Chapter 5 Multilayer Perceptron

5.3: Generalization in Neural Networks

Kiran Bagale

July 2025

What is Generalization?

- A network **generalizes well** when the input-output mapping computed by the network is correct (or nearly so) for test data never used in creating or training the network
- The term "generalization" is borrowed from psychology
- Test data are assumed to be drawn from the same population used to generate the training data
- Generalization allows the network to perform correctly on unseen examples

Generalization as Curve Fitting

- The learning process can be viewed as a "curve-fitting" problem
- The network acts as a nonlinear input-output mapping
- Generalization is the effect of good nonlinear interpolation of input data
- Multilayer perceptrons with continuous activation functions lead to continuous output functions
- This continuity enables useful interpolation between training points

Good Generalization: Smooth Interpolation

- The network learns from training data points creating a smooth nonlinear mapping
- Successfully interpolates to predict output for new inputs
- The red point shows successful generalization through interpolation

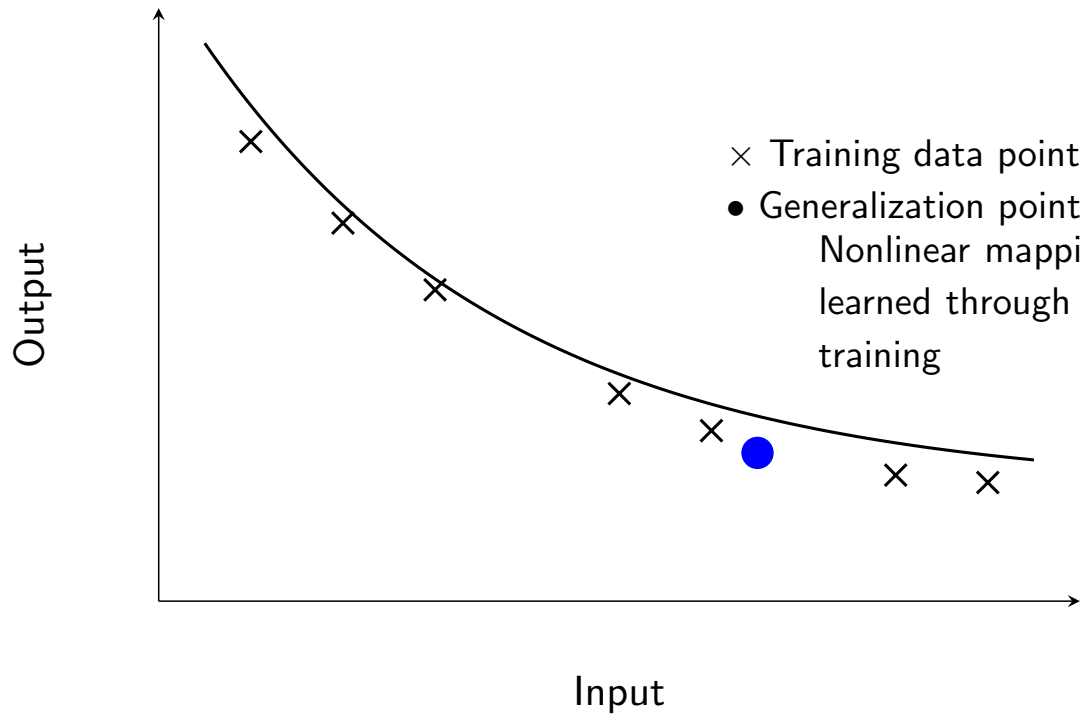

Figure 5.3 (a) Properly fitted nonlinear mapping with good generalization.

Poor Generalization: Overfitting and Memorization

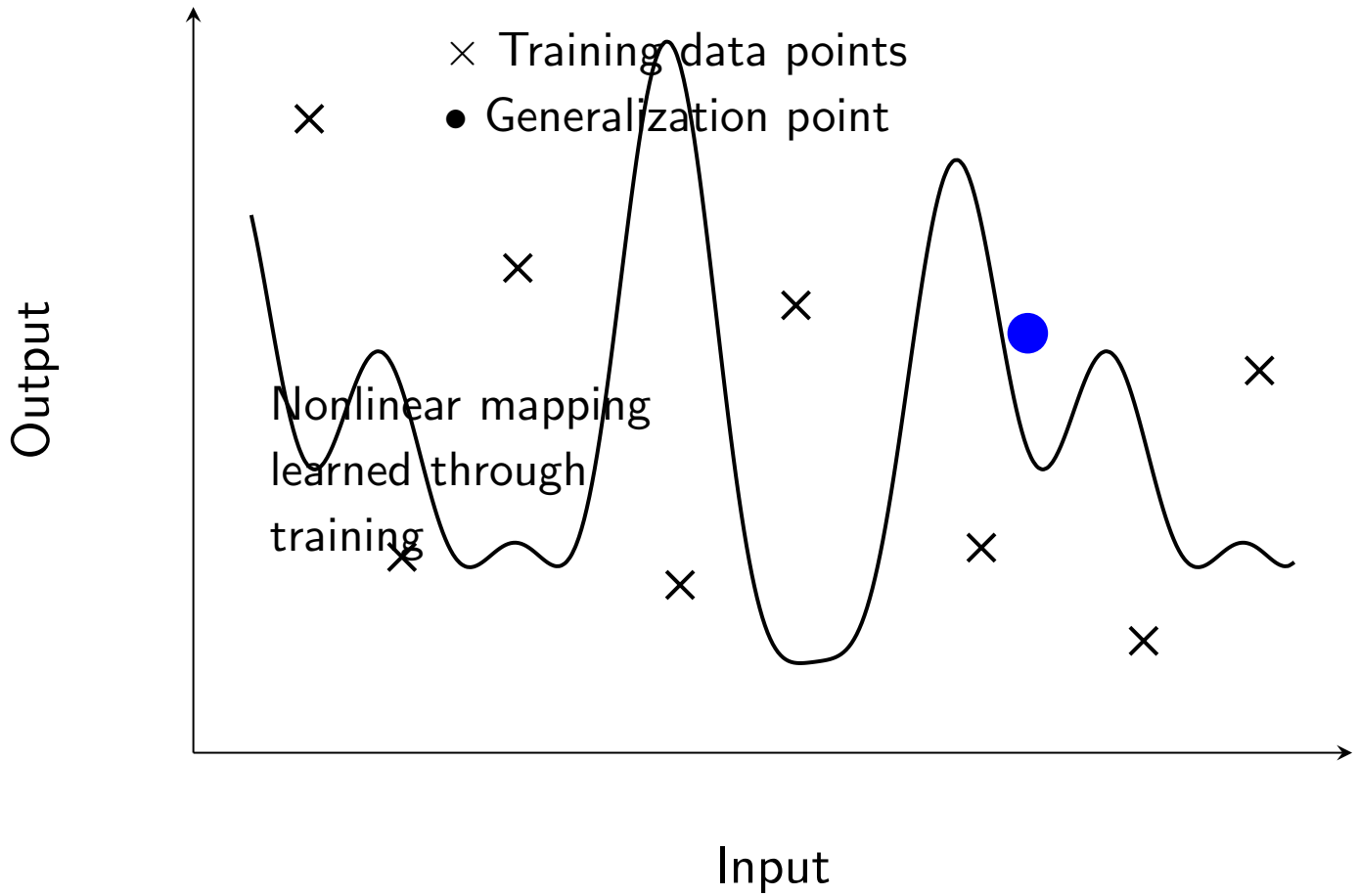

Figure 5.3 (b) Overfitted nonlinear mapping with poor generalization.

Comparison: Good vs Poor Generalization

Good Generalization (a):

- Smooth, continuous curve
- Reasonable interpolation
- Captures underlying trend
- Blue point fits well on curve

Poor Generalization (b):

- Highly oscillating curve
- Memorizes training noise
- Overfits to training data
- Poor prediction for new inputs

Key Lesson: Smooth interpolation leads to better generalization than exact memorization of training data.

The Importance of Smoothness

- Smoothness of input-output mapping is closely related to model-selection criteria
- **Occam's Razor:** Select the "simplest" function in the absence of prior knowledge
- In this context: Choose the smoothest function that approximates the mapping for a given error criterion
- Smooth functions generally demand fewer computational resources

Factors Influencing Generalization

- Generalization is influenced by **three key factors**:
  1. **Size and representativeness** of the training sample
  2. **Architecture** of the neural network
  3. **Physical complexity** of the problem at hand

- We have no control over the physical complexity of the problem
- Two different perspectives for addressing generalization:
  - Fixed architecture $\rightarrow$ Determine training sample size
  - Fixed training sample $\rightarrow$ Determine best network architecture

Training Sample Size: The Rule of Thumb

- For good generalization, the training sample size N should satisfy:

$$N = O\left(\frac{W}{\epsilon}\right)$$

where:

  - W = total number of free parameters (synaptic weights and biases)
  - ϵ = fraction of classification errors permitted on test data
  - $O(\cdot)$ = order of the quantity enclosed

- **Example:** With 10% error rate ($\epsilon = 0.1$), need approximately **10 times** the number of free parameters in training examples

- This follows **Widrow's rule of thumb** for the LMS algorithm

5.4 APPROXIMATIONS OF FUNCTIONS

What is the minimum number of hidden layers in a multilayer perceptron with an input–output mapping that provides an approximate realization of any continuous mapping?

Universal Approximation Theorem

Main Result

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotone-increasing continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0,1]^{m_0}$. Given any function $f \in C(I_{m_0})$ and $\varepsilon > 0$, there exist an integer m_1 and real constants α_i, b_i, and w_{ij} such that:

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi \left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right)$$

approximates $f(\cdot)$ with $|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \varepsilon$

Key Point: Single hidden layer MLPs can approximate any continuous function to arbitrary accuracy.

Approximation Error Bounds

Barron's Result (1993)

For continuous function $f(x)$ with finite first moment C_f:

$$\mathcal{E}_{av}(N) \leq O\left(\frac{C_f^2}{m_1}\right) + O\left(\frac{m_0 m_1}{N} \log N\right)$$

where $C_f = \int_{\mathbb{R}^{m_0}} |\tilde{f}(\omega)| \times \|\omega\|^{1/2} d\omega$

Trade-off:

- Accuracy of best approximation: Large m_1 (many hidden neurons)
- Empirical fit accuracy: Small m_1 (avoid overfitting)

Curse of Dimensionality

Optimal Hidden Layer Size

Risk $\mathcal{E}_{av}(N)$ is minimized when:

$$m_1 \simeq C_f \left(\frac{N}{m_0 \log N} \right)^{1/2}$$

Risk bound: $\mathcal{E}_{av}(N) = O\left(C_f \sqrt{\frac{m_0 \log N}{N}} \right)$

Convergence Rates:

- **Neural networks:** $O((1/N)^{1/2})$ (with log factor)
- **Traditional smooth functions:** $O((1/N)^{2s/(2s+m_0)})$

Advantage: Neural networks avoid exponential dependence on input dimension m_0

Practical Implications

Why Multiple Hidden Layers?

Single layer limitations:

- Global neuron interactions make local improvements difficult
- Theoretical existence practical construction

Two layer advantages:

- **First layer:** Extract local features, partition input space
- **Second layer:** Extract global features, combine local information

Generalization Rule: For good generalization, training samples N should be larger than $\frac{\text{total parameters}}{\text{estimation error variance}}$

Purpose of Cross-Validation

The essence of back-propagation learning is to encode an input–output mapping into the synaptic weights and thresholds of a multilayer perceptron. Cross-validation provides an appealing guiding principle for model selection.

- Network selection problem: choosing the "best" parameterization according to a certain criterion
- Standard tool in statistics for model validation
- Particularly useful when designing large neural networks with good generalization

Cross-Validation: Basic Methodology

Data Partitioning

The available data set is randomly partitioned into:

1. **Training sample** (further divided into two subsets)
2. **Test set** (for final performance evaluation)

Training Sample Division

- **Estimation subset**: used to select the model
- **Validation subset**: used to test or validate the model

Key Principle

Validate the model on a data set different from the one used for parameter estimation to prevent overfitting.

Variants of Cross-Validation

Holdout Method

Standard approach where data is split into training and validation sets once.

Multifold Cross-Validation (K-fold)

- Divide N examples into K subsets
- Train on all subsets except one
- Validate on the remaining subset
- Repeat for K trials, each time using a different validation subset
- Average validation error over all trials

Leave-One-Out Method

- Extreme case: $K = N$ (when data is severely limited)
- Use $N - 1$ examples for training, 1 for validation
- Repeat N times, each time leaving out a different example

Multifold Cross-Validation Illustration

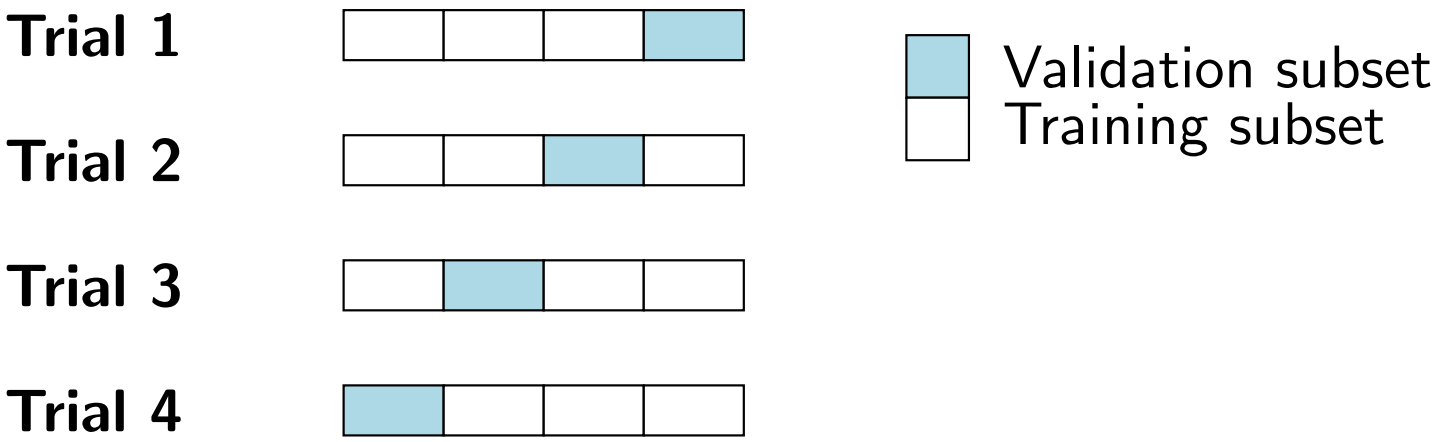

Figure: Multifold cross-validation with K=4 folds

Early-Stopping Method

Procedure

1. Split training data into estimation and validation subsets
2. Train network on estimation subset
3. Periodically test on validation subset
4. Stop training when validation error starts increasing

Key Insight

The minimum point on the validation learning curve indicates optimal stopping to prevent overfitting.

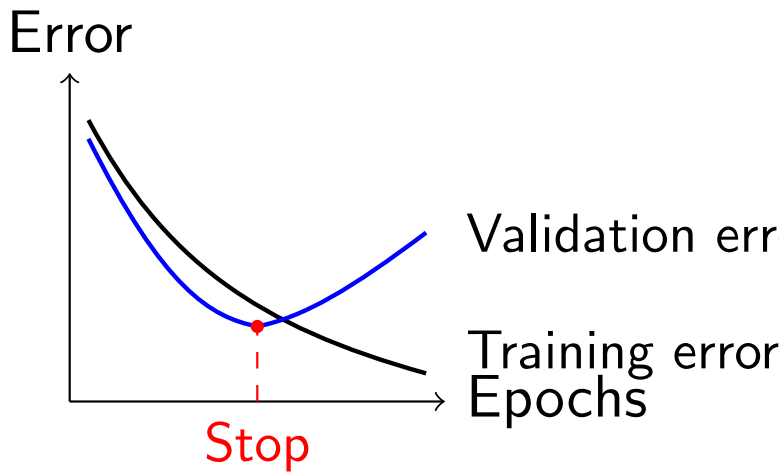

Figure: Early-stopping based on validation error

Model Selection with Cross-Validation

Mathematical Formulation

The cross-validation choice is:

$$\mathcal{F}_{cv} = \min_{k=1,2,\ldots,v} \{e_v''(\mathcal{F}_k)\}$$

where:

- v corresponds to $W_v \leq W_{\max}((1-r)N)$
- $e_v''(\mathcal{F}_k)$ is the classification error for hypothesis $\mathcal{F}_k$ on validation subset
- r determines the split between estimation and validation subsets

Optimal Split Parameter

Based on Kearns (1996), several properties of the optimum r:

- When target function complexity is small relative to sample size N: performance relatively insensitive to r
- As target function becomes more complex: choice of r has more

Nested Function Classes

Model Architecture

Consider nested structure of Boolean function classes:

$$\mathcal{F}_1 \subset \mathcal{F}_2 \subset \cdots \subset \mathcal{F}_n$$

$$\mathcal{F}_k = \{F_k\} = \{F(\mathbf{x}, \mathbf{w}); \mathbf{w} \in \mathcal{W}_k\}, \quad k = 1, 2, \ldots, n$$

- Each class $\mathcal{F}_k$ encompasses multilayer perceptrons with similar architecture
- Weight vectors $\mathbf{w}$ drawn from multidimensional weight space $\mathcal{W}_k$
- Model selection: choose perceptron with best value of $\mathbf{w}$ (number of free parameters)

Generalization Error

For desired response $d \in \{0, 1\}$:

$$\varepsilon_g(F) = P(F(\mathbf{x}) \neq d) \quad \text{for } \mathbf{x} \in \mathcal{X}$$

Practical Considerations

Advantages of Cross-Validation

- Provides robust estimate of model performance
- Helps prevent overfitting
- Useful for comparing different architectures
- Works well with limited data (leave-one-out)

Disadvantages

- Computationally expensive (especially K-fold with large K)
- May require training the model K times
- Validation error may exhibit local minima in practice

Best Practices

- Use $r = 0.2$ (80% training, 20% validation) as default
- Consider computational cost vs. accuracy trade-off
- Always use separate test set for final performance evaluation

Summary

Key Takeaways

- Cross-validation is essential for model selection and preventing overfitting
- Multiple variants available: holdout, K-fold, leave-one-out
- Early stopping uses validation error to determine optimal training duration
- Mathematical framework provides principled approach to model selection
- Computational cost must be balanced against validation accuracy

Remember

The goal is to select a model that generalizes well to unseen data, not one that simply minimizes training error.

5.6 COMPLEXITY REGULARIZATION AND NETWORK PRUNING

The Bias-Variance Dilemma

- When designing multilayer perceptrons, we build **nonlinear models** of physical phenomena
- Challenge: Balance between **reliability** of training data and **goodness** of the model
- This is the classic **bias-variance dilemma** from statistical learning theory
- Need appropriate trade-off for optimal generalization performance

Key Insight

In supervised learning, we must minimize **total risk** rather than just training error

Total Risk Formulation

The total risk is expressed as a function of parameter vector $\mathbf{w}$:

$$R(\mathbf{w}) = \mathcal{E}_{av}(\mathbf{w}) + \lambda \mathcal{E}_c(\mathbf{w}) \tag{1}$$

Where:

- $\mathcal{E}_{av}(\mathbf{w})$ is the **standard performance metric**
- $\mathcal{E}_c(\mathbf{w})$ is the **complexity penalty**
- λ is the **regularization parameter**

Performance Metric

Typically defined as mean-square error evaluated over output neurons for all training examples on epoch-by-epoch basis

Understanding the Regularization Parameter

The regularization parameter λ controls the trade-off:

When $\lambda \to 0$:

- Back-propagation learning is unconstrained
- Network determined entirely by training examples
- Risk of overfitting

When $\lambda \to \infty$:

- Complexity penalty dominates
- Network specification constrained
- Training examples become "unreliable"

Practical Applications

In practice, λ is assigned a value between these limiting cases based on regularization theory

Weight-Decay Regularization

A simplified yet effective form of complexity regularization:

$$\mathcal{E}_c(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_{i \in \mathcal{C}_{total}} w_i^2 \tag{2}$$

Where $\mathcal{C}_{total}$ refers to all synaptic weights in the network.

Mechanism

- Forces some synaptic weights toward zero
- Allows other weights to retain relatively large values
- Weights grouped into two categories based on influence

Weight Categories

1. **Significant weights**: Have substantial influence on network performance
2. **Excess weights**: Have little or no influence on network performance

Problem with Excess Weights

Without complexity regularization, excess weights can:

- Take completely arbitrary values
- Cause network to overfit training data
- Lead to poor generalization performance

Solution

Weight-decay encourages excess weights to assume values close to zero, thereby improving generalization

Optimal Brain Surgeon (OBS)

Goal: Use second-order derivatives of error surface for analytical trade-off between network complexity and training-error performance

Key Idea

Construct local model of error surface to predict effect of perturbations in synaptic weights

Starting Point: Taylor series approximation around operating point

$$\mathcal{E}_{av}(\mathbf{w} + \Delta\mathbf{w}) = \mathcal{E}_{av}(\mathbf{w}) + \mathbf{g}^T(\mathbf{w})\Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} + O(\|\Delta\mathbf{w}\|^3) \quad (3)$$

Where $\mathbf{g}(\mathbf{w})$ is the gradient and $\mathbf{H}$ is the Hessian matrix

OBS Approximations

Two key approximations for practical implementation:

1. Extremal Approximation

Parameters deleted only after training convergence (network fully trained)

- Gradient $\mathbf{g}$ can be set to zero
- Eliminates linear term in Taylor expansion

2. Quadratic Approximation

Error surface around local/global minimum is "nearly quadratic"

- Higher-order terms can be neglected
- Simplifies analysis significantly

Simplified OBS Formulation

Under the two approximations:

$$\Delta\mathcal{E}_{av} = \mathcal{E}(\mathbf{w} + \Delta\mathbf{w}) - \mathcal{E}(\mathbf{w}) = \frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} \tag{4}$$

OBS Objective: Minimize the quadratic form $\frac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w}$ subject to:

$$\mathbf{1}_i^T\Delta\mathbf{w} + w_i = 0 \tag{5}$$

Where $\mathbf{1}_i$ is unit vector with i-th element equal to unity, others zero

Lagrangian Formulation

Constrained optimization solved using Lagrangian:

$$S = \frac{1}{2}\Delta \mathbf{w}^T \mathbf{H} \Delta \mathbf{w} - \lambda(\mathbf{1}_i^T \Delta \mathbf{w} + w_i) \tag{6}$$

Where λ is the Lagrange multiplier.

Solution: Taking derivative with respect to $\Delta \mathbf{w}$ and using matrix inversion:

$$\Delta \mathbf{w} = -\frac{w_i}{[\mathbf{H}^{-1}]_{i,i}} \mathbf{H}^{-1} \mathbf{1}_i \tag{7}$$

$$S_i = \frac{w_i^2}{2[\mathbf{H}^{-1}]_{i,i}} \tag{8}$$

Saliency Measure

The **saliency** S_i represents increase in mean-square error from deleting weight w_i:

$$S_i = \frac{w_i^2}{2[\mathbf{H}^{-1}]_{i,i}} \tag{9}$$

Key Insights

- Saliency proportional to w_i^2 (larger weights more important)
- Inversely proportional to diagonal elements of $\mathbf{H}^{-1}$
- Small $[\mathbf{H}^{-1}]_{i,i}$ means even small weights can significantly affect error

OBS Procedure

Select weight with **smallest saliency** for deletion, then update remaining weights according to optimal changes

Cost Function for Single Output

For multilayer perceptron with single output neuron:

$$\mathcal{E}_{av}(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(d(n) - o(n))^2 \tag{10}$$

Where:

- $o(n)$ is actual network output for n-th example
- $d(n)$ is corresponding desired response
- N is total number of training examples
- $o(n) = F(\mathbf{w}, \mathbf{x})$ where F is input-output mapping function

Hessian Derivatives

First derivative of $\mathcal{E}_{av}$ with respect to $\mathbf{w}$:

$$\frac{\partial \mathcal{E}_{av}}{\partial \mathbf{w}} = -\frac{1}{N}\sum_{n=1}^{N}\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}(d(n)-o(n)) \tag{11}$$

Second derivative (Hessian):

$$\mathbf{H}(N) = \frac{\partial^2 \mathcal{E}_{av}}{\partial \mathbf{w}^2} \tag{12}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\left\{\left(\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\right)\left(\frac{\partial F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}}\right)^{T}\right. \tag{13}$$

$$\left. -\frac{\partial^2 F(\mathbf{w},\mathbf{x}(n))}{\partial \mathbf{w}^2}(d(n)-o(n))\right\} \tag{14}$$

Hessian Approximation

Under assumption of fully trained network ($d(n) \approx o(n)$):

$$\mathbf{H}(N) \approx \frac{1}{N} \sum_{n=1}^{N} \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right) \left(\frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \right)^T \tag{15}$$

Define the W-by-1 vector:

$$\boldsymbol{\xi}(n) = \frac{1}{\sqrt{N}} \frac{\partial F(\mathbf{w}, \mathbf{x}(n))}{\partial \mathbf{w}} \tag{16}$$

Then Hessian can be written recursively:

$$\mathbf{H}(n) = \sum_{k=1}^{n} \boldsymbol{\xi}(k) \boldsymbol{\xi}^T(k) \tag{17}$$

$$= \mathbf{H}(n-1) + \boldsymbol{\xi}(n) \boldsymbol{\xi}^T(n), \quad n = 1, 2, \ldots, N \tag{18}$$

Recursive Hessian Inverse

Using matrix inversion lemma (Woodbury's equality):
For matrices $\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}\mathbf{C}^T$:

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}(\mathbf{D} + \mathbf{C}^T\mathbf{B}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{B} \tag{19}$$

For our problem: $\mathbf{A} = \mathbf{H}(n)$, $\mathbf{B}^{-1} = \mathbf{H}(n-1)$, $\mathbf{C} = \boldsymbol{\xi}(n)$, $\mathbf{D} = 1$

$$\mathbf{H}^{-1}(n) = \mathbf{H}^{-1}(n-1) - \frac{\mathbf{H}^{-1}(n-1)\boldsymbol{\xi}(n)\boldsymbol{\xi}^T(n)\mathbf{H}^{-1}(n-1)}{1 + \boldsymbol{\xi}^T(n)\mathbf{H}^{-1}(n-1)\boldsymbol{\xi}(n)} \tag{20}$$

Initialization

$\mathbf{H}^{-1}(0) = \delta^{-1}\mathbf{I}$ where δ is small positive number, $\mathbf{I}$ is identity matrix

OBS Performance Results

Benchmark Comparisons:

- OBS procedure resulted in **smaller networks** than weight-decay
- Applied to NETtalk multilayer perceptron:
  - Original: Single hidden layer, over 18,000 weights
  - After OBS: Pruned to mere 1,560 weights
  - **Dramatic reduction**: 91.3% weight elimination

NETtalk Background

NETtalk (Sejnowski and Rosenberg, 1987) was landmark neural network for text-to-speech conversion, described in Section 4.18

Key Advantage

OBS provides principled approach to network pruning while maintaining performance

Algorithm Summary: Optimal Brain Surgeon

Algorithm 1 Optimal Brain Surgeon (OBS)

Objective

Train the given multilayer perceptron to minimum mean-square error while optimizing network structure.

1. Train the multilayer perceptron to minimum mean-square error.
2. Compute the sensitivity vector: $\xi(n) = \frac{1}{\sqrt{N}} \frac{\partial F(w, x(n))}{\partial w}$
3. Use recursion to compute the inverse Hessian H^{-1}.
4. Find the weight i that corresponds to the smallest saliency: $S_i = \frac{w_i^2}{2[H^{-1}]_{i,i}}$
5. Update all synaptic weights: $\Delta w = -\frac{w_i}{[H^{-1}]_{i,i}} H^{-1} I_i$
6. Stop when no more weights can be deleted without large error increase.

Key Takeaways

1. **Complexity Regularization**: Essential for managing bias-variance trade-off in neural networks
2. **Weight-Decay**: Simple yet effective regularization technique using L_2 norm penalty
3. **Optimal Brain Surgeon**: Sophisticated pruning method using second-order information
4. **Hessian Matrix**: Central role in understanding network sensitivity and optimal weight removal
5. **Practical Impact**: Dramatic network size reduction (¿90%) while maintaining performance

Future Directions

These techniques form foundation for modern neural network compression and regularization methods

Thank You!

Questions and Discussion

Topics for further exploration:

- Modern regularization techniques (Dropout, Batch Normalization)
- Network architecture search and pruning
- Relationship to information theory and generalization bounds