Machine Learning Scribe: Overfitting and Regularization

Course Code: CS60020

Instructor: Prof. Aritra Hazra

Scribe: Sushrut Joshi

Date: 3 April 2025

Overfitting Fundamentals

Overfitting occurs when a model learns not just the underlying patterns in the training data, but also the noise or randomness. This leads to poor generalization on unseen data.

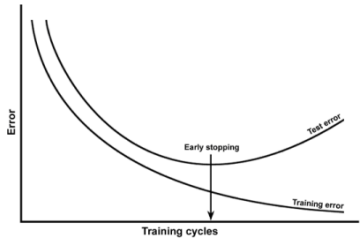
Types of Noise and Overfitting

1. **Stochastic Noise**:
   This refers to the **random noise inherent in the data** — due to measurement errors, natural randomness, etc. If a model fits this noise, it's overfitting because it's modeling randomness rather than signal.

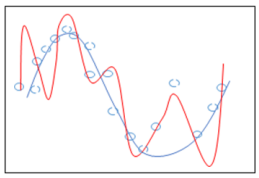
2. **Deterministic Noise**:
   This is caused by the **mismatch between the complexity of the true function and the hypothesis class**. For example, if the true target function is quadratic and we're using a linear model, we can't capture the true function without error. Attempting to use an excessively complex model (e.g., high-degree polynomial) to fit such data introduces overfitting due to trying to force-fit the noise.

Error vs. Number of Training Examples

- **As the number of training examples increases**, overfitting **decreases**, because the model gets a more complete picture of the underlying data distribution.



- **More noise** (stochastic or deterministic) causes **more overfitting**, because the model might try to fit patterns that don't generalize.
- **Higher target function complexity** also increases overfitting, especially if the model is powerful enough to memorize.



Relation to VC-Dimension

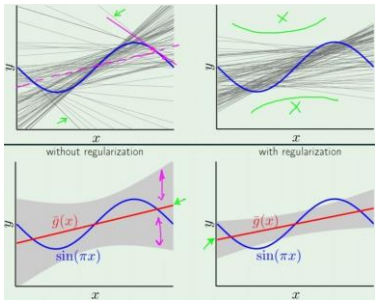
- **VC-dimension (Vapnik–Chervonenkis dimension)** measures the capacity or complexity of a model class — how well it can shatter data points.
- The **generalization bound** derived using VC-dimension gives an upper bound on how bad the test error could be, but it's often **too pessimistic** in practice.
- So, even though theory allows for high worst-case errors, the actual **test error (E_{out}) is usually much better** than the bound.

Remedies for Overfitting

1. **Regularization**:
   Penalize complex models (e.g., L1 or L2 regularization). This constrains the hypothesis space and discourages overfitting.
2. **Validation**:
   Use a validation set to tune model complexity or hyperparameters. Early stopping and cross-validation fall under this.

Regularization

Regularization is a process that converts the answer of a problem to a simpler one.



In machine learning, especially in **polynomial regression or neural networks**, the model has the **freedom to fit many different functions** (i.e., many "possible fits") — especially when it's highly flexible or complex.

- Without any restrictions, the model may **fit the training data perfectly**, even modeling the noise. This is **overfitting**, and it leads to **high variance** — the model performs well on training data but poorly on unseen data.

Regularization adds constraints to the hypothesis space, so:

- You're no longer letting the model pick **any arbitrary function**.
- It penalizes overly complex models (e.g., large weights or high-degree polynomials).
- This **shrinks the solution space** — meaning only a **subset of simpler models** are considered.
- As a result, the model becomes **less sensitive to fluctuations in the data → lower variance**, **better generalization**.

This constrains weights to control model flexibility.

Mathematics behind this 😊

Legendre Polynomials

- **Orthogonal polynomials** defined on $[-1, 1]$.
- Used as **basis functions** in function approximation and regression.
- Help reduce **feature correlation** and improve **numerical stability**.
- Common in physics, numerical methods, and **regularized machine learning**.
- First few:
  $L_0(x) = 1$,
  $L_1(x) = x$,
  $L_2(x) = \frac{1}{2}(3x^2 - 1)$,
  etc.

Why Use Legendre Polynomials in Hypothesis Space?

1. **Orthogonal Basis**
   - Reduces feature correlation
   - Leads to more stable regression
2. **Improved Numerical Stability**
   - Prevents issues with ill-conditioned matrices
   - Better for high-degree polynomial fitting
3. **Efficient Nonlinear Modeling**
   - Allows modeling complex functions using a linear combination of nonlinear bases
4. **Better Control of Model Complexity**
   - Easy to truncate at desired degree (e.g., H_2, H_{10})
   - Supports both hard and soft regularization
5. **Decorrelated Features**
   - Unlike $x, x^2, x^3, \ldots$, Legendre terms are uncorrelated → better generalization
6. **Compatible with Regularization**
   - Works well with L2 (Ridge) regularization to manage bias-variance tradeoff

1. Legendre Polynomial-Based Hypothesis Space

Legendre polynomials $L_q(x)$ are **orthogonal** (like orthogonal vectors) — this helps reduce correlation among features.

We build our hypothesis as a linear combination:

$$H_Q(x) = \sum_{q=0}^{Q} W_q L_q(x)$$

Let's define:

- $Z = [1, L_1(x), L_2(x), ..., L_n(x)]$ → **transformed input**
- Training set becomes (Z_n, y_n)

2. Linear Regression in Z-space

Now we want to find weights W that minimize training error:

$$E_{in}(W) = \frac{1}{N} \sum_{n=1}^{N} (W^T Z_n - y_n)^2 = \frac{1}{N} (ZW - Y)^T (ZW - Y)$$

Unconstrained (basic) solution:

$$W_{\text{lin}} = (Z^T Z)^{-1} Z^T Y$$

This solution can **overfit** if Q is large (high polynomial degree).

3. Hard Constraint: Hypothesis Reduction

If we want to go from hypothesis space H_{10} to H_2, we:

- Set $W_q = 0$ for all $q > 2$
- This is a **hard constraint**: strictly limiting the model's complexity.

4. Soft Constraint: Regularization

Instead of setting weights to zero, we **penalize their magnitude**:

> **Minimize error, subject to weight magnitude constraint:**

$$\text{minimize } E_{in}(W) \quad \text{subject to } W^T W \leq C$$

This is equivalent to **L2 regularization (Ridge)**:

$$\min \left[E_{in}(W) + \frac{\lambda}{N} W^T W \right]$$

- λ: regularization strength
- C: constraint limit
- These two are **duals** — adjusting one affects the other

5. Gradient Connection (Deriving the Equivalence)

From optimization theory:

- We want to **minimize** the regularized objective
- First-order condition (gradient = 0):

Equation (1):

$$\nabla E_{in}(W_{\text{reg}}) \propto -W_{\text{reg}}$$

Equation (2):

$$\nabla E_{in}(W_{\text{reg}}) = -\frac{2\lambda}{N} W_{\text{reg}}$$

Equation (3):

$$\nabla E_{in}(W_{\text{reg}}) + \frac{2\lambda}{N} W_{\text{reg}} = 0$$

This shows that minimizing:

$$E_{in}(W) + \frac{\lambda}{N} W^T W$$

is **equivalent to** minimizing $E_{in}(W)$ under the constraint $W^T W \leq C$

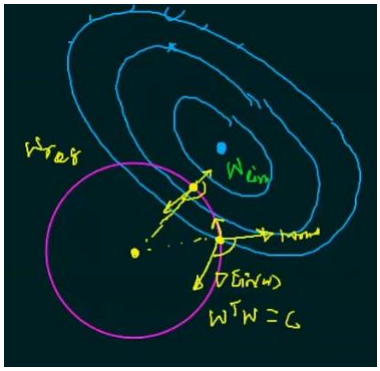
Why Duality Matters

- Hard constraints ($W^T W \leq C$) are easier to analyze **theoretically**, e.g., when applying **VC-dimension**
- Soft constraints (penalizing $W^T W$) are easier to **optimize in practice** (solving regression problems)

So we:

- **Analyze with constraints**
- **Solve with regularization**

Relation of C with W



> "If $C \downarrow \Rightarrow \lambda \uparrow \Rightarrow$ **We get mostly** E_{in}" "If $C \uparrow \Rightarrow \lambda \downarrow \Rightarrow$ **W dominates**"

- Small C (tight constraint): high regularization, model is **simple**, fits less → low variance, high bias
- Large C: low regularization, model is **complex**, can overfit

No choice gives **perfect low error** — it's a **bias-variance tradeoff**

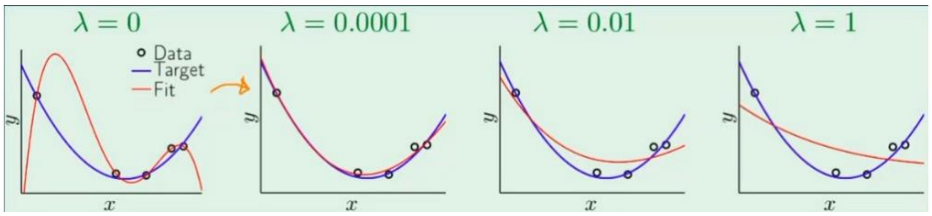
Augmented Error function

$$E_{aug}(W) = E_{in}(W) + (\frac{\lambda}{N})W^T W = \frac{1}{N}[(ZW - Y)^T(ZW - Y) + \lambda W^T W]$$

In equation both part are quadratic hence we can solve it using quadratic programming

$$\nabla E_{in}(W) = 0 \Rightarrow Z^T(ZW - Y + \lambda W = 0)$$
$$W_{req} = (Z^T Z + \lambda I)^{-1} Z^T Y$$
$$Opposed to W_{in} = (Z^T)^{-1} Z^T Y$$

This implies
If λ is large $\Rightarrow$ Constraining Solution that is Making $W_{reg} = 0$
More λ makes the **circle smaller** and **decreases** chances of getting to W_{in}
Makes the **curve flatter(smother)**
If λ is small $\Rightarrow$ Encapsulating $W_{in} inside of it that is making W_{reg} = W_{in}$



over – Fitting – – > Reg1 – – > Reg1' ... – –– > ...under – Fitting

Choice of λ is very Important. We have to choose λ, validate it and choose out which is the best.

Weight decay strategy

Instead of solving in **one-shot** (like with normal equation), we use **gradient descent**:

$$W(t+1) = W(t) - \eta \nabla \left[E_{in}(W(t)) + \frac{2\lambda}{N} W(t) \right]$$

Breaking it down:

$$W(t+1) = W(t) \left[1 - \frac{2\eta\lambda}{N} \right] - \eta \nabla E_{in}(W(t))$$

- η: learning rate
- λ: regularization strength
- The term $W(t)[1 - \cdots]$ is called **weight decay**
- This shrinks weights on each step → prevents overfitting

In neural networks, the weight penalty becomes:

$$\Omega(h) = \sum\sum\sum (W_{ij}^{l})^2$$

- Sum over all weights in all layers
- This is the same idea: penalize large weights to control model complexity

Weighted Regression

Instead of treating all weights equally, we **weight the penalty per coefficient**:

$$\sum \gamma_q W_q^2 \leq C$$

- γ_q: custom weight for each coefficient
  - If $\gamma_q = 2^q$ → penalize higher-order terms more → favors **smooth, low-degree fits**
  - If $\gamma_q = 2^{-q}$ → allows higher-degree terms → can fit **complex, noisy curves**

Choice of C and λ:

High-order polynomials capture noise.
We prefer ≤ C to constrain weight magnitude and favor smoother, simpler models.
This aligns with Occam's Razor — simpler explanations are better.

Selecting λ is critical:
- Too large → underfit.
- Too small → overfit.

Choose λ through validation or cross-validation.
Optimal λ balances complexity and generalization.