 - (0.1)(6) = 2.4$$

We moved from 3 to 2.4. We are getting closer to 0!



The "Weights" of a Neural Network

- When you hear "This Neural Network has 7 Billion Parameters"...
- It just means it has 7 Billion w's.

$$\min_{\theta} \|f(X;\theta) - Y\|^2 = \min_{\theta} \|\hat{Y} - Y\|^2$$

Training an AI is just calculating the slope (Gradient) for all 7 billion weights and nudging them slightly downhill, over and over again.

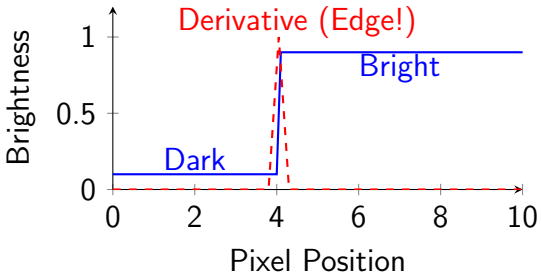
How do Computers "See"?

We learned that images are just matrices of numbers. We learned that we can optimize functions. Now let's apply this to vision.

Question: How do we find an "Edge" (like the side of a table)?

Answer: We look for rapid changes in color.

- Change in Math = **Derivative**.

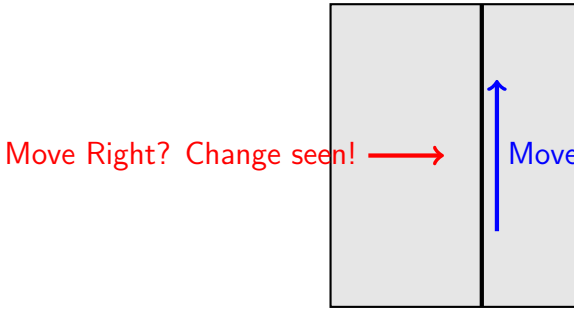


The Problem of Motion (The Aperture Problem)

Finding edges is great, but can we track how the camera moves?

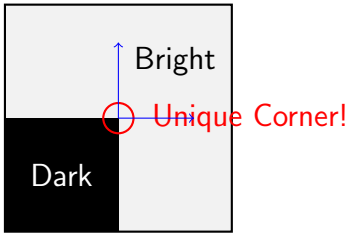
Imagine looking at a white wall with a black line.

- If I move the camera **Along** the line... the image looks identical!
- The gradient (derivative) didn't change, even though we moved.
- **Conclusion:** Edges are ambiguous.



The Solution: Keypoints (Corners)

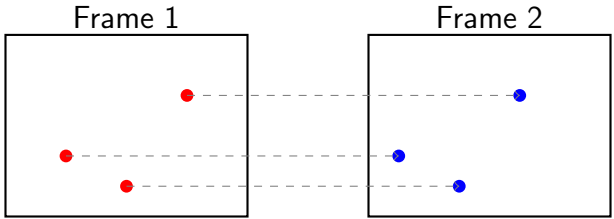
To solve the puzzle, we need points that are unique in ALL directions. We call these **Keypoints** (or Corners).



Change in ANY direction

Understanding the World via Matching

Matching $\rightarrow$ we can calculate how the camera moved.



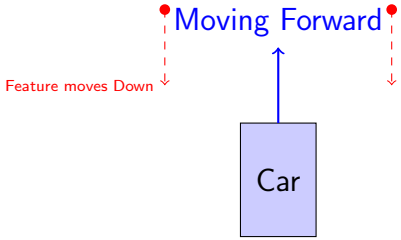
Logic: Points moved LEFT $\Longrightarrow$ Camera moved RIGHT.

From "Optical Flow" to Self-Driving Cars

Why do we care about "Corners" and matching them?

- **The Logic:** A Tesla uses these corners to triangulate its own position.
- If the corner moves **LEFT** in the image...
- ...the car knows it moved **RIGHT**.

By tracking thousands of static corners, the car builds a map of the world and finds its place in it.



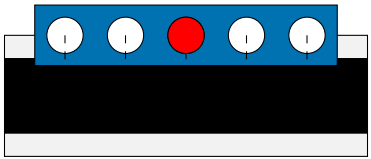
Relative Motion:
Features go back → Car goes forward

Case Study: How to make a Robot Fast?

Let's combine everything: Vectors, Functions, and Optimization.

The Setup: A robot with 5 Infrared (IR) sensors looking at the floor.

- **Sensor sees Black (Line):** 1
- **Sensor sees White (Floor):** 0



The Input: Another Vector

Just like an image or a word, the sensor readings are a **Vector**.

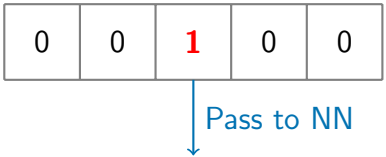
State Vector x:

$$x = [S_1, S_2, S_3, S_4, S_5]$$

Robot Brain Input

Example States:

- Center: $[0, 0, \mathbf{1}, 0, 0]$
- Left: $[0, \mathbf{1}, \mathbf{1}, 0, 0]$
- Way off: $[\mathbf{1}, 0, 0, 0, 0]$



Redefining Output: Differential Drive

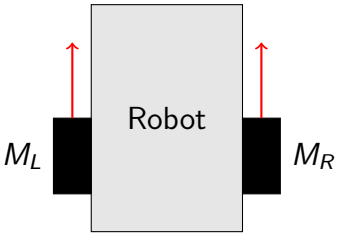
We don't just steer; we control two motors independently.

Output Vector y:

$$y = [M_{Left}, M_{Right}]$$

$$M \in [0, 255]$$

- **Go Straight:** $[255, 255]$
- **Turn Left:** $[50, 255]$
- **Turn Right:** $[255, 50]$



Traditional Coding vs. Machine Learning

Traditional (Rule-Based):

- `IF center is 1 THEN go straight.`
- `IF left is 1 THEN turn left.`

Problem: It's jerky. It reacts, it doesn't flow.

Machine Learning (Regression): We want a function that outputs a **smooth motor speeds**.

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

The Secret to Speed: Behavioral Cloning

How do we find the parameters θ for speed? **We cheat. We use a human teacher.**

1. **Data Collection:** A human drives the robot via remote control as fast as possible.
2. **Record:** We save the pairs:

$$(\text{Sensor Vector}, \text{Motor Speeds})$$

3. **Train:**

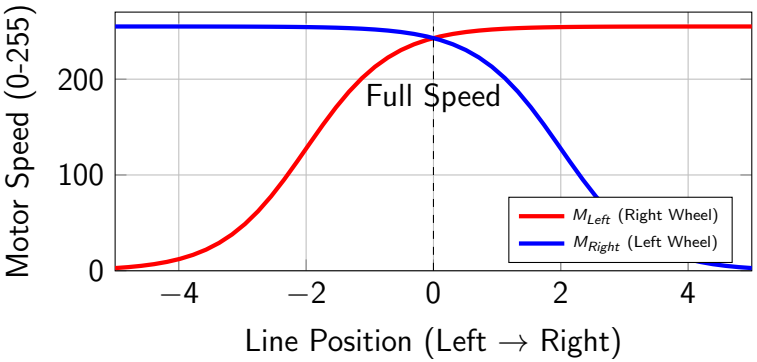
$$\min_{\theta} \mathcal{L}(\theta) = \min_{\theta} \|\hat{Y} - Y_{\text{human}}\|^2$$

Table: Training data from Human driver

x	y
$[0, 0, 1, 0, 0]$	$(255, 255)$
$[0, 1, 1, 0, 0]$	$(100, 180)$
$[1, 0, 0, 0, 0]$	$(0, 255)$
$[0, 0, 0, 1, 1]$	$(180, 100)$

The Result: Learned Differential Steering

The Neural Network learns to mix the motors smoothly.



Conclusion: Demystified

We started with "AI Magic" and ended with math.

1. **Representation:** Everything is a vector. (Images, Words, Sensors).
2. **Model:** AI is just a function $f(x)$ with tunable params (θ).
3. **Optimization:** Learning is just sliding down a hill to minimize error.
4. **Application:** Whether it's ChatGPT or a Line Follower, the recipe is the same.

Don't be afraid of the Black Box.
It's just math.

Thank You!

Questions?