Unit 5 Multilayer Perceptron

Heuristics for Making the Back-Propagation Algorithm Perform Better

Kiran Bagale

July 2025

Introduction

Key Insight

The design of a neural network using the back-propagation algorithm is often said to be more of an **art** than a **science**.

- Many design factors come from personal experience
- Nevertheless, systematic methods exist to significantly improve performance
- We'll explore 8 key heuristics for better back-propagation

1. Stochastic versus Batch Update

Stochastic Mode

- Pattern-by-pattern updating
- Computationally faster
- Especially effective for large, redundant datasets

Batch Mode

- Updates after entire dataset
- Computationally slower
- Problems with redundant data
- Jacobian estimation issues

Recommendation

Use **stochastic mode** for large and highly redundant training datasets.

2. Maximizing Information Content

General Rule

Every training example should be chosen to maximize information content for the task at hand.

Two Ways to Realize This:

- Use examples with largest training error
- Use examples radically different from previous ones

Motivation:

- Search more of the weight space
- Avoid redundant learning
- Improve convergence speed

Common Technique

Randomize (shuffle) the order of examples from epoch to epoch to ensure successive examples rarely belong to the same class.

3. Activation Function Choice

Recommendation

Use a sigmoid activation function that is an **odd function** of its argument:

$$\varphi(-v) = -\varphi(v)$$

Hyperbolic Tangent Function:

$$\varphi(v) = a \tanh(bv)$$

Recommended Parameters:

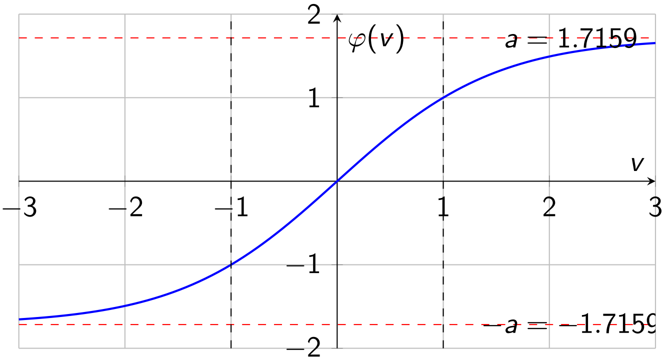
$$a = 1.7159 \quad (1)$$

$$b = \frac{2}{3} \quad (2)$$

Properties:

- $\varphi(1) = 1$
- $\varphi(-1) = -1$
- $\varphi(0) = ab = 1.1424$
- Second derivative maximum at $v = 1$

Hyperbolic Tangent Function Graph



Target values: $+1$ and -1 (offset from limiting values)

4. Target Values

Important Rule

Target values should be chosen within the range of the sigmoid activation function, but **offset** from limiting values.

For the hyperbolic tangent function:

$$d_j = a - \varepsilon \quad \text{(for positive limiting value)} \tag{3}$$

$$d_j = -a + \varepsilon \quad \text{(for negative limiting value)} \tag{4}$$

where ε is an appropriate positive constant.

Example

For $a = 1.7159$, setting $\varepsilon = 0.7159$ gives convenient target values of ± 1.

Why?

Prevents the back-propagation algorithm from driving neurons into saturation, which slows learning.

5. Normalizing the Inputs

Preprocessing Rule

Each input variable should be **preprocessed** so that its mean value is close to zero, or small compared to its standard deviation.

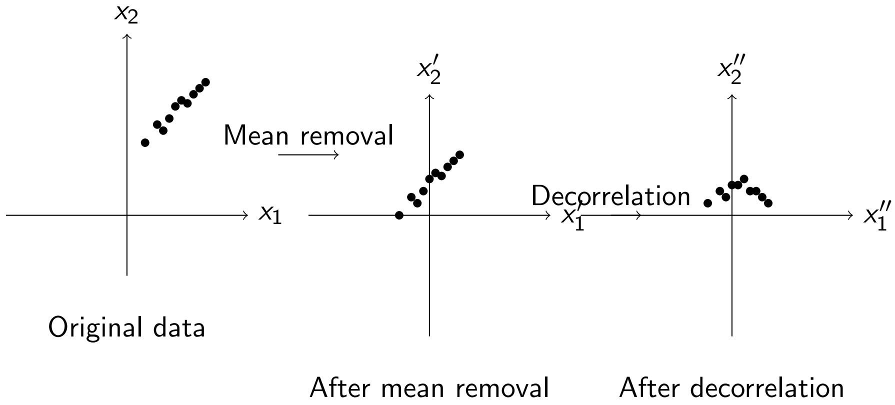
Why is this important?

- When inputs are consistently positive, synaptic weights can only increase or decrease together
- Weight vector changes direction by "zigzagging" through error surface
- This is typically slow and should be avoided

Two normalization measures:

1. Input variables should be uncorrelated (use PCA)
2. Decorrelated variables should be scaled so their covariances are approximately equal

Normalization Process Illustration



Result: Effective gain of sigmoid function over useful range is approximately unity.

6. Initialization Strategy

Avoid These

- Large initial values → saturation
- Small initial values → saddle points

Key Insight

The origin is a **saddle point** for sigmoid functions - negative curvature across saddle, positive along saddle.

Mathematical Framework: For induced local field of neuron j:

$$v_j = \sum_{i=1}^{m} w_{ji} y_i$$

Assumptions:

- Inputs: $\mu_y = 0$, $\sigma_y^2 = 1$, uncorrelated
- Weights: uniformly distributed, $\mu_w = 0$, variance σ_w^2

Result: Variance of induced field is $\sigma_v^2 = m\sigma_w^2$

Optimal Weight Initialization

Objective

Standard deviation of induced local field should equal 1 to operate in transition area between linear and saturated parts.

Setting $\sigma_v = 1$ in the equation $\sigma_v^2 = m\sigma_w^2$:

$$\sigma_w = m^{-1/2}$$

Recommendation

Initialize synaptic weights from a uniform distribution with:

- Mean: **zero**
- Variance: **reciprocal of number of synaptic connections**

This represents *learning from hints* - using prior knowledge about the activation function to improve initialization.

7. Learning from Hints

Concept

Learning from a sample of training examples deals with an unknown input-output mapping function $f(\cdot)$.

Key Idea: The learning process can be enhanced by including **learning from hints**.

Types of Hints:

- Invariance properties
- Symmetries
- Any other prior knowledge about $f(\cdot)$

Example

The weight initialization strategy ($\sigma_w = m^{-1/2}$) is an example of learning from hints about the activation function properties.

Benefits:

- Accelerates search for approximate realization
- Improves quality of final estimate

8. Learning Rates

Ideal Goal

All neurons in the multilayer perceptron should learn at the same rate.

Problem: Last layers usually have larger local gradients than front layers.
Solution: Assign smaller learning-rate parameter η to last layers than to front layers.

LeCun's Recommendation (1993)

For a given neuron, the learning rate should be **inversely proportional to the square root of synaptic connections** made to that neuron.

Reasoning:

- Neurons with many inputs need smaller learning rates
- Maintains similar learning time for all neurons
- Balances gradient magnitudes across layers

Summary of 8 Heuristics

1. **Stochastic vs Batch:** Use stochastic for large, redundant datasets
2. **Information Content:** Choose examples with max error or diversity
3. **Activation Function:** Use odd sigmoid (hyperbolic tangent)
4. **Target Values:** Offset from activation function limits
5. **Input Normalization:** Zero mean, decorrelated, equal covariance
6. **Weight Initialization:** Variance $= m^{-1/2}$ (m = connections)
7. **Learning from Hints:** Incorporate prior knowledge
8. **Learning Rates:** Inversely proportional to $\sqrt{\text{connections}}$

Key Takeaway

While neural network design has artistic elements, these systematic heuristics can significantly improve back-propagation performance.

5.6 Back Propagation and Differentiation

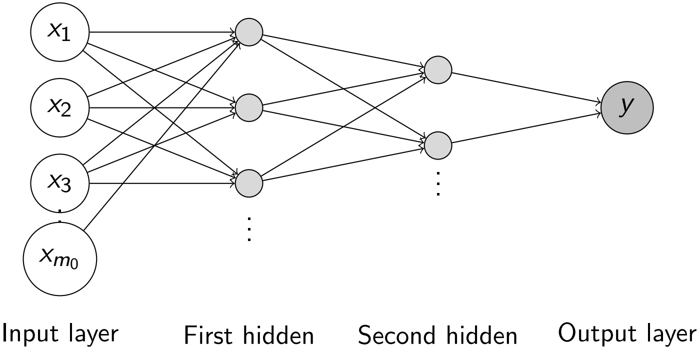
Core Concept

Back propagation is a specific technique for implementing **gradient descent** in weight space for a multilayer perceptron.

Basic Idea: Efficiently compute partial derivatives of an approximating function $F(\mathbf{w}, \mathbf{x})$ with respect to all elements of the adjustable weight vector $\mathbf{w}$.

- For example, for $l = 2$ (i.e., a single hidden layer and a linear output layer), we have

$$F(\mathbf{w}, \mathbf{x}) = \sum_{j=0}^{m_1} w_{oj} \, \varphi \left(\sum_{i=0}^{m_0} w_{ji} x_i \right)$$



Network Function

For a multilayer perceptron with architecture $\mathcal{A}$ and weight vector $\mathbf{w}$:
$F(\mathbf{w}, \mathbf{x}) = \varphi(\mathcal{A}_1^{(3)})$

Sensitivity Analysis

Partial Derivatives for 3-layer network:

$$\frac{\partial F(\mathbf{w},\mathbf{x})}{\partial w_{jk}^{(3)}} = \varphi'(\mathcal{A}_1^{(3)})\varphi(\mathcal{A}_k^{(2)}) \quad (5)$$

$$\frac{\partial F(\mathbf{w},\mathbf{x})}{\partial w_{kj}^{(2)}} = \varphi'(\mathcal{A}_1^{(3)})\varphi'(\mathcal{A}_k^{(2)})\varphi(\mathcal{A}_j^{(1)})w_{1k}^{(3)} \quad (6)$$

$$\frac{\partial F(\mathbf{w},\mathbf{x})}{\partial w_{ji}^{(1)}} = \varphi'(\mathcal{A}_1^{(3)})\varphi'(\mathcal{A}_j^{(1)})x_i \left[\sum_k w_{1k}^{(3)}\varphi'(\mathcal{A}_k^{(2)})w_{kj}^{(2)}\right] \quad (7)$$

Sensitivity Definition

The sensitivity of $F(\mathbf{w},\mathbf{x})$ with respect to weight ω is: $S_\omega^F = \frac{\partial F/F}{\partial \omega/\omega}$

This forms the basis for the "sensitivity graph" in signal-flow analysis.

The Jacobian Matrix

Definition

Let W = total number of free parameters (weights and biases)
Let N = total number of training examples

Using back propagation, we compute W partial derivatives of approximating function $F[\mathbf{w}, \mathbf{x}(n)]$ for each example $\mathbf{x}(n)$.
Result: An $N \times W$ matrix of partial derivatives called the **Jacobian J**.

Properties

- Each row: one training example
- Each column: one weight parameter
- Rank $= \min(N, W)$

Rank Deficiency

$\mathbf{J}$ is rank deficient if rank $< \min(N, W)$

- Causes partial information loss
- Leads to long training times

Experimental Evidence: Many neural network training problems are numerically ill-conditioned due to almost rank-deficient Jacobians.

5.7 The Hessian and Its Role in On-line Learning

Definition

The **Hessian matrix H** of cost function $\mathcal{E}_{av}(\mathbf{w})$ is the second derivative with respect to weight vector $\mathbf{w}$:

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

Three Key Roles of the Hessian

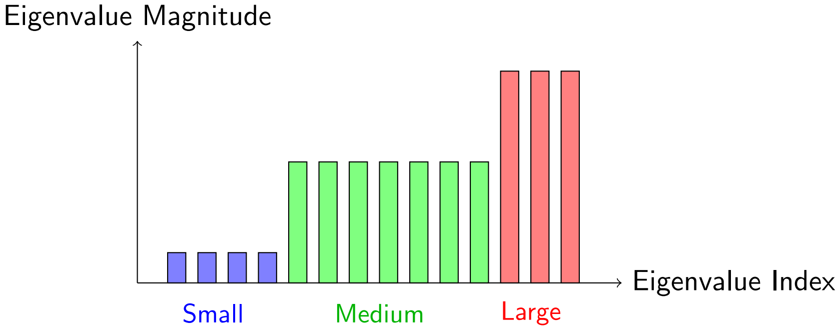
1. **Learning Dynamics:** Eigenvalues profoundly influence back-propagation learning dynamics
2. **Network Pruning:** Inverse of Hessian provides basis for deleting insignificant synaptic weights
3. **Second-order Optimization:** Basic to formulation of second-order methods as alternatives to back-propagation

Focus: We concentrate on the influence of Hessian eigenstructure on convergence properties.

Eigenvalue Composition of the Hessian

Typical Eigenvalue Distribution

For a multilayer perceptron trained with back-propagation, the Hessian typically has:



- Small number of small eigenvalues
- Large number of medium-sized eigenvalues
- Small number of large eigenvalues

Consequence: There is a **wide spread** in the eigenvalues of the Hessian, leading to ill-conditioning.

Factors Affecting Eigenvalue Composition

The eigenvalues of the Hessian are affected by:

1. **Nonzero-mean signals:**
   - Nonzero-mean input signals
   - Nonzero-mean induced neural output signals
2. **Signal correlations:**
   - Correlations between input signal vector elements
   - Correlations between induced neural output signals
3. **Derivative variations:**
   - Wide variations in second-order derivatives of cost function
   - Derivatives often smaller in lower layers
   - First hidden layer learns slowly, last layers learn quickly

Key Insight

These factors create a wide spread in Hessian eigenvalues, affecting convergence dynamics.

Avoidance of Nonzero-mean Inputs

Problem Statement

Learning time of back-propagation is sensitive to the condition number $\lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are the largest and smallest nonzero eigenvalues of the Hessian.

Key Observation: For inputs with nonzero mean, the ratio $\lambda_{max}/\lambda_{min}$ is larger than for zero-mean inputs.

Solutions:

1. Input layer: Easy to remove mean from each element of $\mathbf{x}$
2. Hidden/Output layers: Use hyperbolic tangent function (odd-symmetric)

Benefit

Hyperbolic tangent allows outputs to assume both positive and negative values in $[-1, 1]$, making mean likely to be zero.

Result: Back-propagation with odd-symmetric activation functions can yield faster convergence than with nonsymmetric functions.

Learning Curve Analysis: Asymptotic Behavior of On-line Learning

Components of Learning Curve

The learning curve consists of three terms:

1. **Minimal loss:** Determined by optimal parameter $\mathbf{w}^*$ (local/global minimum)
2. **Additional loss:** Caused by fluctuations in weight-vector estimator $w(n)$ around the mean: $\lim_{n \to \infty} \mathbb{E}[\hat{\mathbf{w}}(n)] = \mathbf{w}^*$
3. **Time-dependent term:** Effect of decreasing speed of error convergence on algorithmic performance

Learning Rate Trade-off

- Large η: Fast convergence, large fluctuations around minimum
- Small η: Small fluctuations, slow convergence

5.8 Optimal Annealing and Adaptive Control of the Learning Rate

Importance of online learning

- **Simplicity**: Minimal memory requirements
  - Only stores previous weight vector estimate
  - Memory efficient compared to batch methods
- **Adaptivity**: Built-in tracking capability
  - Each example $\{x, d\}$ used only once
  - Learning rate plays crucial role
  - Can track statistical variations in environment
- **Performance**: Can match batch learning asymptotically
  - Amari (1967), Opper (1996) theoretical results
  - Optimally annealed online learning $\approx$ batch learning

Optimal Annealing of the Learning Rate: Cost Function and Gradient

Instantaneous Cost Function:

$$\mathcal{E}(x(n), d(n); w) = \frac{1}{2}\|d(n) - F(x(n); w)\|^2 \tag{8}$$

Mean-Square Error (Expected Risk):

$$J(w) = \mathbb{E}_{x,d}[\mathcal{E}(x, d; w)] \tag{9}$$

Optimal Parameter Vector:

$$w^* = \arg\min_{w}[J(w)] \tag{10}$$

Instantaneous Gradient:

$$g(x(n), d(n); w) = \frac{\partial}{\partial w}\mathcal{E}(x(n), d(n); w) \tag{11}$$

$$= -(d(n) - F(x(n); w))F'(x(n); w) \tag{12}$$

Online Learning Algorithm Formulation

Weight Update Rule:

$$\hat{w}(n+1) = \hat{w}(n) - \eta(n)g(x(n+1), d(n+1); \hat{w}(n)) \tag{13}$$

Equivalent Form:

$$\hat{w}(n+1) = \hat{w}(n) + \eta(n)[d(n+1) - F(x(n+1); \hat{w}(n))]F'(x(n+1); \hat{w}(n)) \tag{14}$$

Components

- $\hat{w}(n)$: Old weight estimate
- $\eta(n)$: Learning rate parameter
- $d(n+1) - F(x(n+1); \hat{w}(n))$: Error signal
- $F'(x(n+1); \hat{w}(n))$: Partial derivative of network function

Continuous-Time Analysis

Ensemble-Averaged Dynamics:

$$\frac{d}{dt}\hat{w}(t) = -\eta(t)\mathbb{E}_{x,d}[g(x(t), d(t); \hat{w}(t))] \tag{15}$$

Approximation (Murata, 1998):

$$\mathbb{E}_{x,d}[g(x, d; \hat{w}(t))] \approx -K^*(w^* - \hat{w}(t)) \tag{16}$$

where the ensemble-averaged matrix K^* is:

$$K^* = \mathbb{E}_{x,d}\left[\frac{\partial}{\partial w} g(x, d; w)\right] \tag{17}$$

$$= \mathbb{E}_{x,d}\left[\frac{\partial^2}{\partial w^2} \mathcal{E}(x, d; w)\right] \tag{18}$$

Simplified Dynamics:

$$\frac{d}{dt}\hat{w}(t) \approx -\eta(t)K^*(w^* - \hat{w}(t)) \tag{19}$$

Let q be an eigenvector of K^*:

$$K^* q = \lambda q \tag{20}$$

Projection Function:

$$\xi(t) = \mathbb{E}_{x,d}[q^T g(x, d; \hat{w}(t))] \tag{21}$$

Approximation:

$$\xi(t) \approx -q^T K^*(w^* - \hat{w}(t)) \tag{22}$$
$$= -\lambda q^T (w^* - \hat{w}(t)) \tag{23}$$

Differential Equation:

$$\frac{d}{dt}\xi(t) = -\lambda \eta(t) \xi(t) \tag{24}$$

Solution:

$$\xi(t) = c \exp\left(-\lambda \int \eta(t) dt\right) \tag{25}$$

Darken-Moody Annealing Schedule

Time-Dependent Learning Rate:

$$\eta(t) = \frac{\tau}{t + \tau}\eta_0 \tag{26}$$

where τ and η_0 are positive tuning parameters.
Resulting Function:

$$\xi(t) = c(t + \tau)^{-\lambda\tau\eta_0} \tag{27}$$

Convergence Condition: For $\xi(t) \to 0$ as $t \to \infty$, we need:

$$\lambda\tau\eta_0 > 1$$

This can be satisfied by setting $\eta_0 = \alpha/\lambda$ for positive α.

Optimal Eigenvector Choice

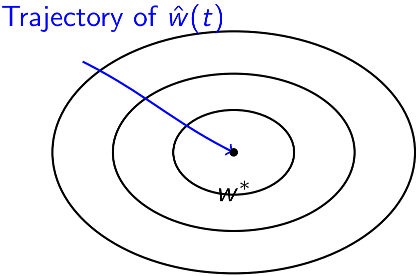
Key Insight: The convergence speed is dominated by the smallest eigenvalue λ_{min} of the Hessian H.

Eigenvector Selection:

$$q = \frac{\mathbb{E}_{x,d}[g(x, d; \hat{w})]}{\|\mathbb{E}_{x,d}[g(x, d; \hat{w})]\|} \tag{28}$$

Distance Measure:

$$\xi(t) = \|\mathbb{E}_{x,d}[g(x, d; \hat{w}(t))]\| \tag{29}$$



Optimal Annealing Results

Key Properties of Optimal Schedule:

1. **Stochastic Approximation Conditions:**

$$\sum_t \eta(t) \to \infty \quad \text{and} \quad \sum_t \eta^2(t) > \infty, \quad \text{as } t \to \infty \tag{30}$$

2. **Asymptotic Convergence:**

$$\xi(t) \to 0 \Rightarrow \hat{w}(t) \to w^* \text{ as } t \to \infty$$

3. **Trajectory Alignment:** The estimator trajectory becomes almost parallel to the eigenvector of K^* with smallest eigenvalue λ_{min}.
4. **Stability Condition:** For fixed learning rate: $\eta_0 < 1/\lambda_{max}$
   For optimal annealing: $\eta_0 < 1/\lambda_{min}$

Murata Adaptive Algorithm

Motivation: Practical limitation of optimal annealing:

- Requires knowledge of time constant n_{switch} a priori
- Environment may be non-stationary
- Need for adaptive control mechanism

Algorithm Objectives:

1. **Automatic adjustment** of learning rate
2. **Generalization** - avoid prescribed cost function requirement

Flow Function Approach:

$$\frac{d}{dt}\hat{w}(t) = -\eta(t)\mathbb{E}_{x,d}[f(x(t), d(t); \hat{w}(t))] \tag{31}$$

where flow f must satisfy: $\mathbb{E}_{x,d}[f(x, d; w^*)] = 0$

Murata's Dynamic System:

$$\frac{d}{dt}\xi(t) = -\lambda\eta(t)\xi(t) \tag{32}$$
$$\frac{d}{dt}\eta(t) = \alpha\eta(t)(\beta\xi(t) - \eta(t)) \tag{33}$$

where $\xi(t) > 0$ always, and $\alpha, \beta > 0$ are constants.
Asymptotic Behavior:

$$\xi(t) = \frac{1}{\beta}\left(\frac{1}{\lambda} - \frac{1}{\alpha}\right)\frac{1}{t}, \quad \alpha > \lambda \tag{34}$$
$$\eta(t) = \frac{c}{t}, \quad c = \lambda^{-1} \tag{35}$$

Key Result: The system exhibits desired annealing $\eta(t) = c/t$ for large t, which is optimal for convergence to w^*.

Discrete-Time Implementation

Murata Adaptive Algorithm:

$$\hat{w}(n+1) = \hat{w}(n) - \eta(n) f(x(n+1), d(n+1); \hat{w}(n)) \tag{36}$$

$$r(n+1) = r(n) + \delta f(x(n+1), d(n+1); \hat{w}(n)), \quad 0 < \delta < 1 \tag{37}$$

$$\eta(n+1) = \eta(n) + \alpha\eta(n)(\beta\|r(n+1)\| - \eta(n)) \tag{38}$$

Key Features:

- Auxiliary vector $r(n)$ accounts for continuous-time function $\xi_X(t)$
- Leakage factor δ controls running average of flow f
- Links discrete and continuous-time formulations

Important Limitation:

$$\lim_{n\to\infty} \hat{w}(n) \neq w^* \tag{39}$$

Unlike optimal annealing, this algorithm is suboptimal but more practical.

Algorithm Comparison

Property	Fixed LR	Optimal Annealing	Murata Adap
Convergence to w^*	No	Yes	No
Requires n_{switch}	N/A	Yes	No
Stability condition	$\eta < 1/\lambda_{max}$	$\eta_0 < 1/\lambda_{min}$	Built-in
Non-stationary env.	Poor	Poor	Good
Implementation	Simple	Moderate	Complex
Memory requirements	Low	Low	Moderate

Practical Considerations:

- **1/n rule**: Good when optimal $\hat{w}^*$ changes slowly
- **Adaptive control**: Essential for non-stationary environments
- **Trade-off**: Optimality vs. adaptability

Key Takeaways

Theoretical Results

- Optimal annealing can achieve batch learning performance asymptotically
- Learning rate schedule: $\eta(n) = \frac{n_{switch}}{n+n_{switch}}\eta_0$
- Convergence trajectory aligns with smallest eigenvalue eigenvector

Practical Implications

- Online learning's importance lies in its adaptability
- Built-in mechanism to track environmental variations
- Adaptive control enables broader applicability

Future Directions

- Combine optimality with adaptability
- Non-stationary environment handling
- Computational efficiency improvements

Questions?

Thank You!

Questions and Discussion