Content

30 July 2025 08:52

ML vs DL Basics

30 July 2025 17:39

🔍 What is Machine Learning (ML)?

- ML is a way to **teach machines using data** and some **rules**.
- It learns patterns from past data and uses them to make predictions.
  Analogy: ML is like teaching a kid to recognize fruits. You tell them: 'Mango is yellow, oval, sweet'. The kid remembers these features.

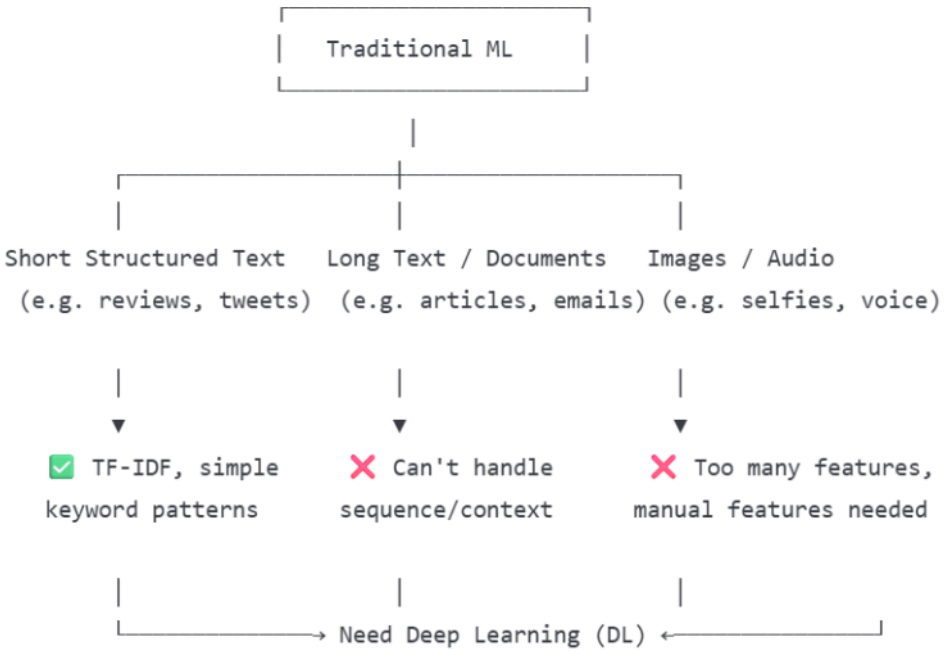
🤖 What is Deep Learning (DL)?

- DL is a **type of ML** that uses **neural networks** to learn **on its own** from large amounts of data — no need to hand-pick features.
  🧠 **Analogy**: It's like giving a kid **thousands of fruit images**, and without explaining anything, they figure out on their own what a mango or apple looks like.

🧬 Why Did We Need DL? (Birth of DL)

Traditional ML struggled when:

- Data became too large or complex (images, audio, language)
- Feature selection became difficult
- Accuracy plateaued on big data



🧠 Analogy to Explain

Traditional ML is like trying to judge a book by how many times each word appears, while **Deep Learning** actually *reads* the book and understands the story.

❌ Traditional ML struggles with:

- **Unstructured + High-dimensional data**
  - **Images**: Every pixel is a feature (a 224x224 image = ~50,000+ features!)
  - **Audio**: Thousands of time-steps per second
  - **Language**: Long sentences, context, grammar, semantics

Why it **fails**:

1. **Manual feature extraction** is hard
   - You'd have to define what features make a "cat" in an image (ear shape? fur pattern?)
   - In long text, simple word counts can't capture **meaning or context**
2. **Too many features, too little generalization**
   - Classical ML (like SVM, Decision Trees) overfits or breaks with **high-dimensional input**
3. **Context and sequence are ignored**
   - ML doesn't understand that in language, **"not good" ≠ "good"**

🔥 **DL was born because:**

- More **data** became available (Big Data)
- **GPUs** made faster training possible
- Need to understand **unstructured data** (images, audio, text)

📊 **Core Comparison: ML vs DL**

Feature	Machine Learning (ML)	Deep Learning (DL)
Data Dependency	Works well on **small/medium data**	Needs **large data** to perform well
Hardware Dependency	Can run on **CPU**	Needs **GPU/TPU** for speed
Training Time	**Faster training** (minutes to hours)	**Slower training** (hours to days)
Feature Selection	Needs **manual feature engineering**	Learns **features automatically**
Interpretability	**Easy to explain** (e.g. Decision Trees)	**Hard to interpret** (black box)

🧠 **Example to Compare**

Problem: You want to classify whether a person is happy or sad from a photo.

ML	DL
You extract features: mouth curve, eye openness	You just give thousands of happy/sad photos
You feed those features to ML model	DL learns features by itself from pixels

🎯 **Why Start from DL for LLMs?**

LLMs (like GPT, BERT) = **Large Deep Learning Models** trained on **massive data** using **neural networks**.

☑ You need to know DL because:

- LLMs = **deep neural networks** with **billions of parameters**
- They use **layers of attention**, not manual rules
- ML is not enough for **language generation, understanding context, translation, etc.**

☑ **ML: Benefits and Limitations**

Benefits of ML

- Fast to train
- Easy to understand (interpretability)

- Good for **structured data** (Excel-like data)

Limitations

- Doesn't work well on images, audio, or raw text
- Needs **feature engineering**
- Lower performance on **large unstructured data**

🚀 **What DL Solves**

☑ DL Benefits

- Works on **images, text, voice, video**
- No need for feature engineering
- Learns complex patterns
- Powers LLMs, self-driving, voice assistants, etc.

🚫 DL Limitations

- Needs **lots of data**
- Requires **expensive hardware**
- Hard to debug or explain decisions

🌍 Use Cases

Task	ML	DL
Predict House Prices	☑	☑
Face Recognition	❌	☑
Chatbots / LLMs	❌	☑
Fraud Detection	☑	☑
Language Translation	❌	☑

🩹 **Fun Mini Exercise**

Activity:

Show 5 pictures (cat, dog, mango, car, face) — ask:

"How would you tell a machine to recognize this using ML?"

"How would DL handle it differently?"

Let them explain how they'd extract features manually (ML) vs just give images (DL).

Important Links:

https://teachablemachine.withgoogle.com/train

https://playground.tensorflow.org/

Important Terms

31 July 2025 22:33

1. Weights and Biases

- **Weights** are the parameters in a neural network that determine the strength of the connection between neurons.
- **Bias** is an additional parameter that helps the model fit the data better by shifting the activation function.
- They are both **learnable parameters**—meaning the model adjusts them during training.
- Think of **weights as slopes** and **bias as the y-intercept** in a line equation.
- Without biases, your model becomes less flexible and may not fit certain patterns well.

2. Why Bias is Important

- Bias lets the activation function **shift left/right** instead of being fixed at the origin.
- It helps the model learn patterns **even when input is zero**.
- Without bias, models may fail to learn simple relationships.
- It makes the model **more expressive** and capable of fitting real-world data better.

3. Forward Pass

- Input data passes through the layers of the neural network.
- Each layer computes a value: output = activation(weight * input + bias)
- These outputs move forward until the final prediction is made.
- It's how the model **makes predictions** before learning (inference).

4. Backward Pass (Backpropagation)

- After the forward pass, the model **calculates the error** using a loss function.
- Then, it goes **backward through the network**, layer by layer.
- It computes **gradients** (derivatives) of the loss w.r.t. weights and biases.
- These gradients are used to **update the weights/biases** using an optimizer (like SGD, Adam).

5. How Weights Change

- Weights change during the **backward pass** using gradient descent.
- The formula:
  new_weight = old_weight - learning_rate * gradient
- This process reduces the error and improves predictions.
- Every time data is passed and backpropagated, weights are adjusted.

6. Are Weights Updated Every Epoch?

- Yes, weights **are updated during each epoch**, but also after each **batch**.
- If you use **batch gradient descent**, weights update **once per epoch**.
- If you use **mini-batch or stochastic gradient descent**, weights update **multiple times per epoch**.
- So generally, **yes**, weights update in every epoch (often multiple times).

7. What is an Epoch?

- One **epoch** = one full pass through the **entire training dataset**.
- If you have 1000 training samples and use batch size 100, then 1 epoch = 10 batches.
- You typically train a model for **many epochs** to let it learn better.
- More epochs = more learning, but too many may cause **overfitting**.

Bias

06 August 2025 16:41

1 What is Bias in Simple Words?

- **Bias** is like a fixed extra number you add to the total input of a neuron (perceptron) before deciding the output.
- It **shifts the decision boundary** — meaning it changes *when* the perceptron says "Yes" (1) or "No" (0).

Formula without bias:
Output=f(w1x1+w2x2)\text{Output} = f(w_1x_1 + w_2x_2)
Formula with bias:
Output=f(w1x1+w2x2+b)\text{Output} = f(w_1x_1 + w_2x_2 + b)
That **+ b** is the bias.

2 Analogy

Imagine you are a **judge in a singing competition**:

- **Inputs** = singer's performance factors (voice, stage presence, song choice).
- **Weights** = how important each factor is to you.
- **Bias** = your **mood** before the singer even starts.

💡 If you woke up happy (big positive bias) → you're *easier to impress*.
💡 If you woke up grumpy (negative bias) → the singer has to work *extra hard* to get your "YES".
So **bias sets the base level** before any input comes in.

3 Why is Bias Needed?

Without bias, the perceptron's decision **always depends purely on inputs and their weights**.
But sometimes:

- We want the model to say "YES" **even when inputs are small** → positive bias helps.
- Or require **stronger inputs to say YES** → negative bias helps.

4 Importance

- **Flexibility** – The neuron can activate even without strong inputs.
- **Shifting Threshold** – It moves the activation point left or right.
- **Better Learning** – Allows the model to fit data that doesn't pass through the origin (0,0).

📌 **Super Short Summary**:
Bias is like a ***preloaded opinion*** of a perceptron — it makes it easier or harder to say "YES" regardless of inputs, giving flexibility in decision-making.

Learning rate

06 August 2025 21:53

1️⃣ Simple Definition

The **learning rate** (η\etaη) is a number that controls **how big a step** we take when updating weights during gradient descent.
It's like the speed dial of learning — too fast or too slow can ruin the training.

2️⃣ Analogy

Imagine walking down a hill to reach the lowest point (minimum loss):

- **Small steps** (small learning rate) → very safe, but takes forever.
- **Huge steps** (large learning rate) → you might overshoot and keep bouncing around, never settling at the bottom.
- **Just right** → you smoothly reach the lowest point quickly.

3️⃣ Math

Weight update formula:

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

- η = learning rate.
- $\frac{\partial L}{\partial w}$ = gradient (direction & slope).
- If η is too large → unstable learning.
- If η is too small → very slow learning.

Gradient Descent

06 August 2025 21:46

1 Simple Idea

Gradient Descent is a method to **find the best parameters** (weights & biases) that make your neural network's predictions as accurate as possible by **reducing the error (loss)** step-by-step.

2 Analogy

Imagine you're **standing on top of a hill** in dense fog.
Your goal: **reach the lowest point in the valley** (minimum loss).

- You can't see the whole valley (you don't know the best weights immediately).
- You feel the slope beneath your feet (this is the **gradient**).
- You take small steps **downhill** in the steepest direction (update weights).
- If your steps are too big (**large learning rate**), you might overshoot.
- If too small, it will take forever to reach the bottom.

3 Math Behind It

We want to **minimize** the loss function $L(w)$

where w = model parameters (weights).

At each step:

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

- $\frac{\partial L}{\partial w}$ → slope of the loss curve at current weight (gradient).
- η → learning rate (how big the step is).
- Subtract because we want to go **downhill** (toward smaller loss).

4 Example

Let's say:

$$L(w) = (w - 5)^2$$

The minimum is at $w = 5$.

If current $w = 0$,
Gradient:

$$\frac{\partial L}{\partial w} = 2(w - 5) = 2(0 - 5) = -10$$

Update step (with $\eta = 0.1$):

$$w_{\text{new}} = 0 - 0.1 \cdot (-10) = 0 + 1 = 1$$

Now we're **closer to 5**, and we keep going until loss is minimal.

5 In Neural Networks

- **Forward pass**: compute predictions & loss.
- **Backward pass**: compute gradients for each weight.
- **Gradient descent step**: update each weight with
- $$w = w - \eta \cdot \frac{\partial L}{\partial w}$$
- Repeat for many epochs.

Back-Propagation

06 August 2025 21:50

Why It's Important

Without backpropagation, the network wouldn't know **which weights to change** or **how much to change them**.

It's the backbone of training deep learning models efficiently.

Goal: Find $\frac{\partial L}{\partial w}$ — how much the loss changes if w changes a tiny bit.

Step-by-step with Chain Rule:

For output layer:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

- $\frac{\partial L}{\partial a}$ → how output affects loss.
- $\frac{\partial a}{\partial z}$ → how activation changes with input z.
- $\frac{\partial z}{\partial w}$ → how weight affects z.

Activation function

06 August 2025 21:56

1 Why Activation Functions?

- They introduce **non-linearity** so neural networks can learn complex patterns.

Types:

A) Sigmoid

Equation:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Range: (0, 1)

Shape: S-curve

Graph behavior:

- Small x → output near 0
- Large x → output near 1
- Centered at $x = 0$ → output = 0.5

Pros: Smooth, interpretable as probability.

Cons: Vanishing gradients for large |x|. ????

B) Tanh (Hyperbolic Tangent)

Equation:

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

Range: (-1, 1)

Shape: S-curve centered at zero

Pros: Centered around 0 (faster learning than sigmoid).

Cons: Still suffers from vanishing gradients.

C) ReLU (Rectified Linear Unit)

Equation:

$$f(x) = \max(0, x)$$

Range: [0, ∞)

Graph behavior:

- Negative x → output 0
- Positive x → output x

Pros: Simple, efficient, solves vanishing gradient for positive side.

Cons: Can cause "dead neurons" if weights make input negative forever.

D) Linear (Identity)

Equation:

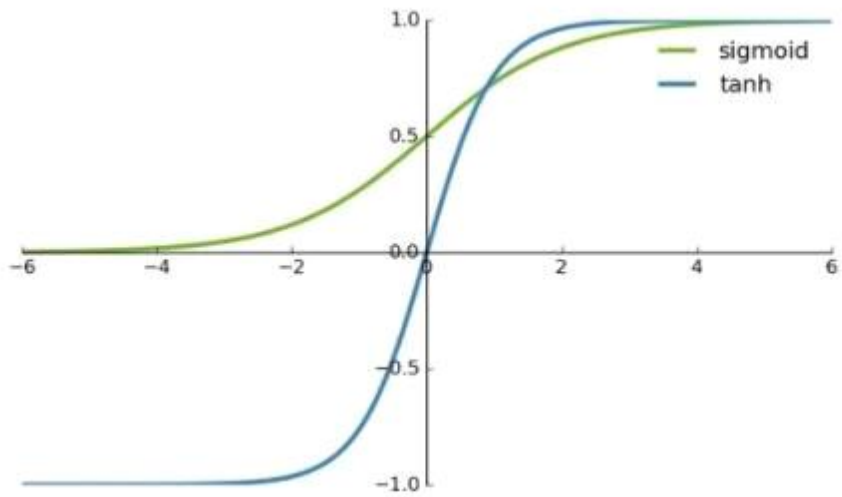
$$f(x) = x$$

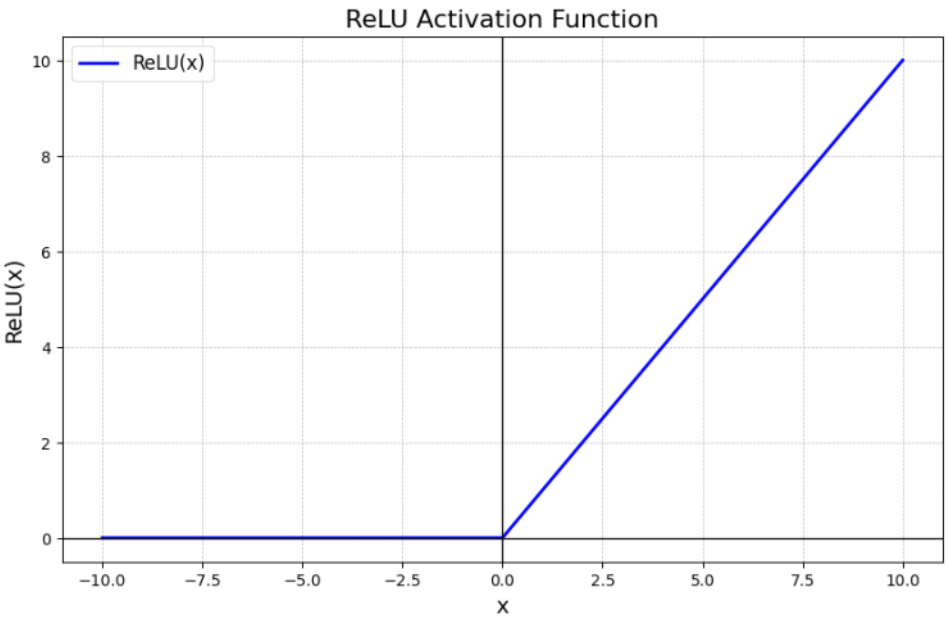
Range: (-∞, ∞)

Graph behavior: Straight line

Pros: No distortion, useful for regression outputs.

Cons: No non-linearity, so no complex pattern learning if used in hidden layers.




ReLU Activation Function
ReLU(x)
ReLU(x)
x
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Optimizer

06 August 2025 21:59

3️⃣ How It Works

- Takes **gradients** from backpropagation (the slope info).
- Uses a strategy (e.g., simple Gradient Descent, Adam, RMSProp) to update weights.
- Balances **speed** (learning rate) and **stability** (not overshooting).

General update rule:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot (\text{adjustment based on optimizer algorithm})$$

4️⃣ Examples of Optimizers

- **SGD (Stochastic Gradient Descent)** → Simple downhill steps.
- **Momentum** → Adds speed in a consistent direction.
- **Adam** → Adapts step size for each weight automatically.
- **RMSProp** → Adjusts learning rate based on recent gradients.

1️⃣ What is an Optimizer?

An **optimizer** is the part of training that decides **how to adjust the weights and biases** in a neural network so the model makes better predictions.

It's like the **coach** telling a player exactly how to improve after every game.

Exercises

06 August 2025 08:30

1. Create a straight line dataset using the linear regression formula (`weight * X + bias`).
- Set `weight=0.3` and `bias=0.9` there should be at least 100 datapoints total.
- Split the data into 80% training, 20% testing.
- Plot the training and testing data so it becomes visual.
2. Build a PyTorch model by subclassing `nn.Module`.
- Inside should be a randomly initialized `nn.Parameter()` with `requires_grad=True`, one for `weights` and one for `bias`.
- Implement the `forward()` method to compute the linear regression function you used to create the dataset in 1.
- Once you've constructed the model, make an instance of it and check its `state_dict()`.
- Note: If you'd like to use `nn.Linear()` instead of `nn.Parameter()` you can.
3. Create a loss function and optimizer using `nn.L1Loss()` and `torch.optim.SGD(params, lr)` respectively.
- Set the learning rate of the optimizer to be 0.01 and the parameters to optimize should be the model parameters from the model you created in 2.
- Write a training loop to perform the appropriate training steps for 300 epochs.
- The training loop should test the model on the test dataset every 20 epochs.
4. Make predictions with the trained model on the test data.
- Visualize these predictions against the original training and testing data (note: you may need to make sure the predictions are *not* on the GPU if you want to use non-CUDA-enabled libraries such as matplotlib to plot).
5. Save your trained model's `state_dict()` to file.
- Create a new instance of your model class you made in 2. and load in the `state_dict()` you just saved to it.
- Perform predictions on your test data with the loaded model and confirm they match the original model predictions from 4.>

6. Without optimizer plot vs with optimizer plot (experiment)

Links

07 August 2025 00:17

https://claude.ai/public/artifacts/69fbdf8e-cd84-4b5f-a8b6-31f7cb588f6f?fullscreen=true

https://playground.tensorflow.org/#activation=sigmoid&regularization=L1&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.001&regularizationRate=0.001&noise=0&networkShape=3,3,2,2,2,2&seed=0.36083&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=true&ySquared=true&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

https://app.eraser.io/workspace/uQJMRzpJmMo7T21L3hTq?origin=share

https://docs.google.com/presentation/d/1SKs31BP5UwQ1W-8lOdD786QOQhSvZ4DP_2Qdx4MxzHo/edit?usp=sharing

https://x.com/mrdbourke/status/1501330041183121420

https://codeshare.io/G71dnn (optimizer)