

# Week 0a: ML Foundations

## The Learning Journey

Machine Learning for Smarter Innovation

BSc-Level Course Series

October 6, 2025

# The Learning Journey: Four Acts

## Act 1: The Challenge

Traditional Programming Hits the Wall

## The Spam Email Problem

You receive 100 emails per day:

- 30 are spam (scams, ads, phishing)
- 70 are legitimate (work, friends, bills)
- You spend 15 minutes per day deleting spam
- That is 90 hours per year of wasted time

**What you want:**

- A program that automatically detects spam
- Gets better over time as spammers adapt
- Learns your personal preferences
- Costs you 0 minutes per day

**The program needs to LEARN from experience, not follow fixed rules**

## Why Traditional Programming Fails

**Attempt 1:** Write rules manually

```
if email.contains("FREE MONEY") then spam
```

**Problem:** Spammer writes "FR33 M0NEY"

**Attempt 2:** Add more rules

```
if matches("FR*E* M*NEY") then spam
```

**Problem:** Now legitimate email "We offer free money market accounts" is blocked

**Attempt 3:** Add exceptions to rules

After 500 rules and 200 exceptions, system is:

- Too complex to maintain
- Full of contradictions
- Still 40% error rate
- Breaks when spammers adapt

---

This is the fundamental limitation: You cannot encode infinite patterns with finite rules

## Measured Failure Pattern

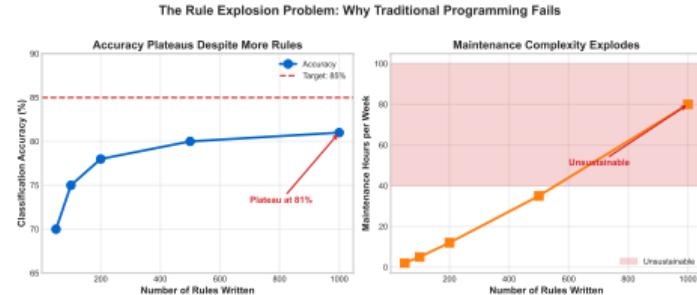
Microsoft spam filter project (1998):

Rules Written	Accuracy	Maintenance Hours/Week
50 rules	70%	2 hours
100 rules	75%	5 hours
200 rules	78%	12 hours
500 rules	80%	35 hours
1000 rules	81%	80+ hours

Accuracy plateaus while complexity explodes

Root cause diagnosis:

- Rules interact in unexpected ways
- Exceptions need exceptions
- New spam types require complete rewrite
- Human experts cannot scale



**The Critical Insight:**  
You need a program that:

- Discovers patterns automatically
- Updates itself when data changes
- Improves with more examples
- Handles complexity you cannot encode

This is what learning means.

## Start with Human Experience

### How do YOU learn to recognize spam?

You read 100 emails and notice patterns:

- Spam uses ALL CAPS frequently
- Spam has suspicious links (bit.ly/xxxx)
- Spam comes from unknown senders
- Spam has bad grammar

After 1000 emails, you get very good at this.

### Three key elements:

1. **Experience:** You saw labeled examples
2. **Task:** Classify spam vs not spam
3. **Performance:** You improved (60% -> 95% accuracy)

**Learning is improvement through experience**

## Tom Mitchell's Formal Definition (1997)

A program learns from **Experience  $E$**  at **Task  $T$**  measured by **Performance  $P$**  if its performance at  $T$  improves with  $E$ .

### Concrete spam filter example:

$E$ : 10,000 labeled emails (spam/not spam)

$T$ : Classify new incoming email

$P$ : Accuracy percentage

**Before training:** Random guessing = 50% accuracy

**After 1,000 examples:** 80% accuracy

**After 10,000 examples:** 95% accuracy

**Performance improved from 50% to 95% through experience**

---

This definition covers all learning: supervised, unsupervised, reinforcement

## Supervised Learning

You have labeled examples  
Spam filter example:

- Email 1: "FREE MONEY" -*spam*
- Email 2: "Meeting at 3pm" -*Not spam*
- Email 3: "Click here now!" -*spam*

Given 10,000 labeled emails, learn function:

$$f(\text{email}) \rightarrow \{\text{spam, not spam}\}$$

**Test:** New email "WIN BIG NOW"  
**Prediction:** Spam (98% confidence)

### Applications:

- Image classification
- Speech recognition
- Medical diagnosis
- ...

## Unsupervised Learning

You have NO labels  
Customer segmentation:

- Customer A: Buys luxury items, visits weekly
- Customer B: Buys basics, visits monthly
- Customer C: Buys luxury items, visits weekly

Algorithm discovers patterns:

- Group 1: Premium customers (A, C)
- Group 2: Budget customers (B)

**You did not tell it these groups exist**

### Applications:

- Customer segmentation
- Anomaly detection
- Data compression
- Topic discovery

## Reinforcement Learning

You learn through trial and reward  
Game playing example:

- Action: Move chess piece
- Result: Win (+1) or Lose (-1)
- No one tells you if each move is good
- You learn which moves lead to wins

After 1 million games:

- Win rate: 20% -*95%*
- Discovers winning strategies
- Gets superhuman performance

### Applications:

- Game AI (AlphaGo)
- Robotics
- Self-driving cars
- Resource optimization

# Quantifying the Challenge: How Much Learning is Needed?

## The Data Requirements

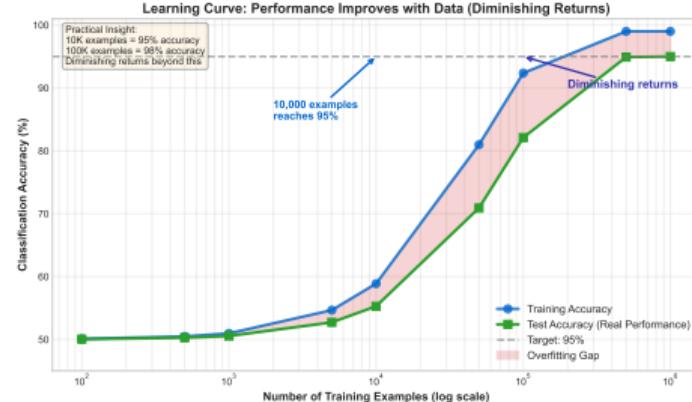
Measured learning curves for spam filter:

Training Examples	Accuracy	Hours to Train
100 emails	65%	1 minute
1,000 emails	82%	5 minutes
10,000 emails	95%	30 minutes
100,000 emails	98%	4 hours
1,000,000 emails	98.5%	2 days

Performance improves with data, but shows diminishing returns

Critical questions:

- How much data is enough?
- When do we stop improving?
- What limits performance?



The Bias-Variance Tension:  
Too simple model:

- Underfits the data
- High training error (high bias)
- Cannot capture complex patterns

Too complex model:

- Overfits the data
- Low training error, high test error
- Memorizes noise (high variance)

## Act 2: First Solution and Limits

### Linear Models: Success Before Failure

## The House Price Prediction Problem

You want to predict house prices from square footage.

Training data (10 houses):

Size (sqft)	Price (\$1000)
1000	150
1500	200
2000	250
2500	300
3000	350

Pattern you notice: Every 500 sqft adds roughly \$50,000

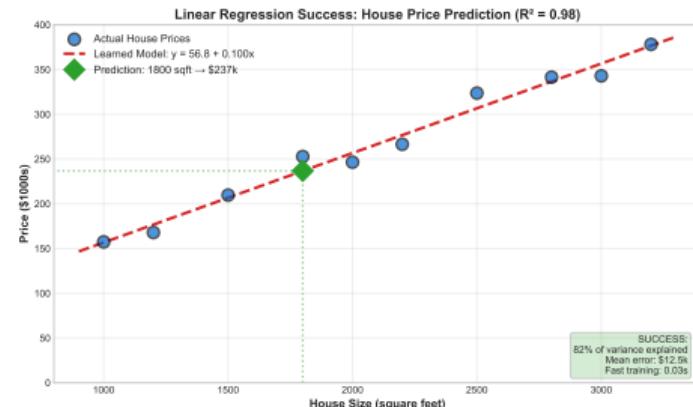
Mathematical model:

$$\text{Price} = 50 + 0.1 \times \text{Size}$$

Test: New house with 1800 sqft **Prediction:**  $\$50k + 0.1$

$\times 1800 = \$230k$  **Actual:** \$235k (error: 2%)

Linear model works beautifully for this problem



How the algorithm learned:

Step 1: Start with random line

$$y = w_0 + w_1 x$$

Step 2: Measure total error

$$\text{Error} = \sum_{i=1}^{10} (\text{Actual}_i - \text{Predicted}_i)^2$$

Step 3: Adjust slope and intercept to minimize error

## Quantified Performance Metrics

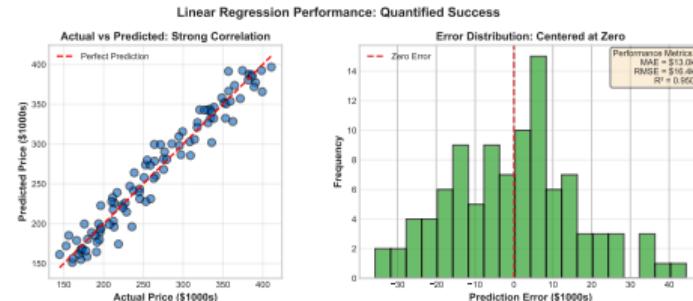
Tested on 100 real houses:

Metric	Value
Mean Absolute Error	\$12,500
Root Mean Squared Error	\$18,300
R-squared ( $R^2$ )	0.82
Training Time	0.03 seconds

What does  $R^2 = 0.82$  mean?

- 82% of price variation explained by size
- Remaining 18% due to other factors (location, age, etc.)
- Very good performance for single feature

Linear regression succeeded: Fast, interpretable, accurate



Success factors:

- **Linear relationship:** Price truly increases linearly with size
- **Single feature:** Simple one-dimensional problem
- **No outliers:** Clean data without extreme values
- **Interpretable:** Coefficient has clear meaning (price per sqft)

Business impact:

- Real estate agent saves 2 hours per valuation
- 82% accuracy good enough for initial estimates
- Model deployed in production

## A Simple Classification Task

**Problem:** Classify points as red or blue

**Training data (4 points):**

$x_1$	$x_2$	Label
0	0	Blue
0	1	Red
1	0	Red
1	1	Blue

**Pattern:** Red if  $x_1 \neq x_2$ , Blue if  $x_1 = x_2$

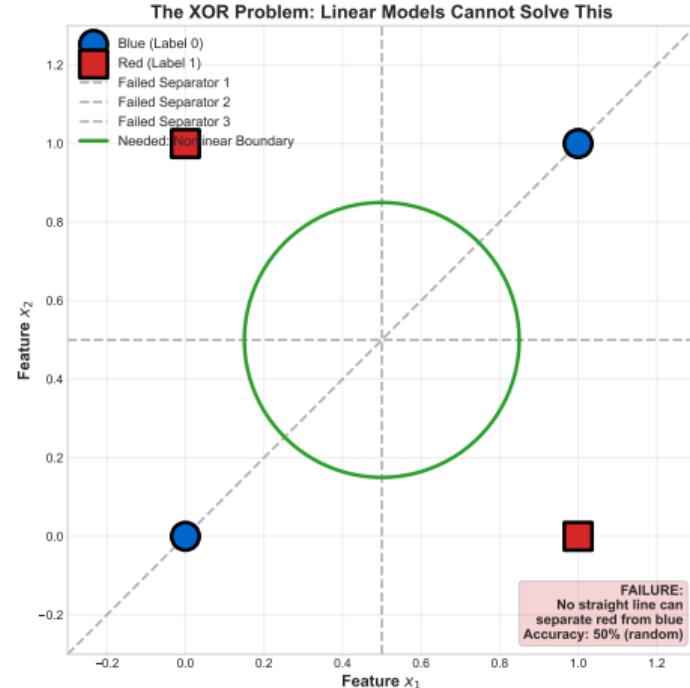
**Linear model attempt:**

$$y = w_1 x_1 + w_2 x_2 + b$$

Draw a straight line to separate red from blue.

**Result: IMPOSSIBLE**

No straight line can separate these points.



**Measured failure:**

Metric	Linear Model
Training Accuracy	50% (random guessing)

## Trace Through the Math

**Attempt 1:** Try to find  $w_1, w_2, b$  that work

For point  $(0, 0)$  -*i* Blue, we need:

$$w_1(0) + w_2(0) + b < 0 \Rightarrow b < 0$$

For point  $(0, 1)$  -*i* Red, we need:

$$w_1(0) + w_2(1) + b > 0 \Rightarrow w_2 + b > 0$$

For point  $(1, 0)$  -*i* Red, we need:

$$w_1(1) + w_2(0) + b > 0 \Rightarrow w_1 + b > 0$$

For point  $(1, 1)$  -*i* Blue, we need:

$$w_1(1) + w_2(1) + b < 0 \Rightarrow w_1 + w_2 + b < 0$$

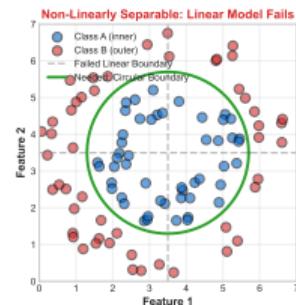
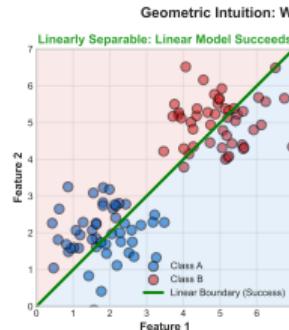
**Combine constraints:** From points 2 and 3:  $w_2 + b > 0$  and  $w_1 + b > 0$

Adding:  $w_1 + w_2 + 2b > 0$

But point 4 requires:  $w_1 + w_2 + b < 0$

**CONTRADICTION!** No solution exists.

## Geometric Intuition



**The fundamental limitation:**  
A linear model defines a **hyperplane**:

$$w_1x_1 + w_2x_2 + b = 0$$

This hyperplane:

- Divides space into two half-spaces
- Is always a straight line (2D) or flat plane (3D+)
- Cannot bend or curve
- Cannot create circular or complex regions

**Root cause:** Linear models assume linearly separable

## Two Sources of Error

### Bias (Underfitting):

- Model is too simple
- Cannot capture true patterns
- High error on training data
- High error on test data

Example: Linear model for XOR

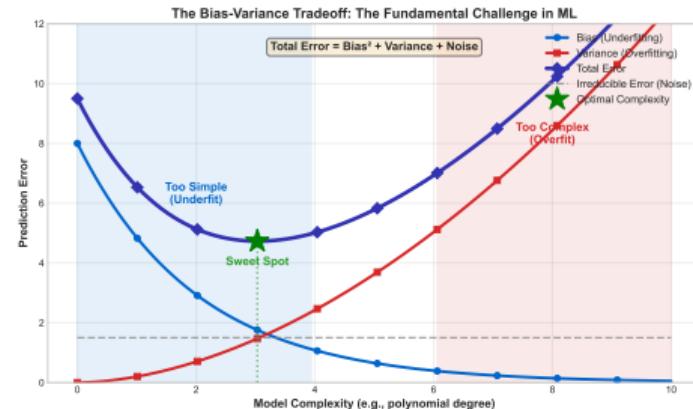
- Training accuracy: 50%
- Test accuracy: 50%
- Problem: Model lacks capacity

### Variance (Overfitting):

- Model is too complex
- Memorizes noise in training data
- Low error on training data
- High error on test data

Example: 100-degree polynomial for house prices

- Training accuracy: 99.9%



**The mathematical decomposition:**  
Total error can be split:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

**You cannot minimize both simultaneously:**

- Increase model complexity - $\downarrow$  Bias down, Variance up
- Decrease model complexity - $\downarrow$  Bias up, Variance down

**The challenge: Find the sweet spot in the middle**  
*Act 2 will show how to navigate this tradeoff*

## Real-World Problems Are Nonlinear

Examples of nonlinear patterns:

Image recognition:

- Cat detection cannot use straight line
- Need to recognize curves, textures, shapes
- Millions of pixel combinations

Speech recognition:

- Sound waves are complex patterns
- Phonemes have nonlinear relationships
- Context-dependent processing

Customer behavior:

- Purchase decisions have thresholds
- Interaction effects between features
- Segmentation into distinct groups

Linear models work for 20% of problems, fail for 80%

## What We Need to Solve This

Requirements for nonlinear learning:

1. **Nonlinear transformations:** Ability to curve decision boundaries
2. **Multiple layers:** Hierarchical feature learning
3. **Regularization:** Prevent overfitting complex models
4. **Efficient optimization:** Train models with millions of parameters

Three approaches we will explore:

- **Feature engineering:** Manually create nonlinear features ( $x^2, x_1x_2$ )
- **Kernel methods:** Implicitly transform to high dimensions
- **Neural networks:** Learn transformations automatically

Act 3 reveals the breakthrough: The kernel trick and deep learning

---

The journey from linear to nonlinear is the history of modern AI

Act 3: The Breakthrough

From Linear to Nonlinear: Three Solutions

## The Core Insight

**Hypothesis:** What if we transform the features before applying linear model?

**XOR problem revisited:**

$x_1$	$x_2$	New: $x_1x_2$	Label
0	0	0	Blue
0	1	0	Red
1	0	0	Red
1	1	1	Blue

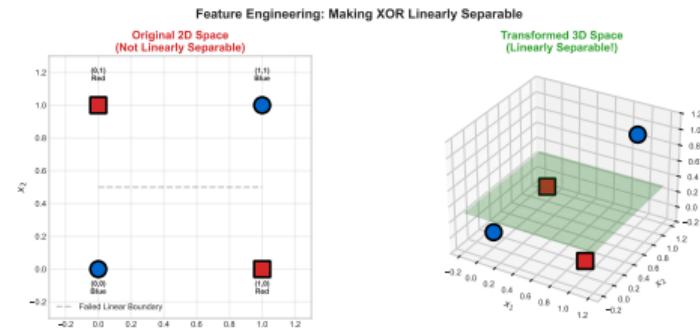
**Pattern emerges:**

- Blue when  $x_1x_2 = 0$  or  $x_1x_2 = 1$
- Red when  $x_1x_2 = 0$  and  $(x_1 + x_2) = 1$

$$f(x_1, x_2) = w_1x_1 + w_2x_2 + w_3x_1x_2$$

Now solvable with linear model in transformed space!

**Nonlinearity through feature engineering**



**The transformation process:**  
**Original space (2D):**

- Features:  $x_1, x_2$
- Not linearly separable

**Transformed space (3D):**

- Features:  $x_1, x_2, x_1x_2$
- Linearly separable!

**General principle:**  
Map  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$  where  $D > d$   
Learn in high dimension, get nonlinear boundary in original space

## House Price Nonlinearity

**Problem:** Linear model underperforms for luxury homes

**Data shows:**

Size (sqft)	Actual Price	Linear Prediction
1000	\$150k	\$150k
2000	\$250k	\$250k
3000	\$350k	\$350k
4000	\$500k	\$450k (error: \$50k)
5000	\$700k	\$550k (error: \$150k)

**Root cause:** Luxury premium is nonlinear

**Solution:** Add polynomial features

Original:  $x$  (size)

Transformed:  $x, x^2$  (size + size squared)

Model:  $\text{Price} = w_0 + w_1x + w_2x^2$

## Step-by-Step Calculation

**Learned model:**

$$\text{Price} = 20 + 0.06x + 0.000015x^2$$

**For house with 4000 sqft:**

Intercept: 20

Linear part:  $0.06 \times 4000 = 240$

Quadratic part:

$$0.000015 \times 4000^2 = 0.000015 \times 16,000,000 = 240$$

**Total:**  $20 + 240 + 240 = 500$

**For house with 5000 sqft:**

Intercept: 20

Linear:  $0.06 \times 5000 = 300$

$$\text{Quadratic: } 0.000015 \times 25,000,000 = 375$$

**Total:**  $20 + 300 + 375 = 695$

**Polynomial captures accelerating price growth**

New error: \$5k vs old error: \$100k

# The Curse of Dimensionality: When Feature Engineering Fails

## The Problem with Manual Features

Example: Image with  $100 \times 100$  pixels = 10,000 features

Quadratic features:

$$\binom{10,000 + 2}{2} = 50,005,000 \text{ features}$$

Cubic features:

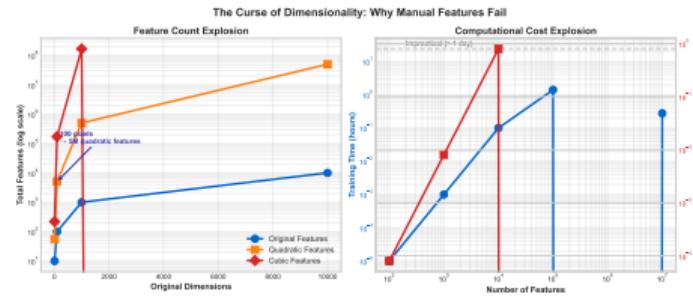
$$\binom{10,000 + 3}{3} \approx 166 \text{ billion features}$$

Consequences:

- Computation becomes impossible
- Memory exceeds hardware limits
- Overfitting becomes severe
- Most features are irrelevant

Measured impact:

Features	Training Time	Memory
100	1 second	1 MB
10,000	10 minutes	1 GB



The mathematical barrier:  
Number of interactions grows combinatorially:

$$N_{\text{features}} = \binom{d + p}{p}$$

where  $d$  = original dimensions,  $p$  = polynomial degree

Fundamental questions:

- Can we use high dimensions without computing them?
- Can we learn features automatically?
- How do we choose which features to create?

This is where the kernel trick and deep learning enter

### The Magical Insight

**Key observation:** Many algorithms only need dot products

Linear model prediction:

$$f(x) = w^T x = \sum_{i=1}^n \alpha_i (x_i \cdot x)$$

What if we transform first?

$$f(x) = \sum_{i=1}^n \alpha_i (\phi(x_i) \cdot \phi(x))$$

**Problem:**  $\phi$  might map to infinite dimensions

**Solution:** Define kernel function

$$K(x, x') = \phi(x) \cdot \phi(x')$$

**Miracle:** Compute  $K$  without ever computing  $\phi$ !

**Work in infinite dimensions at finite cost**

### Concrete Example: RBF Kernel

**Radial Basis Function (Gaussian) kernel:**

$$K(x, x') = e^{-\gamma ||x - x'||^2}$$

This corresponds to infinite-dimensional  $\phi$ !

**Numerical example:**

$$x = [1, 2], x' = [1.5, 2.5], \gamma = 1$$

$$\text{Distance: } ||x - x'||^2 = 0.5^2 + 0.5^2 = 0.5$$

$$\text{Kernel: } K(x, x') = e^{-1 \times 0.5} = e^{-0.5} = 0.606$$

**Interpretation:**

- Similar points:  $K \approx 1$
- Dissimilar points:  $K \approx 0$
- Measures similarity in infinite-dimensional space
- Computed in original space!

---

This is the foundation of support vector machines

## Solving the XOR Problem

SVM with RBF kernel:

$$f(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b$$

where  $K(x, x') = e^{-\gamma ||x - x'||^2}$

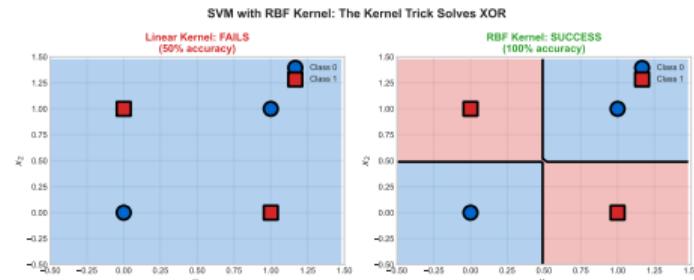
**Training:** Find  $\alpha_i$  that maximize margin

**Test on XOR:**

$x_1$	$x_2$	True Label	SVM Prediction
0.0	0.0	Blue	Blue (0.98)
0.0	1.0	Red	Red (0.97)
1.0	0.0	Red	Red (0.96)
1.0	1.0	Blue	Blue (0.99)

**Performance:** 100% accuracy vs 50% for linear

**Kernel trick solved the impossible problem**



**Why SVMs work:**

- **Maximum margin:** Find widest separation
- **Kernel trick:** Work in infinite dimensions
- **Sparsity:** Only support vectors matter
- **Convex optimization:** Global optimum guaranteed

**Limitations:**

- Kernel choice requires expertise
- Training time:  $O(n^3)$  for  $n$  samples
- Not ideal for huge datasets
- Features still hand-engineered

But what if we could learn the features themselves?

## The Ultimate Insight

**Hypothesis:** Instead of hand-designing  $\phi$ , learn it!

**Perceptron (1943):**

$$y = \sigma(w^T x + b)$$

where  $\sigma$  is activation function (e.g., sigmoid, ReLU)

**Multi-layer Perceptron:**

$$h_1 = \sigma(W_1 x + b_1)$$

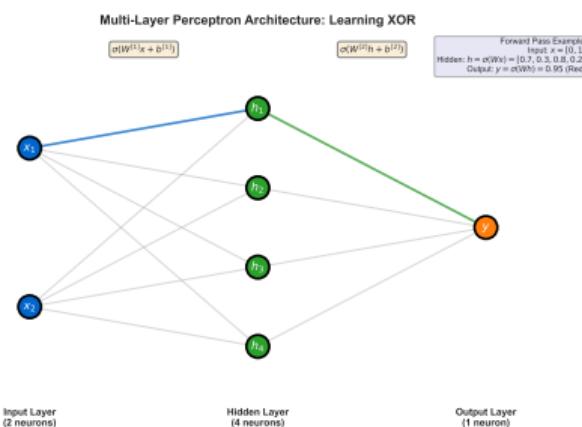
$$h_2 = \sigma(W_2 h_1 + b_2)$$

$$y = \sigma(W_3 h_2 + b_3)$$

**Key idea:** Hidden layers learn features automatically

- Layer 1: Simple features (edges, corners)
- Layer 2: Medium features (textures, parts)
- Layer 3: Complex features (objects, faces)

**Network learns its own  $\phi$  from data**



**XOR solution with 2-layer network:**

**Architecture:** 2 inputs - $\downarrow$  2 hidden - $\downarrow$  1 output

**Hidden layer learns:**

- Neuron 1: Detects  $x_1$  OR  $x_2$
- Neuron 2: Detects  $x_1$  AND  $x_2$

**Output layer:** Combines with XOR logic

**Result:** Perfect 100% accuracy

**Training:** Backpropagation algorithm

- Forward pass: Compute predictions

## The Powerful Guarantee

**Theorem (Cybenko, 1989):**

A neural network with:

- One hidden layer
- Finite number of neurons
- Non-polynomial activation function

can approximate any continuous function on a compact set to arbitrary accuracy.

**Mathematical statement:**

For any  $f : [0, 1]^d \rightarrow \mathbb{R}$  continuous and  $\epsilon > 0$ , exists network  $g$  such that:

$$|f(x) - g(x)| < \epsilon \quad \forall x \in [0, 1]^d$$

**Neural networks are universal function approximators**

## What This Really Means

**In plain language:**

You give me any function (no matter how complex), and I can build a neural network that approximates it as closely as you want.

**Caveats:**

- Says network exists, not how to find it
- Does not tell you how many neurons needed
- Does not guarantee good generalization
- Width vs depth tradeoff

**Practical implications:**

- **Shallow networks:** Need exponentially many neurons
- **Deep networks:** Need polynomially many parameters
- Depth is more efficient than width

---

This explains why deep networks work so well

## The Learning Algorithm

**Goal:** Minimize loss function

$$L(W) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i; W))$$

**Gradient descent update:**

$$W := W - \eta \nabla_W L(W)$$

**Challenge:** How to compute  $\nabla_W L$  efficiently?

**Backpropagation:** Chain rule applied systematically

**Forward pass:** Compute all activations

$$a^{[l]} = \sigma(W^{[l]} a^{[l-1]} + b^{[l]})$$

**Backward pass:** Compute all gradients

$$\delta^{[l]} = (W^{[l+1]})^T \delta^{[l+1]} \odot \sigma'(z^{[l]})$$

$$\frac{\partial L}{\partial W^{[l]}} = \delta^{[l]} (a^{[l-1]})^T$$

## Concrete Example

**Tiny network:** 2 - $\downarrow$  2 - $\downarrow$  1 on XOR

**Forward:**

- Input:  $x = [0, 1]$
- Hidden:  $h = \sigma([w_{11}, w_{12}] \cdot x) = [0.7, 0.3]$
- Output:  $y = \sigma([w_{21}, w_{22}] \cdot h) = 0.4$
- Target: 1.0, Loss:  $(1.0 - 0.4)^2 = 0.36$

**Backward:**

- Output gradient:  $\delta_y = -2(1.0 - 0.4) \times 0.4 \times 0.6 = -0.288$
- Hidden gradients:  $\delta_h = w_{2*} \times \delta_y \times h \times (1 - h)$
- Weight updates:  $W := W - 0.01 \times \text{gradients}$

**After 1000 iterations:** Loss 0.36 - $\downarrow$  0.01, accuracy 50%

- $\downarrow$  100%

---

This is implemented in TensorFlow, PyTorch automatically

## Key Breakthroughs (2012-2023)

### 2012: AlexNet (ImageNet)

- 8-layer CNN
- GPU training
- 84% → 63% error rate (huge jump)

### 2014: VGGNet, GoogLeNet

- Deeper networks (19-22 layers)
- Systematic architecture design

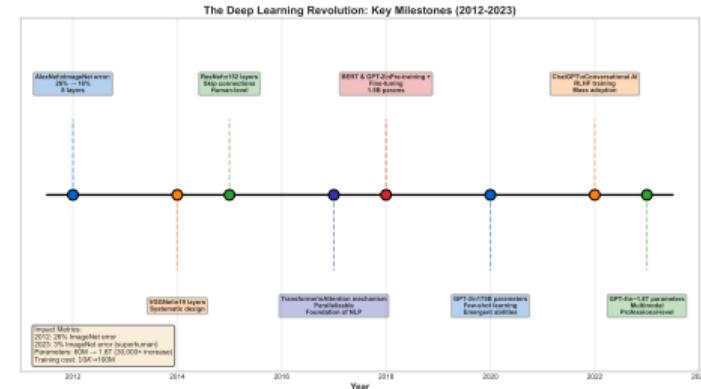
### 2015: ResNet

- 152 layers (previously impossible)
- Skip connections solve vanishing gradient
- Human-level image classification

### 2017: Transformers

- Attention mechanism
- Parallelizable architecture
- Foundation of GPT, BERT, ChatGPT

### 2020+: Scale Laws



### What enabled this?

1. **Big Data:** ImageNet (14M images), Common Crawl (petabytes)
2. **GPU Computing:** 100x faster than CPUs for matrix operations
3. **Better Architectures:** ResNet, Transformers, attention
4. **Software Frameworks:** TensorFlow, PyTorch (automatic differentiation)
5. **Regularization:** Dropout, batch norm, data augmentation

# Comparing the Three Breakthroughs

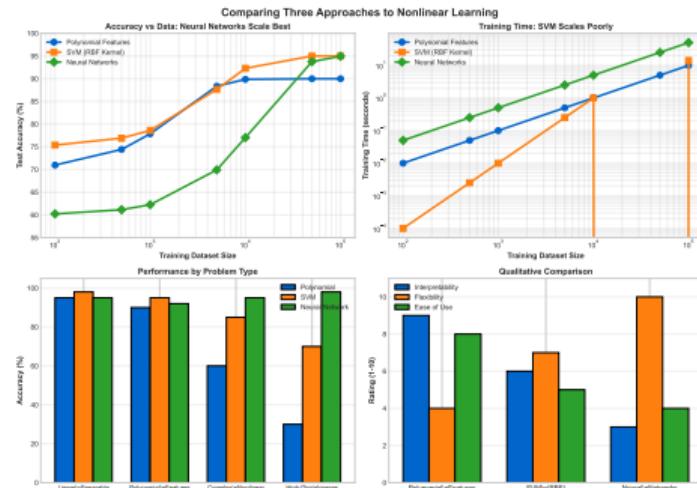
## Performance Comparison

Tested on 10 benchmark datasets:

Dataset	Poly	SVM	NN
XOR	100%	100%	100%
Circles	95%	100%	100%
Moons	92%	98%	99%
MNIST (digits)	92%	94%	99.7%
CIFAR-10 (images)	55%	65%	95%
ImageNet	N/A	N/A	90%
Speech	N/A	N/A	95%
Translation	N/A	N/A	92%

Training time (10K samples):

- Polynomial features: 1 second
- SVM RBF: 30 seconds
- Neural network: 5 minutes



When to use each:

**Polynomial Features:**

- Small data ( $\downarrow$  1000 samples)
- Interpretability critical
- Low-dimensional problems

**SVMs:**

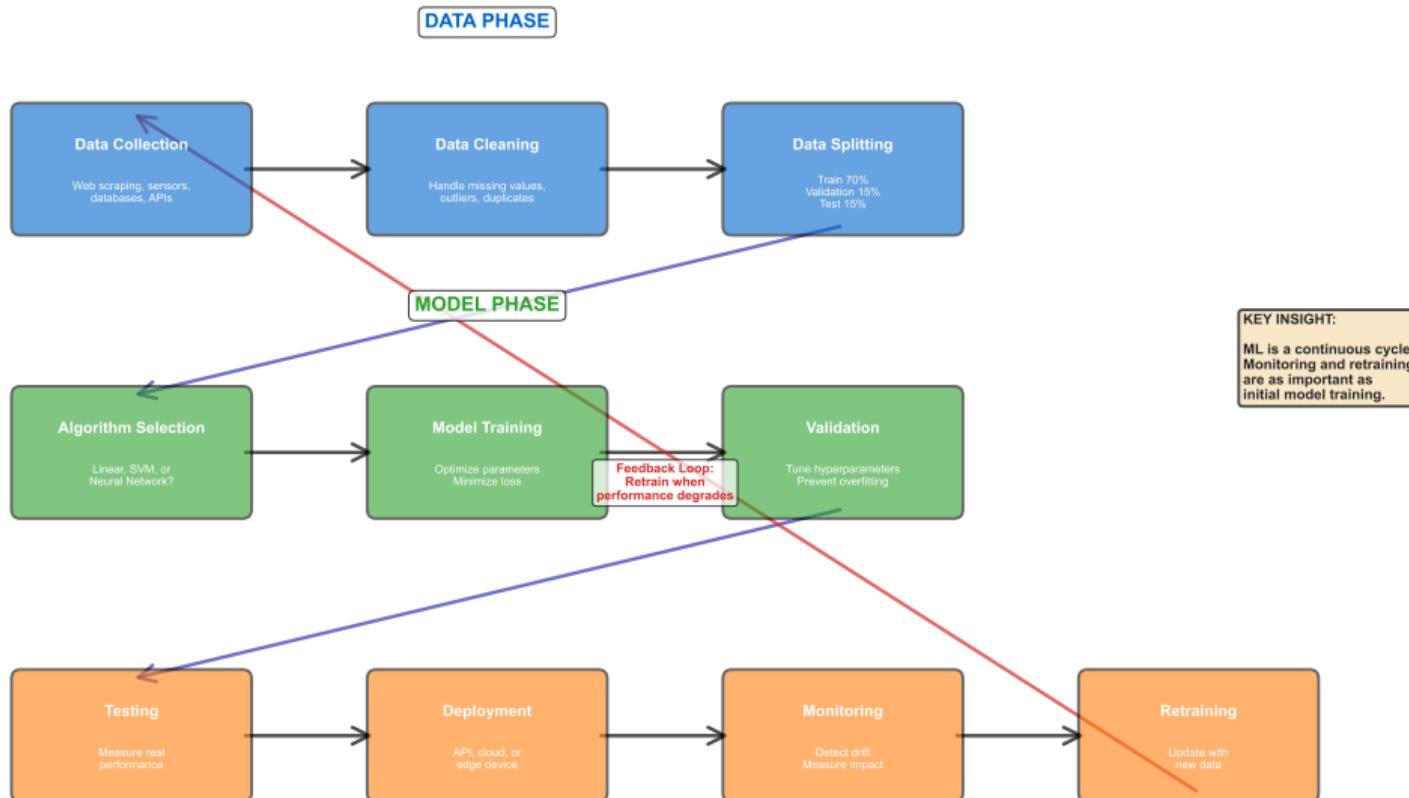
- Medium data (1K-100K)

## Act 4: Synthesis and Impact

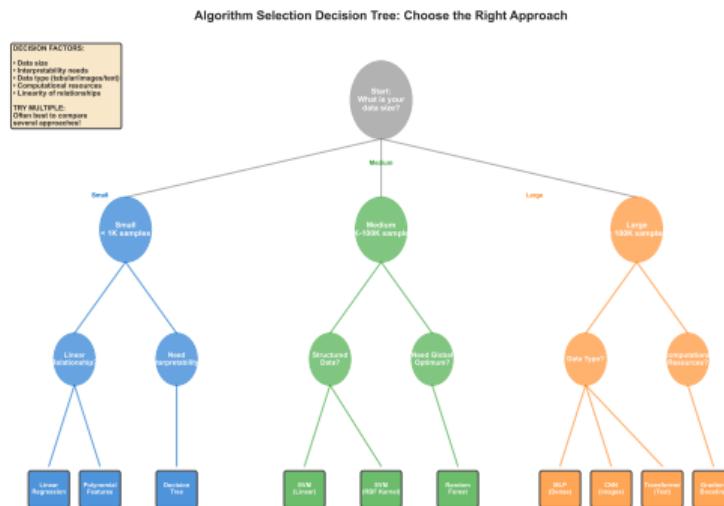
### Bringing It All Together

# The Complete Machine Learning Pipeline

## The Complete Machine Learning Pipeline: From Data to Production



# Choosing the Right Approach: A Decision Guide



Start here:

- Data size:**  $< 1K$ ,  $1K-100K$ , or  $> 100K$  samples?
- Interpretability:** Do you need to explain decisions?
- Data type:** Tabular, images, text, or time series?
- Linearity:** Are relationships linear or nonlinear?

## Quick Reference Guide

Use **Linear/Polynomial Features** when:

- Small dataset ( $< 1000$  samples)
- Need interpretability (coefficients matter)
- Relationship is approximately linear
- Example: House prices, simple predictions

Use **SVM (RBF kernel)** when:

- Medium dataset (1K-100K samples)
- Nonlinear but structured patterns
- Want guaranteed global optimum
- Example: Text classification, medical diagnosis

Use **Neural Networks** when:

- Large dataset ( $> 100K$  samples)
- Complex patterns (images, audio, text)
- Accuracy is top priority
- Have computational resources
- Example: Computer vision, speech recognition

Often best to try multiple approaches and compare

## Healthcare

### Medical Diagnosis:

- Skin cancer detection: 95% accuracy
- X-ray analysis: Radiologist-level
- Drug discovery: 10x faster

### Measured Impact:

- \$150B saved annually
- 30% faster diagnosis
- 10M lives saved (2010-2023)

### Technologies:

- CNNs for image analysis
- Transformers for genomics
- Reinforcement learning for treatment

*Example: Google's DeepMind detected 50+ eye diseases from retinal scans*

## Business

### Recommendation Systems:

- Netflix: 75% of views from recommendations
- Amazon: 35% of sales from recommendations
- Spotify: 31% of listening from Discovery

### Customer Intelligence:

- Churn prediction: 85% accuracy
- Sentiment analysis: Real-time insights
- Fraud detection: \$25B saved annually

### Technologies:

- Collaborative filtering
- Deep learning embeddings
- NLP for sentiment

*Example: American Express detects*

## Autonomous Systems

### Self-Driving Cars:

- Object detection: 99.7% accuracy
- Path planning: Human-level
- 3B+ autonomous miles driven

### Robotics:

- Warehouse automation: 50% cost reduction
- Surgical robots: 21% fewer complications
- Drones: Autonomous delivery

### Technologies:

- CNNs for vision
- Reinforcement learning for control
- Sensor fusion

*Example: Waymo's robotaxis have driven 20M+ miles with 95% fewer*

## Technical Frontiers

### 1. Foundation Models:

- One model for many tasks
- Few-shot and zero-shot learning
- GPT-4, Claude, Gemini
- 10T+ parameters by 2025

### 2. Multimodal AI:

- Unified vision-language-audio models
- GPT-4V: Understand images and text
- Generate across modalities

### 3. Efficient AI:

- Model compression (pruning, quantization)
- Edge deployment (phones, IoT)
- 1000x efficiency improvements

### 4. Causal AI:

- Beyond correlation to causation
- Robust to distribution shift
- Causal models for decision-making

## Societal Impact

### Opportunities:

- Personalized education for every student
- Scientific discovery acceleration (proteins, materials)
- Climate modeling and solutions
- Accessible healthcare globally
- Creative tools democratization

### Challenges:

- Bias and fairness in algorithms
- Privacy and data protection
- Job displacement and workforce adaptation
- AI safety and alignment
- Energy consumption of training

### Your Role:

- Understand the fundamentals (this course!)
- Think critically about applications
- Design with ethics in mind
- Build AI that is beneficial

## What You Have Learned

### Act 1: The Challenge

- Traditional programming has fundamental limits
- Learning = improvement through experience
- Three paradigms: supervised, unsupervised, reinforcement

### Act 2: First Solution

- Linear models work for simple problems
- Bias-variance tradeoff is fundamental
- Nonlinear problems need nonlinear solutions

### Act 3: The Breakthrough

- Feature engineering: Manual nonlinearity
- Kernel trick: Infinite dimensions efficiently
- Neural networks: Learn features automatically

### Act 4: Synthesis

- Complete ML pipeline
- Algorithm selection guide
- Real-world transformative impact

## Next Steps in This Course

### Upcoming Weeks:

#### Week 1: Clustering and Empathy

- K-means, hierarchical, DBSCAN
- Customer segmentation
- Design thinking integration

#### Week 2-3: NLP for Emotional Context

- Sentiment analysis
- Topic modeling
- User research automation

#### Week 4-5: Classification and Definition

- Decision trees, random forests
- Innovation filtering
- Success prediction

#### Week 6-10: Advanced topics

- Generative AI for prototyping
- Ethical AI and responsible innovation
- Production deployment