IN402

Machine Learning

Chapter
3

Overfitting and the Bias-Variance Tradeoff

Author: Abbas El-Hajj Youssef

University: Lebanese University

Department: Department of Computer Science

These notes extend course materials taught by Prof. Ahmad Faour with additional content from textbooks and supplementary resources.

Disclaimer: *This is not an official course document.*

© 2025 Abbas El-Hajj Youssef. All rights reserved.

Contents

1 Introduction: The Paradox of Learning

Picture this: you've built a machine learning model that achieves 99.8% accuracy on your training data. Your colleagues are impressed. Your supervisor nods approvingly. Yet when deployed, this same model performs worse than a simple baseline. What went wrong?

This scenario illustrates one of machine learning's most fundamental challenges—a challenge so pervasive that every practitioner, from novices to experts, must constantly guard against it. We're talking about the delicate balance between learning and memorization, between finding the signal and fitting the noise. This tension manifests as two primary failure modes: **underfitting** and **overfitting**.

An **underfit** model is too simple to capture the underlying trend. It performs poorly on both training and new data. In contrast, an **overfit** model is too complex. It becomes so attuned to the training data that it starts to memorize its noise and random fluctuations. While it performs exceptionally well on the training data, its performance on new data is disastrous.

El-Hajj Youssef

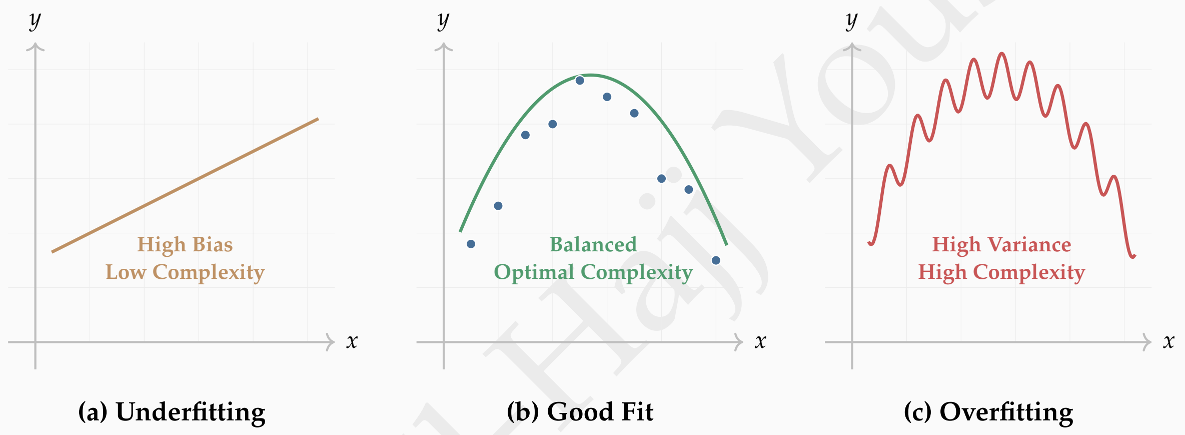

Figure 1: *Model complexity spectrum: underfitting (too simple), good fit (balanced), and overfitting (too complex).*

Analogy: A Student Preparing for an Exam

- An **underfitting** student hasn't studied enough. They fail both the practice questions and the final exam.
- A **well-fitting** student understands the core concepts. They perform well on both practice questions and the final exam.
- An **overfitting** student has memorized the exact answers to every practice question but fails to grasp the underlying concepts. They ace the practice test but fail the final exam when faced with new questions.

Our goal is to train models that are like the well-fitting student: they understand the general patterns and can apply them to new problems.

The Central Paradox of Learning

The very power that allows machine learning models to discover complex patterns—their flexibility—is also their greatest weakness. A model that is too simple fails to capture reality; one that is too complex learns a fantasy.

This chapter will equip you with:

1. **Intuitive understanding** of why this paradox exists, through practical examples.
2. **Mathematical tools** to formally define and analyze prediction error.
3. **Practical strategies** to diagnose and prevent overfitting in your own models.
4. **Diagnostic techniques** to detect when overfitting occurs and how to address it.

Notation Convention

To ensure clarity, we establish the following notation for this chapter:

Symbol	Meaning
$\mathbf{x}$ or x	Input feature(s)
y	True label or target value
$\hat{y}$	Model's predicted value
$h_{\boldsymbol{\theta}}(\mathbf{x})$	Hypothesis function with parameters $\boldsymbol{\theta}$
$\boldsymbol{\theta}$	Vector of model parameters, $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)$
m	Number of training examples
n	Number of features
$J(\boldsymbol{\theta})$	Cost/loss function
λ	Regularization strength hyperparameter
α	Learning rate

2 Building Intuition: A Tale of Two Models

Before diving into formalism, let's develop intuition through a story. This narrative will illuminate why the concepts of bias and variance are not just theoretical curiosities, but practical necessities for every data scientist.

2.1 The Challenge: Learning from Limited Observations

Imagine you're a data scientist at an agricultural institute. Your task: predict crop yield based on fertilizer amount. You've collected careful measurements, but like all real-world data, your observations contain noise—measurement errors, weather variations, soil differences—factors you haven't explicitly modeled.

The Universal Pattern

This scenario represents a universal challenge in machine learning:

- **Signal:** The true, underlying relationship between inputs and outputs.

Abbas El-Hajj Youss

- **Noise:** Random variations and unobserved factors that obscure the signal.
- **Our Goal:** Learn the signal while ignoring the noise.

Whether predicting stock prices or diagnosing diseases, this fundamental challenge remains: how do we extract meaningful patterns from noisy, limited data?

Figure 2 reveals our dataset. We can see an underlying trend—yield increases with fertilizer, then plateaus—but the points don't fall perfectly on a smooth curve.

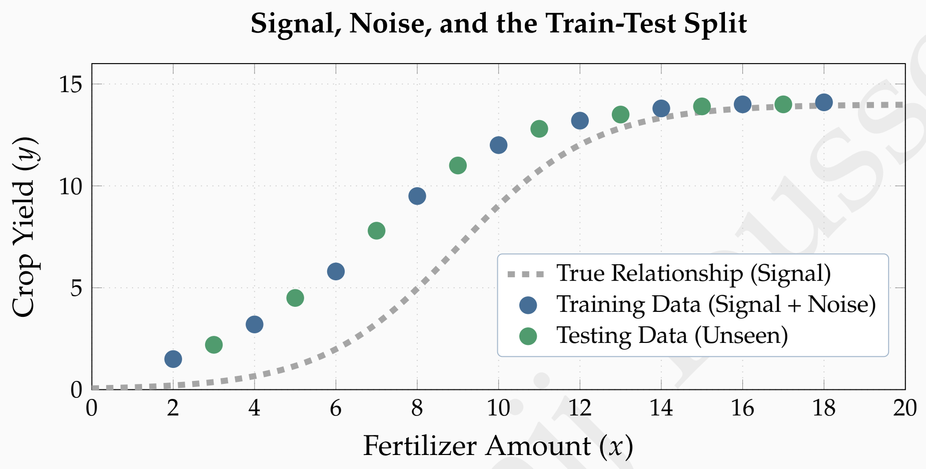

Figure 2: *Our experimental data reveals a non-linear signal obscured by noise. The strategic split into training (blue) and testing (green) sets is crucial for evaluating a model's ability to generalize.*

The Crucial Split: Why We Separate Training and Testing

Splitting data isn't just a procedural step—it's a philosophical commitment to honest evaluation.

- **Training Set:** Where the model learns, makes mistakes, and refines its understanding.
- **Testing Set:** The ultimate judge—unseen data that reveals whether the model truly *learned* or merely *memorized*.

The Acid Test: A model that excels on training data but fails on testing data is like the overfitting student—impressive in familiar territory, but useless when faced with novel challenges.

2.2 Approach One: The Simplicity of Linear Thinking

Our first instinct might be to fit a straight line. Linear regression embodies the principle of Occam's Razor: start simple.

Hajj Youssef

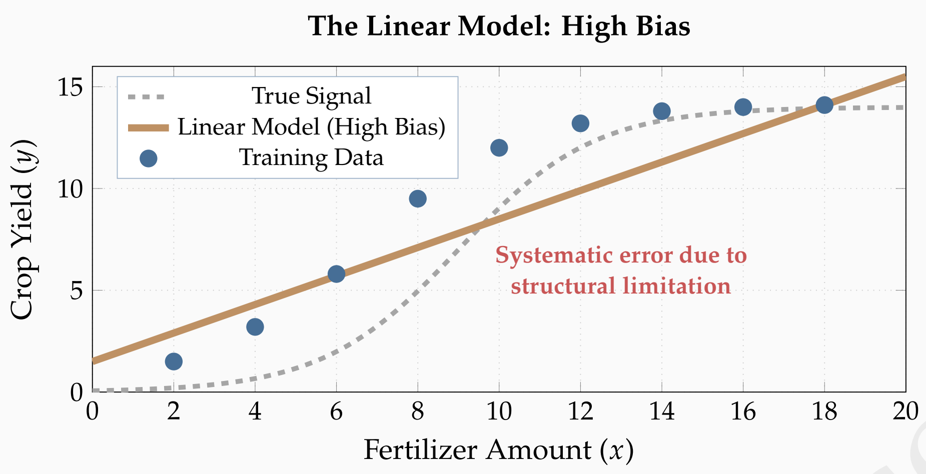

Figure 3: *The linear model's fundamental limitation: no amount of parameter tuning can make a straight line follow an S-curve. This exemplifies* ***high bias****.*

The Price of Simplicity

Observe the systematic nature of the linear model's errors. It consistently underestimates yield in the middle range and overestimates it at the extremes. This isn't a failure of optimization—it's a fundamental architectural limitation. The model is structurally incapable of representing the true relationship.

2.3 Approach Two: The Seduction of Complexity

Frustrated, we swing to the opposite extreme: a highly flexible polynomial model, so adaptable it can contort itself to match any pattern.

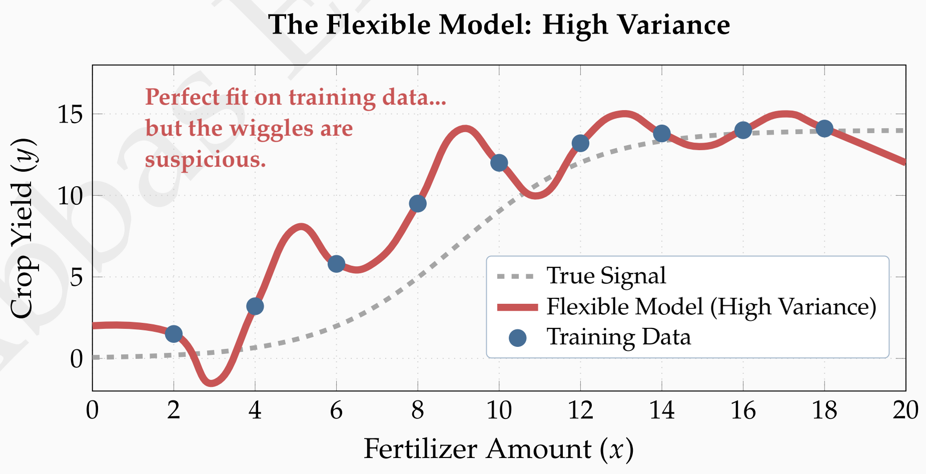

Figure 4: *The flexible model perfectly threads through the training points. This remarkable adaptability—its ability to have* ***low bias*** *on the training set—comes at the cost of learning the noise, a sign of* ***high variance****.*

2.4 The Moment of Truth: Generalization Performance

How do our models perform on data they've never seen? This is where illusions shatter and true understanding is measured.

Abbas El-Hajj Youssef

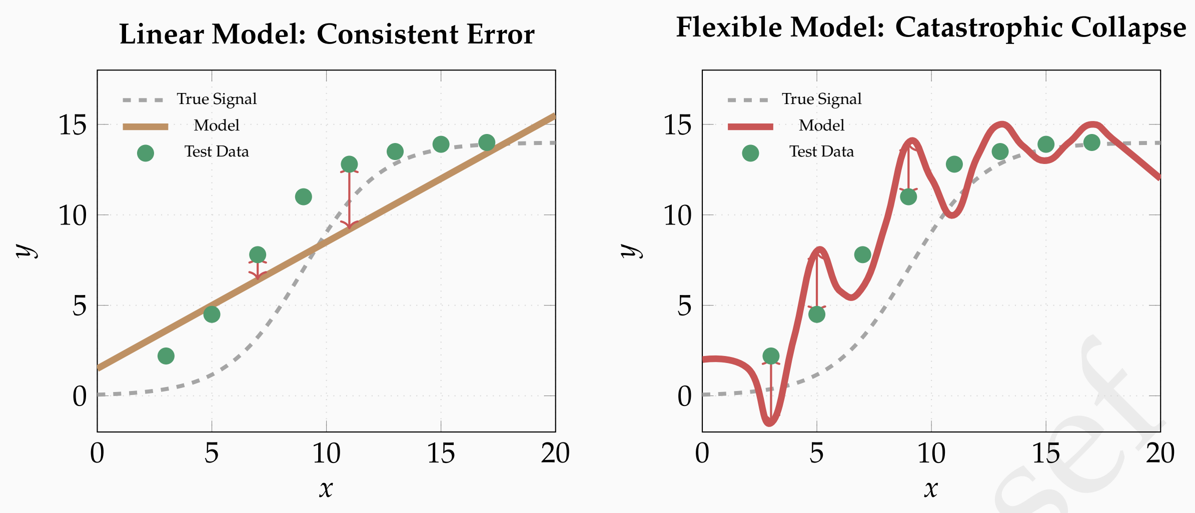

Figure 5: *The test set reveals the truth. The linear model remains mediocre. The flexible model, which seemed perfect, fails catastrophically on new data. Its predictions are wild and unreliable.*

The Surprising Verdict

The Mean Squared Error (MSE) on each dataset tells the full story.
Training Performance:

- Linear Model: MSE = 156.4 (mediocre)
- Flexible Model: MSE ≈ 0.5 (near perfect)

Test Performance—The Plot Twist:

- Linear Model: MSE = 168.2 (consistent with training)
- Flexible Model: MSE = **312.7** (catastrophic degradation)

The Revelation: The "worse" model on the training set proved superior for generalization. The flexible model's apparent mastery was an illusion—it memorized noise rather than learning the signal. This is **overfitting**.

3 The Mathematical Framework: Bias-Variance Decomposition

Having built intuition through our agricultural example, we are now prepared to formalize the concepts of overfitting and underfitting. This section will bridge the gap between intuition and theory by constructing the mathematical framework from the ground up, starting with a concrete thought experiment.

3.1 The Challenge: Sensitivity to the Dataset

Our analysis so far has been based on a single training set. However, a fundamental question remains: if we were to repeat our data collection process, would we obtain the exact same model? The answer is almost certainly no. Each collected dataset is a unique sample from the true underlying process and contains its own specific random noise. A model's predictions, therefore, depend on the particular dataset it was trained on.

The Thought Experiment of Resampling

To understand a model's stability, we conduct a thought experiment:

1. Imagine we could collect many different training sets, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_N$, all drawn from the same underlying reality.
2. We train an identical type of model (e.g., linear regression) on each of these datasets, producing a collection of distinct models: $h_1, h_2, \ldots, h_N$.

The central question of the bias-variance framework is: **How do the predictions of these models behave, both individually and on average?** This variation between models trained on different datasets is precisely what we aim to quantify.

3.2 A Concrete Example: Multiple Models in Action

Let's make this abstract idea tangible with a numerical example.

Three Datasets, Three Linear Models

Suppose the true, noise-free relationship is $f(x) = 2x + 3$. We collect three different small, noisy datasets and fit a linear model to each.

Dataset	**Resulting Model Hypothesis**
Dataset 1 ($\mathcal{D}_1$)	$\hat{y}_1 = h_1(x) = 1.95x + 3.15$
Dataset 2 ($\mathcal{D}_2$)	$\hat{y}_2 = h_2(x) = 1.95x + 3.00$
Dataset 3 ($\mathcal{D}_3$)	$\hat{y}_3 = h_3(x) = 1.90x + 3.55$

Now, let's evaluate each model's prediction at a new point, $x = 4$. The true value is $f(4) = 2(4) + 3 = 11$.

- Model 1 predicts: $\hat{y}_1 = 1.95(4) + 3.15 = 10.95$
- Model 2 predicts: $\hat{y}_2 = 1.95(4) + 3.00 = 10.80$
- Model 3 predicts: $\hat{y}_3 = 1.90(4) + 3.55 = 11.15$

Key Observations:

1. The predictions are all close to the true value of 11, but none are perfect.
2. The predictions vary from each other, spanning a range from 10.80 to 11.15.
3. The **average prediction** is $\frac{10.95+10.80+11.15}{3} = 10.97$.

This simple example reveals two distinct sources of error, which we can now define as bias and variance.

3.3 Formal Definitions with Intuition

Let's translate the observations from our example into formal mathematical definitions. The key piece of notation is $\mathbb{E}_{\mathcal{D}}[\hat{y}]$, which represents the **average prediction** we would get if we could average over all possible training datasets $\mathcal{D}$. In our example, we approximated this with

ousse

10.97.

Bias: The Systematic Error

Bias measures the difference between the model's average prediction and the true value. It quantifies the error from a model's simplifying assumptions.

$$\text{Bias}(\hat{y}) = \mathbb{E}_{\mathcal{D}}[\hat{y}] - f(x) \tag{1}$$

From our example at $x = 4$:

$$\text{Bias} \approx 10.97 - 11 = -0.03$$

A non-zero bias implies that, on average, the model is systematically incorrect. High bias is the hallmark of **underfitting**.

Variance: The Instability Error

Variance measures the expected squared deviation of any single model's prediction from the average model's prediction. It quantifies the model's instability.

$$\text{Var}(\hat{y}) = \mathbb{E}_{\mathcal{D}}\left[\left(\hat{y} - \mathbb{E}_{\mathcal{D}}[\hat{y}]\right)^2\right] \tag{2}$$

From our example at $x = 4$:

$$\begin{aligned}\text{Var} &\approx \frac{(10.95 - 10.97)^2 + (10.80 - 10.97)^2 + (11.15 - 10.97)^2}{3} \\ &\approx 0.0206\end{aligned}$$

High variance means the model is overly sensitive to the training data, changing significantly with new data. This is the hallmark of **overfitting**.

3.4 The Decomposition Theorem

The theoretical cornerstone of this topic is the realization that these error components, along with the inherent noise in the data, sum up to the total expected error.

The Bias-Variance Decomposition

For a regression problem with squared error loss, the expected error of a model at a point **x** can be decomposed into three fundamental components:

$$\boxed{\underbrace{\mathbb{E}_{\mathcal{D}}\left[(y - \hat{y})^2\right]}_{\text{Expected Total Error}} = \underbrace{\text{Bias}(\hat{y})^2}_{\substack{\text{Error from}\\\text{Wrong Assumptions}}} + \underbrace{\text{Var}(\hat{y})}_{\substack{\text{Error from}\\\text{Model Instability}}} + \underbrace{\sigma^2}_{\substack{\text{Irreducible}\\\text{Data Noise}}}} \tag{3}$$

Here, $y = f(x) + \epsilon$ is the noisy observation, and σ^2 is the variance of the noise term ϵ.

Implications of the Decomposition

This theorem provides profound insight into the nature of prediction error:

1. **A Fundamental Tradeoff:** To reduce bias, we often need more complex models (e.g., higher-degree polynomials). However, more complex models are more sensitive to data, which increases their variance. It is impossible to minimize both simultaneously.
2. **An Unbeatable Lower Bound:** The σ^2 term represents the inherent randomness in the data itself. No model, regardless of its quality, can eliminate this error. It establishes the best possible performance for any predictive model on that data.
3. **The Goal of Model Selection:** Our objective is not to find a model with zero bias or zero variance, but one that finds the "sweet spot" of complexity that minimizes the *sum* of these error terms.

3.5 The Tradeoff Visualized

This interplay between bias and variance as a function of model complexity is famously visualized in Figure 6. Simple models are on the left (high bias, low variance), while complex models are on the right (low bias, high variance). Our goal is to find the model complexity that corresponds to the minimum of the "Total Error" curve.

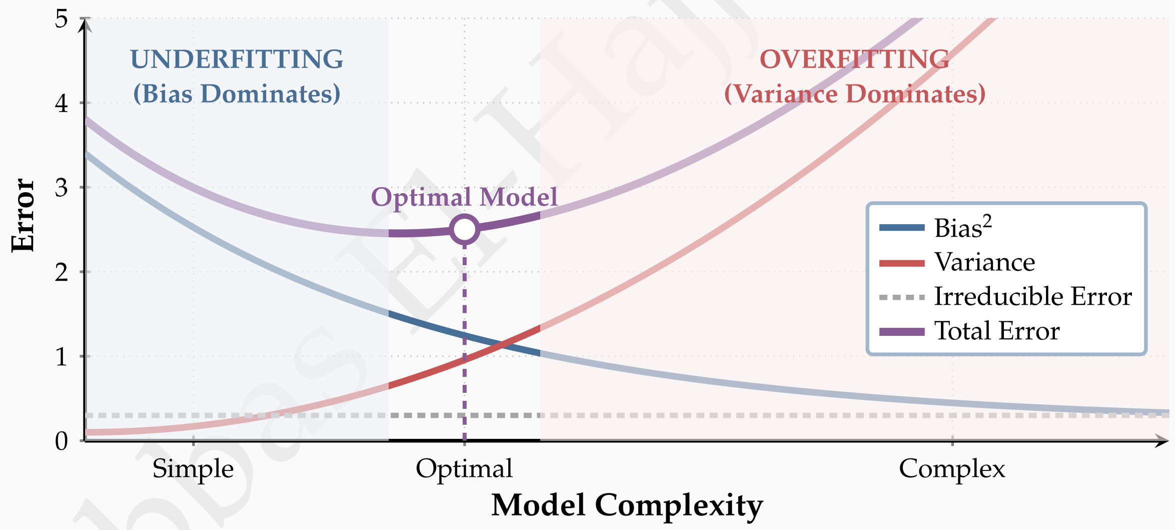

Figure 6: *The bias-variance tradeoff. As model complexity increases, squared bias decreases while variance increases. The art of machine learning lies in finding the optimal complexity that minimizes the total error.*

4 Diagnosing Overfitting in Practice

Now that we understand the theory behind overfitting, we must learn how to detect it in practice. Unlike the previous section where we imagined having many training sets, in reality we have only one dataset. A model that claims high performance is only useful if that performance holds up on new data. This section covers the essential diagnostic techniques using a single dataset.

Abbes El Hajj Youss

4.1 The Performance Gap Metric

The most reliable indicator of overfitting is a significant gap between the model's performance on the data it was trained on versus its performance on a separate, held-out dataset.

Generalization Gap

The generalization gap is the difference between a model's performance on the training data and its performance on unseen test data.
For classification tasks using accuracy:

$$\text{Gap} = \text{Accuracy}_{\text{train}} - \text{Accuracy}_{\text{test}} \tag{4}$$

For regression tasks using Mean Squared Error (MSE), note the reversed order:

$$\text{Gap} = \text{MSE}_{\text{test}} - \text{MSE}_{\text{train}} \tag{5}$$

Why reversed for MSE? Because MSE is a loss (lower is better), while accuracy is a score (higher is better). In both cases, a positive gap indicates potential overfitting.
A **small gap** indicates that the model generalizes well. A **large gap** is a strong symptom of overfitting.

4.2 Practical Guidelines for Acceptable Gaps

While there is no universal threshold for an acceptable performance gap, the following table provides practical guidelines based on typical model complexity. These are rules of thumb developed from experience.

Model Type	Typical Acceptable Gap	Action if Exceeded
Simple (Linear, Logistic)	$< 2\%$	Check for data quality issues
Moderate (Trees, Small NNs)	$< 5\%$	Consider regularization
Complex (Deep NNs)	$< 10\%$	Regularization likely needed
Very Complex (Transformers)	$< 15\%$	Multiple techniques needed

bas El-Hajj

Warning Signs of Overfitting

Red flags to watch for:

- Training accuracy $> 95\%$ while test accuracy $< 80\%$
- Training loss approaching zero while validation loss increases
- Model performance degrades with more training epochs
- Perfect or near-perfect training scores with complex models

4.3 Train-Validation-Test Splitting

To measure the performance gap reliably, we partition our data into distinct sets, each with a specific purpose:

- **Training Set (typically 60-70%):** The data used to train the model—to learn the parameters $\boldsymbol{\theta}$. The model sees and learns directly from this data.
- **Validation Set (typically 15-20%):** Used to evaluate the model during development, tune hyperparameters (like learning rate α, regularization strength λ, or model architecture choices), and make decisions about when to stop training. This set guides the training process but is not used to update parameters.
- **Test Set (typically 15-20%):** Held out completely until the very end of the project. It provides an unbiased estimate of the model's final performance on new, unseen data. This set should *never* influence any training decisions.

Why three sets?

Using only training and test sets can lead to *overfitting to the test set* if we repeatedly evaluate on it and adjust our model based on test performance. The validation set acts as a "practice test" that we can use multiple times during development, keeping the true test set pristine for final evaluation.

4.4 Learning Curves: A Visual Diagnostic Tool

Learning curves plot a model's performance on both the training and validation sets as training progresses (typically measured in epochs or iterations). They are among the most powerful diagnostic tools available because they reveal *when* and *how* overfitting develops during training.

Figure 7 shows two scenarios: one exhibiting overfitting and one showing good generalization.

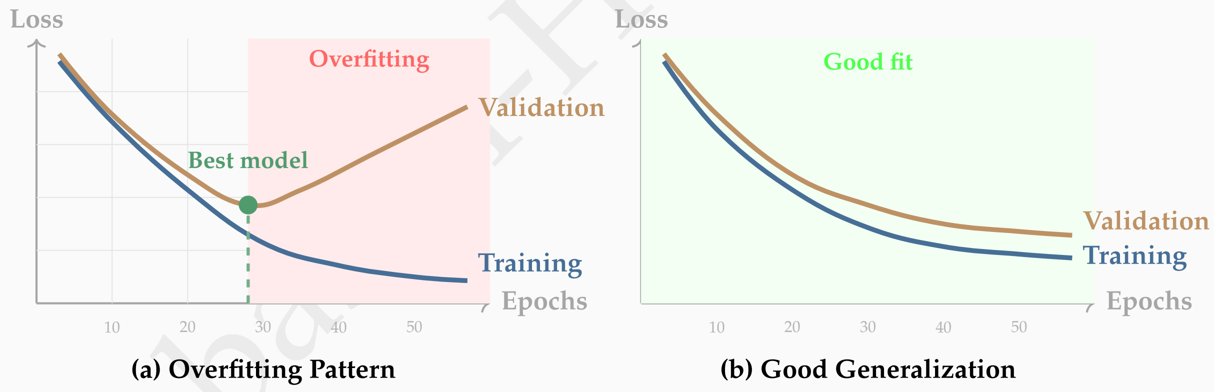

Figure 7: ***Learning curves diagnose overfitting.*** *(a) Diverging curves signal overfitting—validation loss increases while training loss decreases. The optimal model occurs at minimum validation loss (marked point). (b) Parallel curves indicate healthy learning—both losses decrease together.*

Interpreting Learning Curves:

- **Both curves decreasing together:** Good sign—model is learning generalizable patterns
- **Validation loss starts increasing while training loss decreases:** Clear sign of overfitting—model is memorizing training data
- **Large gap between curves:** Model has high variance (overfitting)
- **Both curves high and plateaued:** Model has high bias (underfitting)

Abbas Al-Hajj Youssef

4.5 Early Stopping

Learning curves enable a powerful regularization technique called **Early Stopping**. By monitoring the validation loss, we can halt training at the point where validation performance is best, even if the training loss could still decrease further. This prevents the model from continuing to overfit.

Why does validation loss increase?
After a certain point, the model starts fitting noise and training-set-specific patterns that don't generalize. This reduces training error but increases validation error.

Early Stopping Algorithm

Procedure:

1. Train the model while tracking both training and validation loss after each epoch
2. Keep track of the best validation performance seen so far and save that model state
3. If validation performance doesn't improve for k consecutive epochs (the "patience" parameter), stop training
4. Restore the model to the state with the best validation performance

Typical patience values: 5-20 epochs, depending on how noisy the validation loss is. More noise requires more patience to avoid stopping due to random fluctuations.

4.6 Diagnostic Checklist

Here is a practical workflow for diagnosing overfitting:

1. Split your data into training, validation, and test sets *before* any modeling
2. Train your model on the training set
3. Evaluate on both training and validation sets to compute the performance gap
4. Plot learning curves to visualize training dynamics
5. Watch for these warning signs:
   - Diverging learning curves (validation loss increasing)
   - Large performance gap (>10% for complex models, >5% for moderate models)
   - Near-perfect training scores (>98% accuracy or near-zero loss)
6. Use the validation set to guide all decisions about model complexity, hyperparameters, and when to stop
7. Reserve the test set for the final evaluation only—use it once at the very end

5 Combating Overfitting: Data-Centric Strategies

The quality and quantity of your data form the first line of defense against overfitting. Before applying complex regularization techniques, data-centric approaches often provide the most effective and interpretable solutions.

Abbas El-Hajj

5.1 Increasing Training Data

More data remains the most straightforward cure for overfitting. Additional examples force the model to learn generalizable patterns rather than memorizing specific instances.

Why it works:

- Reduces variance by exposing the model to more diverse examples
- Dilutes the influence of noise and outliers
- Covers previously unseen regions of the input space

Practical limitations:

- Data collection can be expensive and time-consuming
- Benefits diminish as dataset size increases
- Quality trumps quantity—mislabeled data hurts more than it helps

5.2 Data Augmentation

When collecting new data isn't feasible, **data augmentation** artificially expands your dataset by applying domain-appropriate transformations to existing samples.

Domain	Common Augmentations
Images	Rotation, flipping, cropping, color adjustment
Text	Synonym replacement, back-translation
Audio	Time stretching, pitch shifting, noise addition
Time series	Jittering, scaling, window slicing

Key principle: Augmentations must create realistic variations. Rotating a handwritten digit by $15°$ makes sense; rotating it $180°$ creates a different problem entirely.

5.3 Feature Engineering and Selection

Too many features relative to training samples creates opportunities for spurious correlations. **Feature selection** reduces model complexity by retaining only the most informative inputs.

Effective approaches:

- **Filter methods:** Use statistical tests (correlation, mutual information) to rank features
- **Wrapper methods:** Iteratively remove least important features based on model performance
- **Embedded methods:** Let L1 regularization automatically zero out irrelevant features

5.4 Handling Class Imbalance

Imbalanced datasets cause models to overfit to majority classes, achieving high training accuracy but poor generalization.

Resampling strategies:

- **Oversample minority:** Duplicate or synthesize new minority samples (e.g., SMOTE)
- **Undersample majority:** Remove majority samples to balance distribution

Abbas El-Hajj Youssef

- **Hybrid:** Combine both approaches for optimal balance

Data-Centric Checklist

Before adding model complexity, verify:

- ☐ All available data is being utilized
- ☐ Appropriate augmentations have been applied
- ☐ Features are informative and non-redundant
- ☐ Classes are reasonably balanced
- ☐ Data quality has been validated

Remember: Better data beats a better algorithm.

6 Combating Overfitting: Regularization

Beyond data manipulation, we can directly modify the model's learning objective to discourage excessive complexity. This is achieved through a powerful family of techniques called **regularization**.

6.1 The Regularization Principle

The core idea of regularization is to add a **penalty term** to the model's loss function that discourages large parameter values. Models with large parameters can create very complex, "wiggly" decision boundaries that perfectly fit training data's noise. By penalizing large parameters, we encourage the model to find simpler, smoother solutions that are more likely to generalize.

General Regularization Framework

Instead of minimizing just the loss function, we minimize a regularized objective:

$$J(\boldsymbol{\theta}) = \underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{Data Fit Term}} + \underbrace{\lambda \cdot R(\boldsymbol{\theta})}_{\text{Complexity Penalty}} \tag{6}$$

where:

- $\mathcal{L}(\boldsymbol{\theta})$: Original loss function (e.g., MSE for regression, cross-entropy for classification)
- $R(\boldsymbol{\theta})$: Regularization term measuring model complexity (a function of the parameters)
- $\lambda \geq 0$: Regularization strength hyperparameter—controls the tradeoff

This creates a principled trade-off between fitting the training data well and keeping the model simple.

The hyperparameter λ (lambda) controls the strength of this trade-off:

- $\lambda = 0$: No regularization—just the original loss function (risk of overfitting)
- λ small (e.g., 0.01): Slight preference for simpler models
- λ large (e.g., 10): Strong preference for simplicity (risk of underfitting)

El-Hajj Youss

6.2 L2 Regularization (Ridge)

L2 regularization, also known as **Ridge regression** when applied to linear models, adds a penalty proportional to the **square of the magnitude** of the parameters.

L2 Regularization (Ridge)

L2 regularization adds the sum of the squared parameters to the original loss. For linear regression with Mean Squared Error (MSE), the cost function becomes:

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}_{\text{MSE Loss}} + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}_{\text{L2 Penalty Term}} \tag{7}$$

where:

- m: Number of training examples
- n: Number of features
- λ: Regularization hyperparameter
- **Note:** The bias/intercept term θ_0 is typically *not* regularized (the sum starts at $j = 1$)

Why exclude θ_0? The bias term shifts the prediction up or down and doesn't affect model complexity. Regularizing it would prevent the model from matching the mean of the target values.

Effects of L2 regularization:

- **Weight shrinkage:** Large parameters are penalized quadratically, forcing them toward (but rarely to exactly) zero
- **Smooth predictions:** Smaller parameters make the model less sensitive to small changes in input values
- **Distributed influence:** When features are correlated, L2 distributes weight among them rather than relying heavily on one
- **Numerical stability:** Improves the conditioning of optimization problems

L2 regularization produces **dense models**—all features remain in the model with small but non-zero weights.

Abbas El-H

6.3 L1 Regularization (Lasso)

L1 regularization, also known as **Lasso regression** (Least Absolute Shrinkage and Selection Operator), adds a penalty proportional to the **absolute value** of the parameters.

L1 Regularization (Lasso)

L1 regularization adds the sum of the absolute values of the parameters to the original loss:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}|\theta_j| \tag{8}$$

The sparsity effect:

L1 regularization has a unique and powerful property—it can drive some parameters to be **exactly zero**. This occurs because the L1 penalty has a constant gradient magnitude (either $+1$ or -1), creating a constant "push" toward zero regardless of the parameter's current value.

Why is this useful?

Parameters that are exactly zero mean those features are completely excluded from the model. This makes L1 regularization excellent for:

- **Automatic feature selection:** Irrelevant features get zero weight and are effectively removed
- **Model interpretability:** Fewer non-zero parameters create simpler, more interpretable models
- **Computational efficiency:** Sparse models require less memory and computation
- **High-dimensional problems:** When you have many features but suspect only a few are relevant

L1 regularization produces **sparse models**—many parameters become exactly zero.

6.4 L1 vs. L2: Comparison and Geometric Intuition

The choice between L1 and L2 regularization depends on your problem characteristics and goals. Figure 8 provides geometric intuition for why L1 tends to produce sparse solutions while L2 does not.

Abbas El-Hajj Youssef

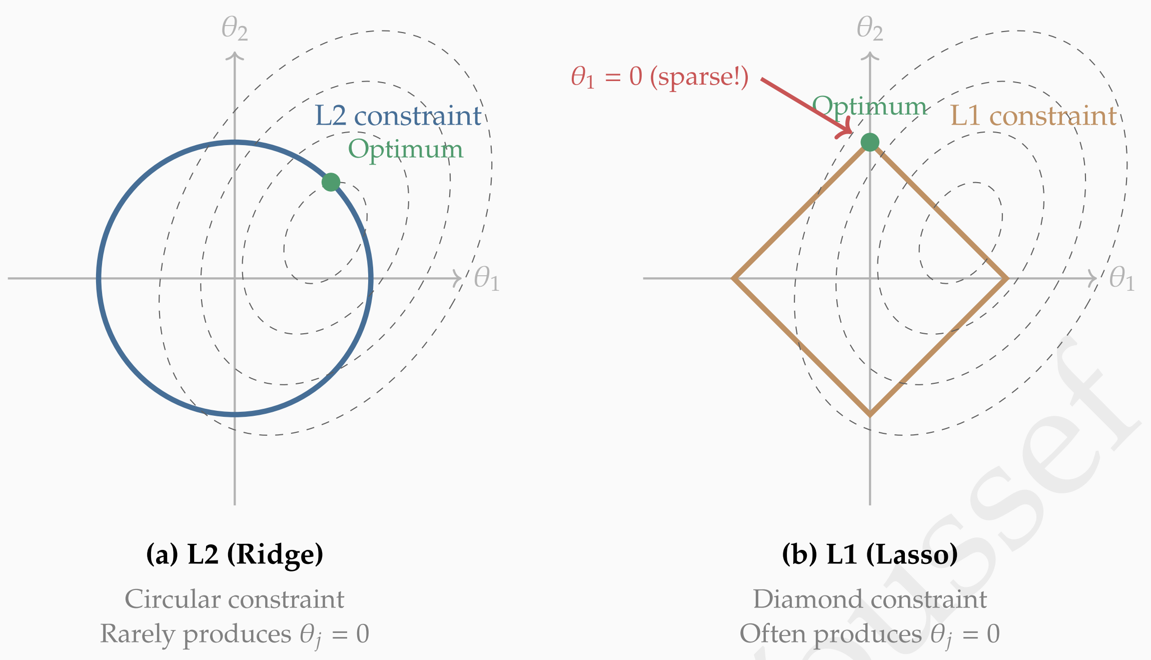

Figure 8: *Geometric interpretation of L1 vs L2 regularization. The ellipses represent contours of the loss function $J(\boldsymbol{\theta})$ (lower loss toward the center). The colored regions show the constraint $R(\boldsymbol{\theta}) \leq t$ for some threshold t. The optimum occurs where a loss contour first touches the constraint region. For L2 (circle), this rarely occurs on an axis (where a parameter is zero). For L1 (diamond), the corners lie on the axes, making $\theta_j = 0$ solutions more likely.*

The following table summarizes the key differences:

Aspect	L1 (Lasso)	L2 (Ridge)
Penalty Form	$\sum_{j=1}^{n} \lvert\theta_j\rvert$	$\sum_{j=1}^{n} \theta_j^2$
Geometry	Diamond (sharp corners)	Circle (smooth)
Sparsity	Produces sparse models	Produces dense models
Parameters	Many become exactly zero	Shrink toward but rarely reach zero
Feature Selection	Automatic	Not automatic
Best Use Case	Many features, few relevant	General-purpose regularization
Correlated Features	Picks one arbitrarily	Distributes weight among all
Optimization	More computationally expensive	Simpler, closed-form solution exists

6.5 Implementing L2 Regularization in Gradient Descent

Regularization modifies the optimization process. When using gradient descent, the gradients must account for the regularization term. For L2 regularization, the gradient of the cost function with respect to parameter θ_j (for $j \geq 1$, excluding the bias term) is:

$$\frac{\partial J}{\partial \theta_j} = \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}}_{\text{Standard gradient}} + \underbrace{\frac{\lambda}{m} \theta_j}_{\text{Regularization term}} \tag{9}$$

This leads to the update rule for gradient descent:

Abbas El-Hajj Youssef

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \tag{10}$$

$$= \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \tag{11}$$

The term $(1 - \alpha\frac{\lambda}{m})$ is slightly less than 1 (assuming reasonable values of α and λ), so each update multiplies the parameter by a factor less than 1 before applying the standard gradient step. This is often called **weight decay** because the parameter "decays" (shrinks) toward zero at each iteration.

Important: The bias/intercept term θ_0 uses the standard update rule *without* the regularization term:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right) \tag{12}$$

il Youssef

6.6 Tuning the Regularization Parameter λ

Choosing the right value for λ is crucial. Too small, and regularization has no effect (overfitting persists). Too large, and the model becomes too simple (underfitting).

Systematic Approach to Tuning λ

Procedure:

1. Define a range of λ values to test, typically on a logarithmic scale:

$$\lambda \in \{0.001, 0.01, 0.1, 1, 10, 100\}$$

2. For each value of λ:
   - Train the model on the training set with that λ
   - Evaluate performance on the validation set
   - Record the validation performance
3. Select the λ that gives the **best validation performance**
4. Optionally, perform a finer search around the best value:

$$\text{If } \lambda^* = 1 \text{ was best, try } \{0.5, 0.7, 1, 1.5, 2\}$$

5. After selecting λ, evaluate the final model on the test set

Important: Never use the test set to choose λ—only the validation set.

6.7 Dropout for Neural Networks

While standard regularization techniques such as L1 and L2 penalties are widely applicable across model types, neural networks benefit from specialized methods. Among these, **dropout** has proven particularly effective.

Dropout Regularization

During training, dropout randomly "drops out" (sets to zero) a fraction p of neuron activations at each iteration:

- **Training phase:** Each neuron is independently deactivated with probability p (commonly between 0.2 and 0.5).
- **Testing phase:** All neurons are active, with outputs appropriately scaled to account for the dropout rate.

Modern implementation: Most frameworks employ **inverted dropout**, scaling activations by $\frac{1}{1-p}$ during training, eliminating the need for scaling at test time and improving computational efficiency.
Benefits of dropout:

- Reduces co-adaptation of neurons, encouraging independent feature learning.
- Promotes redundant representations, providing multiple pathways to the same output.
- Acts as an implicit ensemble of different network architectures.
- Serves as a strong regularizer without explicit weight penalties.

Intuition Behind Dropout

Consider a committee in which members are occasionally absent. Each present member must contribute effectively on its own. Similarly, dropout forces the network to remain robust despite missing units, encouraging generalized, redundant, and resilient feature representations, thereby reducing variance.

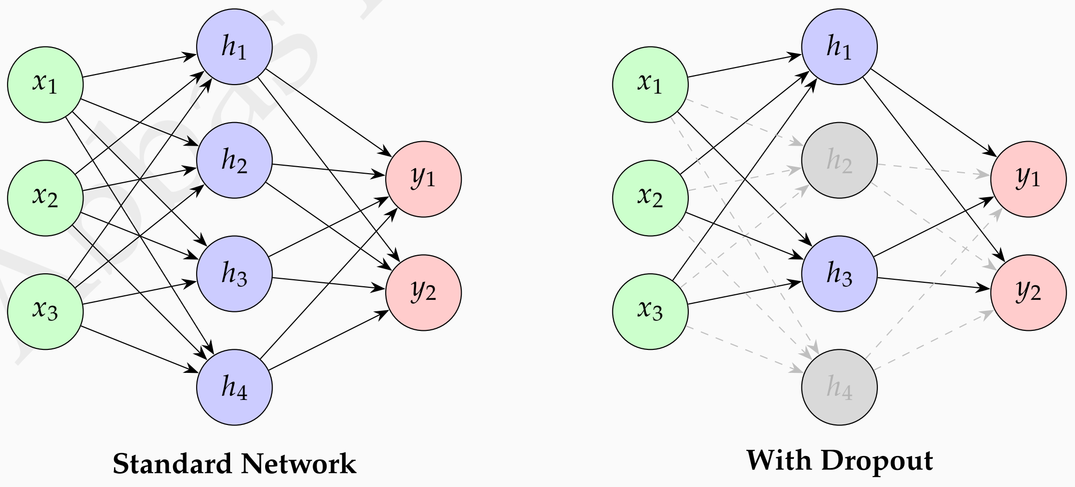

Figure 9: *During dropout, neurons are randomly ignored during forward/backward passes. Grayed-out units (h_2 and h_4) are temporarily removed from this training iteration, forcing remaining neurons to learn robust, independent representations.*

7 Case Studies and Practical Examples

Let's apply these concepts to concrete scenarios. These examples demonstrate how model complexity, regularization, and data splitting principles play out with real numbers.

Case 1: Choosing Model Complexity

You fit three polynomial regression models to a dataset of 10 points generated from a quadratic function with noise. The training and testing Mean Squared Errors (MSE) are:

Model	Degree	MSE (Train)	MSE (Test)
A (Linear)	1	4.2	5.1
B (Quadratic)	2	**1.8**	**2.3**
C (9th Degree)	9	0.0	15.7

Analysis:

- **Model A (Underfitting):** High training error (4.2) indicates the linear model is too simple to capture the quadratic relationship. Both train and test errors are high, typical of underfitting.
- **Model C (Overfitting):** Training MSE of exactly 0 means it perfectly interpolated all 10 training points. However, test MSE of 15.7 shows catastrophic failure on new data—a massive generalization gap of 15.7.
- **Model B (Optimal):** Matches the true data-generating process (quadratic). Low training error (1.8) and lowest test error across the three models (2.3) with a small gap (0.5). This model generalizes best.

Lesson: The optimal model complexity matches the true underlying complexity. More complex isn't better—it's about finding the right fit.

Case 2: Tuning the Regularization Parameter (λ)

You train a Linear Regression model with L2 Regularization on a dataset with multicollinearity, testing different values of λ:

λ	Parameters (θ_1, θ_2)	Train MSE	Test MSE
0.0	(5.2, -3.8)	1.2	6.3
0.1	(3.1, -2.0)	1.8	3.5
1.0	**(1.5, -0.9)**	**2.5**	**2.7**

Analysis:

- $\lambda = 0$ **(No Regularization):** Large weights (5.2, -3.8) indicate overfitting. Training error is lowest (1.2), but test error is highest (6.3)—a gap of 5.1 signals severe overfitting.
- $\lambda = 0.1$ **(Light Regularization):** Weights shrink to (3.1, -2.0). Training error increases slightly to 1.8, but test error drops dramatically to 3.5—already showing the benefit of regularization.
- $\lambda = 1.0$ **(Optimal):** Further weight shrinkage to (1.5, -0.9). Training error is 2.5, but

test error reaches its minimum at 2.7—a tiny gap of only 0.2. Despite "worse" training performance, this model generalizes best.

Lesson: The best model is not the one with the lowest training error, but the one with the best test error. Regularization helps by trading a small increase in training error for a large decrease in test error.

Case 3: Impact of Data Splitting

You train a Decision Tree model on a dataset of 1000 samples, experimenting with different train-test splits:

Train Size	Test Size	Train Acc	Test Acc	Gap
800	200	99%	85%	14%
500	500	98%	82%	16%
200	800	95%	84%	11%

Analysis:

- **Is there overfitting?** Yes, in all cases. Gaps of 11-16% are substantial. A properly regularized model should maintain 95-99% training accuracy without dropping to 82-85% on test data.
- **Effect of training set size:** More training data (800 samples) gives the highest training accuracy (99%) but doesn't help test performance much. With only 200 training samples, the model can't learn as well (95% train accuracy).
- **Which split is most reliable?** The 200/800 split provides the most reliable test score because:
  - Larger test sets have lower variance in performance estimates
  - 800 test samples provide more statistical confidence
  - The test score of 84% is more trustworthy than 85% from only 200 test samples

Lesson: A larger test set provides more reliable performance estimates. The tradeoff is that smaller training sets may limit model performance, but for evaluation purposes, test set size is crucial for confidence in generalization estimates.

8 Chapter Summary

This chapter addressed the central challenge of generalization in machine learning, providing a comprehensive journey from theoretical foundations to practical solutions.

✓ **Core Takeaways**

Fundamental Concepts:

- Machine learning's goal is to build models that **generalize** to new data, not memorize training data
- **Overfitting** (high variance): Model too complex, memorizes training data including noise
- **Underfitting** (high bias): Model too simple, fails to capture underlying patterns
- The **Bias-Variance Tradeoff** explains this mathematically: Total Error = Bias^2 + Variance + Irreducible Error

Diagnostic Tools:

- **Performance gap**: Large difference between train and test performance indicates overfitting
- **Learning curves**: Visualize training dynamics; diverging curves signal overfitting
- **Data splitting**: Proper train/validation/test partitioning prevents data leakage
- **Early stopping**: Halt training when validation performance degrades

Solutions:

- **Data-centric approaches**: Collect more data, augment existing data, select informative features, balance classes
- **L2 regularization (Ridge)**: Shrinks all weights, produces smooth models, general-purpose regularizer
- **L1 regularization (Lasso)**: Creates sparse models, automatic feature selection, useful when many features are irrelevant
- **Dropout**: Specialized for neural networks, prevents neuron co-adaptation

Abbas