

Overfitting vs. Underfitting

The model complexity balance

Underfitting \leftarrow Optimal Model \rightarrow Overfitting

Terms:

- ▶ **Underfitting:** Too simple, misses patterns
- ▶ **Overfitting:** Too complex, captures noise
- ▶ **Generalization:** Performance on unseen data

Key Properties:

- ▶ High training error \rightarrow underfitting
- ▶ High validation error \rightarrow overfitting
- ▶ Regularization tackles overfitting



@AlinMinutes

Bias-Variance Tradeoff

Error decomposition

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

Terms:

- ▶ **Bias:** Error from wrong assumptions
- ▶ **Variance:** Sensitivity to data fluctuations
- ▶ **Noise:** Irreducible error

Key Properties:

- ▶ High bias \rightarrow underfitting
- ▶ High variance \rightarrow overfitting
- ▶ Regularization: \downarrow variance, \uparrow bias



Regularization Fundamentals

Constraint-based learning

$$L_{\text{reg}}(\mathbf{w}) = L(\mathbf{w}) + \lambda \cdot R(\mathbf{w})$$

Terms:

- ▶ $L(\mathbf{w})$: Original loss function
- ▶ $R(\mathbf{w})$: Penalty function
- ▶ λ : Regularization strength

Key Properties:

- ▶ Penalizes large weights
- ▶ Promotes simpler models
- ▶ Improves generalization



Vector Norms

Weight magnitude measures

$$\|\mathbf{w}\|_p = \left(\sum_{i=1}^n |w_i|^p \right)^{1/p}$$

Common Norms:

- ▶ $\|\mathbf{w}\|_1 = \sum_{i=1}^n |w_i|$ (L1: sum of absolutes)
- ▶ $\|\mathbf{w}\|_2 = \sqrt{\sum_{i=1}^n w_i^2}$ (L2: Euclidean)

Key Properties:

- ▶ L1: Creates sparse solutions
- ▶ L2: Creates small, distributed weights
- ▶ Different geometric constraints



Ensemble Learning

Multiple models, one prediction

$$f_{\text{ensemble}}(x) = \frac{1}{M} \sum_{i=1}^M f_i(x)$$

Terms:

- ▶ $f_i(x)$: Individual model
- ▶ M : Number of models
- ▶ Var reduction: $\text{Var}(f_{\text{ensemble}}) \approx \frac{\text{Var}(f_i)}{M}$

Key Properties:

- ▶ Reduces variance through averaging
- ▶ Dropout implicit ensemble
- ▶ Diverse models \rightarrow better performance



Bernoulli Distribution

Random variable with two possible outcomes

$$X \sim \text{Bernoulli}(p) \Rightarrow P(X = 1) = p$$

$$E[X] = p, \quad \text{Var}[X] = p(1 - p)$$

Terms:

- ▶ p : Probability of success (outcome 1)
- ▶ $1 - p$: Probability of failure (outcome 0)
- ▶ X : Random variable

Key Properties:

- ▶ Simplest discrete probability distribution
- ▶ Special case of binomial with $n = 1$



L2 Regularization (Ridge)

Squared penalty that shrinks all weights

$$L_{\text{reg}}(\mathbf{w}) = L(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left(\nabla_{\mathbf{w}} L + \lambda \mathbf{w}^{(t)} \right)$$

Terms:

- ▶ $L(\mathbf{w})$: Loss function
- ▶ λ : Regularization strength parameter
- ▶ \mathbf{w} : Model weight vector
- ▶ $\nabla_{\mathbf{w}} L$: Gradient of loss

Key Properties:

- ▶ Proportional weight reduction
- ▶ Reduces variance, increases bias
- ▶ Equivalent to Gaussian prior
- ▶ Does not induce sparsity (unlike L1 regularization)



L1 Regularization (Lasso)

Absolute value penalty that induces sparsity

$$L_{\text{reg}}(\mathbf{w}) = L(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left(\nabla_{\mathbf{w}} L + \lambda \cdot \text{sgn}(\mathbf{w}^{(t)}) \right)$$

Terms:

- ▶ $\|\mathbf{w}\|_1$: L1 norm (sum of absolute values of weights)
- ▶ $\text{sgn}(\mathbf{w})$: Sign function, where $\text{sgn}(w_i) \in \{-1, 0, 1\}$ for $w_i \neq 0$ and $\text{sgn}(0) \in [-1, 1]$
- ▶ λ : Regularization parameter

Key Properties:

- ▶ Forces exact zeros in weights (sparse solution)
- ▶ Enables feature selection
- ▶ Equivalent to Laplace prior (sharp peak)



Elastic Net Regularization

Hybrid approach balancing sparsity and smoothness

$$L_{\text{reg}}(\mathbf{w}) = L(\mathbf{w}) + \lambda_1 \|\mathbf{w}\|_1 + \frac{\lambda_2}{2} \|\mathbf{w}\|_2^2$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left(\nabla_{\mathbf{w}} L + \lambda_1 \text{sgn}(\mathbf{w}^{(t)}) + \lambda_2 \mathbf{w}^{(t)} \right)$$

Terms:

- ▶ λ_1 : L1 regularization parameter (sparsity control)
- ▶ λ_2 : L2 regularization parameter (smoothness control)
- ▶ $\|\mathbf{w}\|_1$: L1 norm (sum of absolute values of weights)
- ▶ $\|\mathbf{w}\|_2^2$: Squared L2 norm (sum of squared weights)

Key Properties:

- ▶ Combines L1 sparsity and L2 stability
- ▶ Handles correlated features effectively
- ▶ Optimal when features outnumber samples



@AlinMinutes

Standard Dropout

Kind of like ensemble of subnetworks

Training: $\mathbf{O}_{Tr} = \mathbf{m} \odot \mathbf{h}$, $\mathbf{m} \sim \text{Bernoulli}(1 - p)$

Inference: $\mathbf{O}_{In} = (1 - p)\mathbf{h}$

Terms:

- ▶ **m**: Binary mask (0=dropped, 1=kept)
- ▶ **h**: Hidden layer activations
- ▶ \mathbf{O}_{Tr} : Output during training (dropout on)
- ▶ \mathbf{O}_{In} : Output during inference (dropout off)
- ▶ p : Probability of dropping a neuron

Key Properties:

- ▶ Scaling during inference by keep probability $(1 - p)$
- ▶ Expectation of training and inference outputs are equal: $E[\mathbf{O}_{Tr}] = E[\mathbf{O}_{In}]$



Inverted Dropout

Training-time scaling for simplified inference

$$\text{Training: } \mathbf{O}_{Tr} = \frac{\mathbf{m} \odot \mathbf{h}}{1 - p}, \quad \mathbf{m} \sim \text{Bernoulli}(1 - p)$$

$$\text{Inference: } \mathbf{O}_{In} = \mathbf{h}$$

Terms:

- ▶ **m**: Binary mask (0=dropped, 1=kept)
- ▶ **h**: Hidden layer activations
- ▶ **O_{Tr}**: Output during training (dropout on)
- ▶ **O_{In}**: Output during inference (dropout off)
- ▶ **p**: Probability of dropping a neuron

Key Properties:

- ▶ Upscales activations during training
- ▶ Expectation of training and inference outputs are equal: $E[\mathbf{O}_{Tr}] = E[\mathbf{O}_{In}]$



Standard Dropout (Quick Proof)

Why scale during inference?

Training:

- ▶ Drop neurons randomly:

$$O_{Tr,i} = m_i \cdot h_i, \quad m_i \sim \text{Bernoulli}(1 - p)$$

- ▶ Expected output:

$$E[O_{Tr,i}] = E[m_i \cdot h_i] = E[m_i] \cdot E[h_i] = (1 - p) \cdot E[h_i]$$

Inference:

- ▶ Without scaling: $O_{In,i} = h_i$
- ▶ Problem: $E[O_{Tr,i}] = (1 - p) \cdot E[h_i] \neq E[h_i] = E[O_{In,i}]$
- ▶ Scale by $(1 - p)$ during inference: $O_{In,i} = (1 - p) \cdot h_i$
- ▶ Now: $E[O_{Tr,i}] = (1 - p) \cdot E[h_i] = E[O_{In,i}]$



Inverted Dropout (Quick Proof)

Why no scaling at inference?

Training:

- ▶ Drop neurons and scale up:
 $O_{Tr,i} = \frac{m_i \cdot h_i}{1-p}, \quad m_i \sim \text{Bernoulli}(1-p)$
- ▶ Expected output: $E[O_{Tr,i}] = E\left[\frac{m_i \cdot h_i}{1-p}\right] = \frac{E[m_i \cdot h_i]}{1-p} = \frac{E[m_i] \cdot E[h_i]}{1-p} = \frac{(1-p) \cdot E[h_i]}{1-p} = E[h_i]$

Inference:

- ▶ No dropout, no scaling: $O_{In,i} = h_i$
- ▶ Check: $E[O_{Tr,i}] = E[h_i] = E[O_{In,i}]$
- ▶ Success! No scaling needed at inference time.

Benefit: Simpler and faster inference! Implemented this way in most libraries



@AlinMinutes

Early Stopping

Temporal regularization

$$\mathbf{w}^* = \mathbf{w}^{(t^*)} \text{ where } t^* = \arg \min_t L_{\text{val}}(\mathbf{w}^{(t)})$$

Terms:

- ▶ $\mathbf{w}^{(t)}$: Weights at training iteration t
- ▶ L_{val} : Validation loss
- ▶ t^* : Optimal stopping iteration

Key Properties:

- ▶ Training path passes near optimal solution
- ▶ Implicit regularization without penalty term
- ▶ Requires validation set and patience parameter



Batch Normalization

Implicit regularization effect

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad y = \gamma \hat{x} + \beta$$

Terms:

- ▶ μ_B, σ_B^2 : Batch mean and variance
- ▶ γ, β : Learnable scale and shift parameters
- ▶ ϵ : Small constant for numerical stability

Key Properties:

- ▶ Reduces internal covariate shift
- ▶ Adds noise during training (regularizes)
- ▶ Smooths loss landscape



@AlinMinutes

Data Augmentation

Dataset-level regularization

$$\tilde{x} = T(x) \quad \text{where} \quad T \in \mathcal{T}$$

Terms:

- ▶ T : Transformation function
- ▶ \mathcal{T} : Set of valid transformations
- ▶ \tilde{x} : Augmented data sample

Key Properties:

- ▶ Increases dataset size and diversity
- ▶ Encodes domain-specific invariances
- ▶ Common examples: rotation, cropping, mixup

