Gradient Boosting: Complete Study Notes

> **Core Principle:** Gradient Boosting is performing Gradient Descent in Function Space

Overview: Ensemble Methods

> **Ensemble Methods:** Machine learning techniques that combine multiple learning algorithms to obtain better predictive performance than could be obtained from any constituent learning algorithm alone.

Types of Ensemble Methods

- **Bagging (Bootstrap Aggregating):**
  - Train multiple models on different subsets of data
  - Combine predictions by averaging (regression) or voting (classification)
  - Example: Random Forest
  - Reduces variance
- **Boosting:**
  - Train models sequentially, each correcting errors of previous ones
  - Examples: AdaBoost, Gradient Boosting, XGBoost
  - Reduces bias

```
Ensemble Methods ├── Bagging │ ├── Random Forest │ └── Extra Trees └── Boosting ├── AdaBoost
├── Gradient Boosting └── XGBoost/LightGBM
```

> **Bias-Variance Trade-off:**
> • **Bias:** Error from overly simplistic assumptions
> • **Variance:** Error from sensitivity to small fluctuations in training set
> • **Bagging:** Reduces variance, keeps bias same
> • **Boosting:** Reduces bias, might increase variance

1. Function Space vs Parameter Space

Parameter Space: The space of all possible parameter values for a given model structure. Traditional optimization happens here.

Function Space: The space of all possible functions. Gradient boosting optimizes in this space by building functions incrementally.

Parameter Space Approach (Traditional)

Example: Linear regression $y = mx + b$

- Model structure is fixed: $y = f(x) = mx + b$
- We optimize parameters m and b directly
- Loss function: $L(y, \hat{y}) = \Sigma(y_i - \hat{y}_i)^2$
- Update rule: $m_1 = m_0 - \eta \times \partial L/\partial m$
- Dimension: Low (number of parameters)

```
Parameter Update: θ₁ = θ₀ - η × ∂L/∂θ where θ = [m, b] are parameters η = learning rate
```

Function Space Approach (Gradient Boosting)

- Model structure evolves: $F_0(x)$, $F_1(x)$, $F_2(x)$, ...
- We optimize the entire function $F(x)$
- Update rule: $F_1(x) = F_0(x) + h_1(x)$
- $h_1(x)$ is a weak learner (decision tree)
- Dimension: High (function values at all points)

```
Function Update: F_{m+1}(x) = F_m(x) + α × h_{m+1}(x) where F_m(x) = current ensemble h_{m+1}(x) = new weak learner α = learning rate
```

Weak Learner: A learning algorithm that performs only slightly better than random guessing. In gradient boosting, typically shallow decision trees (depth 1-6).

2. Direction of Loss Minimization

Gradient: Vector of partial derivatives indicating the direction of steepest increase of a function.

> We move in the negative gradient direction to minimize loss.

In gradient boosting, we compute gradients with respect to function values (not parameters):

```
For Mean Squared Error: L(y, F(x)) = Σ(yi - F(xi))² Gradient: ∂L/∂F(xi) = ∂/∂F(xi) [Σ(yj - F(xj))²] = -2(yi - F(xi)) Negative Gradient: -∂L/∂F(xi) = 2(yi - F(xi)) = residuals
```

> **Residuals:** The difference between actual and predicted values: $ri = yi - F(xi)$. These represent the "errors" our current model makes.

Key Insight: The negative gradient gives us the residuals!

```
Gradient Direction Visualization: Loss ↑ | Current F(xi) | ○ | /| | / | ← Gradient points up (increasing loss) | / | | / ↓ Negative gradient points down | / (decreasing loss) | / | / +----------→ F(xi) Move in negative gradient direction = Move toward yi
```

Interpretation of Residuals

- **Positive residuals ($yi > F(xi)$):** Model prediction too low → need to increase $F(xi)$
- **Negative residuals ($yi < F(xi)$):** Model prediction too high → need to decrease $F(xi)$
- **Zero residuals ($yi = F(xi)$):** Perfect prediction → no change needed

3. Update the Function

> **Pseudo-residuals:** The negative gradients that weak learners try to predict. For squared loss, these are just the regular residuals.

Step-by-step Algorithm

Step 0: Initialize
$F_0(x) = \text{argmin}\_\gamma \ \Sigma \ L(yi, \gamma)$
For squared loss: $F_0(x) = \text{mean}(y) = \bar{y}$

Step 1: Calculate Pseudo-residuals
$rim = -[\partial L(yi, F(xi))/\partial F(xi)]\_\{F=F\_\{m-1\}\}$
For squared loss: $rim = yi - F\_\{m-1\}(xi)$

Step 2: Train Weak Learner

Train hm(x) to predict pseudo-residuals: hm(x) ≈ rim

Typically use decision trees with limited depth

Step 3: Find Optimal Step Size

γm = argmin_γ Σ L(yi, F_{m-1}(xi) + γ × hm(xi))

This is line search in function space

Step 4: Update Function

Fm(x) = F_{m-1}(x) + γm × hm(x)

> **Line Search:** Optimization technique to find the optimal step size γ along the search direction h(x).

```
Complete Update Formula: F_m(x) = F_{m-1}(x) + γ_m × h_m(x) where: - F_{m-1}(x) is the current ensemble - h_m(x) is trained on pseudo-residuals - γ_m is the optimal step size
```

4. Iterate Process

The algorithm repeats steps 1-4 for M iterations:

```
Iteration Flow: F₀(x) = ȳ ↓ r₁ = y - F₀(x) → h₁(x) → γ₁ → F₁(x) = F₀(x) + γ₁h₁(x) ↓ r₂ = y - F₁(x) → h₂(x) → γ₂ → F₂(x) = F₁(x) + γ₂h₂(x) ↓ r₃ = y - F₂(x) → h₃(x) → γ₃ → F₃(x) = F₂(x) + γ₃h₃(x) ↓ ...continue until convergence or M iterations... ↓ Final: F_M(x) = F₀(x) + Σγᵢhᵢ(x)
```

Stopping Criteria

- **Fixed iterations:** Stop after M boosting rounds
- **Validation loss:** Stop when validation error stops improving
- **Residual threshold:** Stop when residuals become very small

5. Working Example

```
Visual Example - Fitting a Curve: Data: {(x₁,y₁), (x₂,y₂), ..., (xₙ,yₙ)} Stage 0: F₀(x) = ȳ (horizontal line) │ ├─ Residuals₁ = y - ȳ ├─ Train h₁(x) to predict residuals₁ ├─ F₁(x) = ȳ + γ₁h₁(x) (adds some shape) │ ├─ Residuals₂ = y - F₁(x) ├─ Train h₂(x) to predict residuals₂ ├─
```

```
F₂(x) = F₁(x) + γ₂h₂(x) (refines fit) │ └─ Continue until good fit achieved... Each hᵢ(x)
corrects remaining errors
```

6. Loss Functions and Applications

> **Loss Function L(y, F(x)):** Measures how well our model F(x) predicts the true values y. Different losses lead to different types of gradient boosting.

Common Loss Functions

Problem Type	Loss Function	Gradient
Regression	MSE: $\frac{1}{2}(y - F(x))^2$	$y - F(x)$
Regression	MAE: $|y - F(x)|$	$\text{sign}(y - F(x))$
Binary Classification	Log-loss: $\log(1 + e^{-yF(x)})$	$y/(1 + e^{yF(x)})$
Multi-class	Softmax loss	Complex...

7. Key Differences: Gradient Boosting vs Other Methods

Aspect	Gradient Descent	Gradient Boosting	Random Forest
Optimization Space	Parameter space	Function space	Neither (voting)
Training	Single model	Sequential ensemble	Parallel ensemble
Bias/Variance	Depends on model	Reduces bias	Reduces variance
Overfitting Risk	Moderate	High	Low

8. Advantages and Disadvantages

Advantages

- **Flexibility:** Works with any differentiable loss function
- **No distributional assumptions:** Non-parametric approach
- **Feature handling:** Handles mixed data types, missing values
- **Feature selection:** Implicit feature selection through tree splits
- **Robustness:** Robust to outliers (with appropriate loss)
- **Performance:** Often achieves state-of-the-art results
- **Interpretability:** Can analyze feature importance, partial dependence

Disadvantages

- **Sequential training:** Cannot be parallelized easily
- **Overfitting:** Can overfit with too many iterations
- **Hyperparameter sensitivity:** Many parameters to tune
- **Computational cost:** Slower than Random Forest
- **Memory usage:** Stores all trees in memory

9. Regularization Techniques

> **Regularization:** Techniques to prevent overfitting and improve generalization.

- **Learning Rate (Shrinkage):** $F_m(x) = F_{m-1}(x) + \nu \times \gamma_m \times h_m(x)$, where $0 < \nu \leq 1$
- **Tree Constraints:** Limit tree depth, minimum samples per leaf
- **Subsampling:** Train each tree on random subset of data
- **Feature Subsampling:** Use random subset of features per tree
- **Early Stopping:** Stop when validation error increases

10. Popular Implementations

- **Scikit-learn:** GradientBoostingRegressor, GradientBoostingClassifier
- **XGBoost:** Extreme Gradient Boosting with advanced optimizations
- **LightGBM:** Fast, distributed, high performance implementation
- **CatBoost:** Handles categorical features automatically

Summary: Gradient Boosting transforms optimization from parameter space to function space, building complex predictive models by sequentially adding simple weak learners that correct the errors of their predecessors. This approach combines the flexibility of non-parametric methods with the principled optimization of gradient-based techniques.