1

Introduction

Course Objectives

This course gives an introduction to basic neural network architectures and learning rules.

Emphasis is placed on the mathematical analysis of these networks, on methods of training them and on their application to practical engineering problems in such areas as pattern recognition, signal processing and control systems.

What Will Not Be Covered

- Review of all architectures and learning rules
- Implementation
  - VLSI
  - Optical
  - Parallel Computers
- Biology
- Psychology

Historical Sketch

- Pre-1940: von Hemholtz, Mach, Pavlov, etc.
  - General theories of learning, vision, conditioning
  - No specific mathematical models of neuron operation
- 1940s: Hebb, McCulloch and Pitts
  - Mechanism for learning in biological neurons
  - Neural-like networks can compute any arithmetic function
- 1950s: Rosenblatt, Widrow and Hoff
  - First practical networks and learning rules
- 1960s: Minsky and Papert
  - Demonstrated limitations of existing neural networks, new learning algorithms are not forthcoming, some research suspended
- 1970s: Amari, Anderson, Fukushima, Grossberg, Kohonen
  - Progress continues, although at a slower pace
- 1980s: Grossberg, Hopfield, Kohonen, Rumelhart, etc.
  - Important new developments cause a resurgence in the field

Applications

- Aerospace
  - High performance aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations, aircraft component fault detectors
- Automotive
  - Automobile automatic guidance systems, warranty activity analyzers
- Banking
  - Check and other document readers, credit application evaluators
- Defense
  - Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification
- Electronics
  - Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling

Applications

- Financial
  - Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction
- Manufacturing
  - Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process systems
- Medical
  - Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency room test advisement

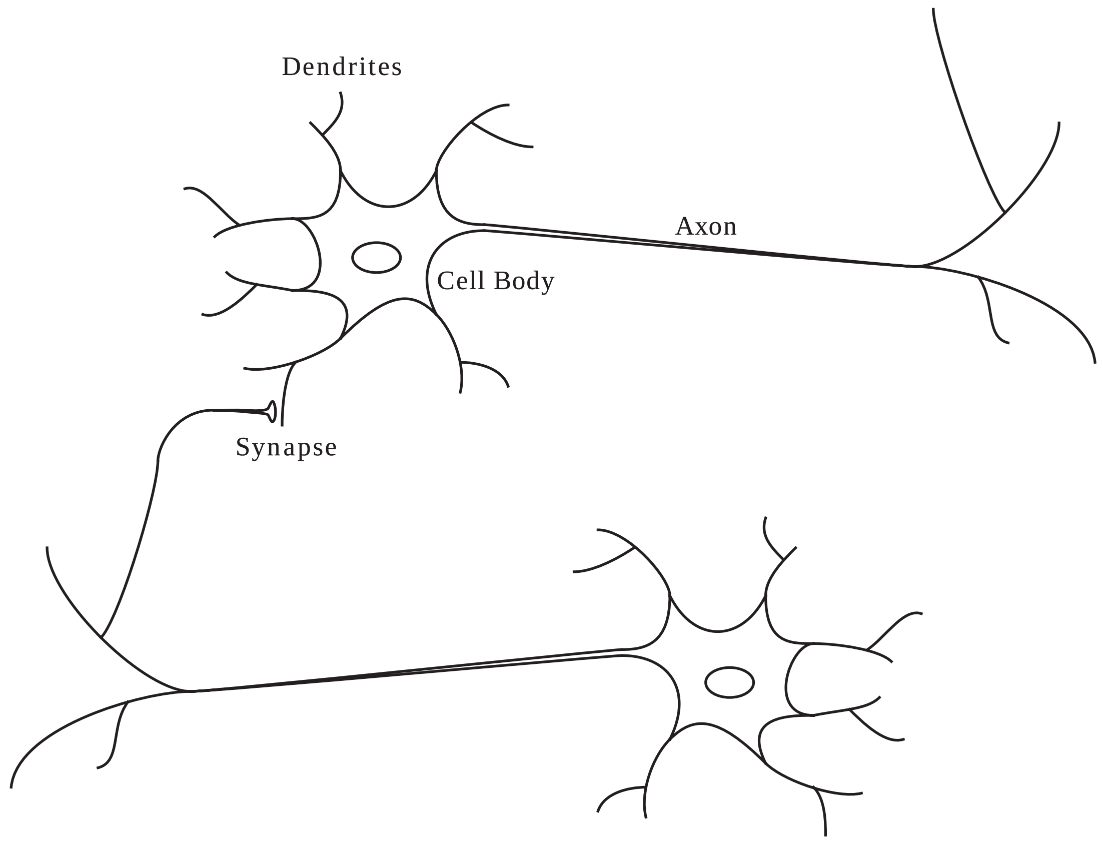
Applications

- Robotics
  - Trajectory control, forklift robot, manipulator controllers, vision systems
- Speech
  - Speech recognition, speech compression, vowel classification, text to speech synthesis
- Securities
  - Market analysis, automatic bond rating, stock trading advisory systems
- Telecommunications
  - Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems
- Transportation
  - Truck brake diagnosis systems, vehicle scheduling, routing systems

1

Biology

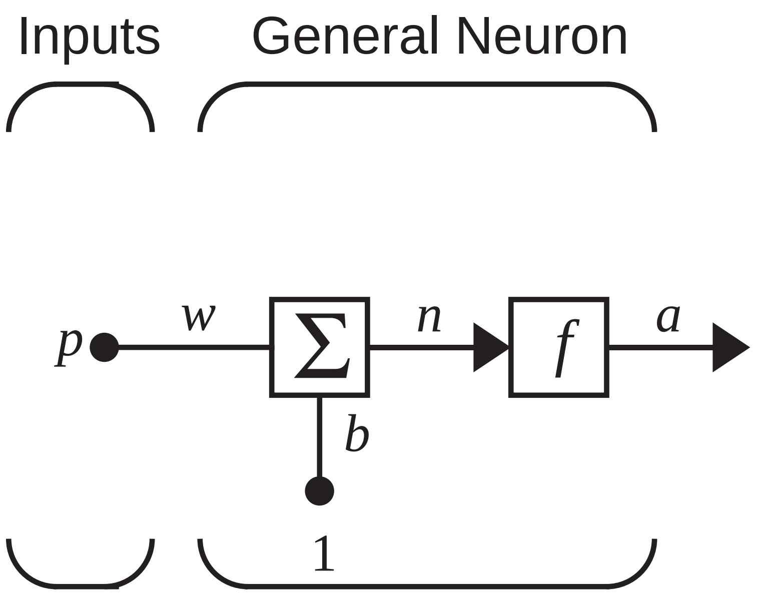
- Neurons respond slowly
  - 10^{-3} s compared to 10^{-9} s for electrical circuits
- The brain uses massively parallel computation
  - $\approx 10^{11}$ neurons in the brain
  - $\approx 10^{4}$ connections per neuron



2

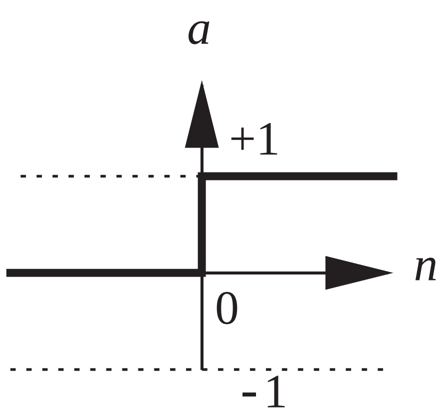
Neuron Model and Network Architectures

Single-Input Neuron



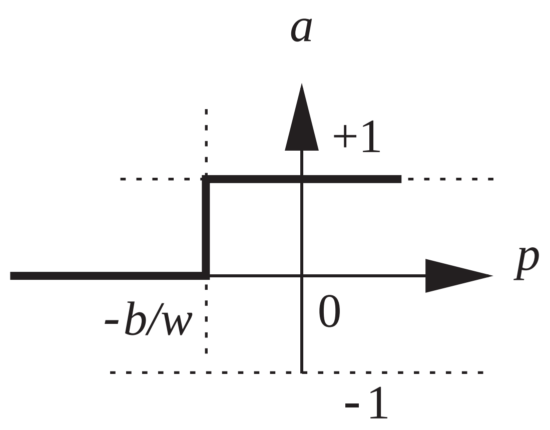
$$a = f(wp + b)$$

Transfer Functions



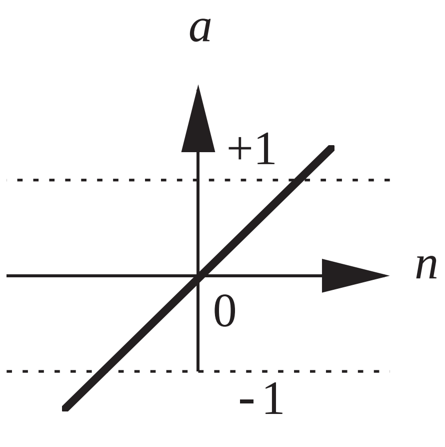
$a = hardlim(n)$

Hard Limit Transfer Function



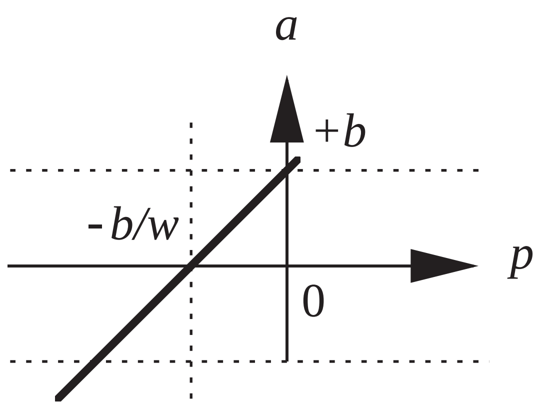
$a = hardlim(wp + b)$

Single-Input *hardlim* Neuron



$a = purelin(n)$

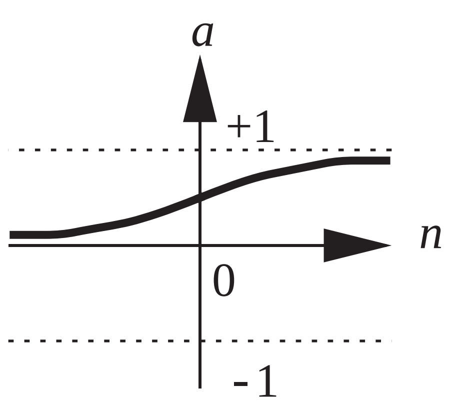
Linear Transfer Function

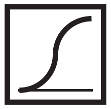


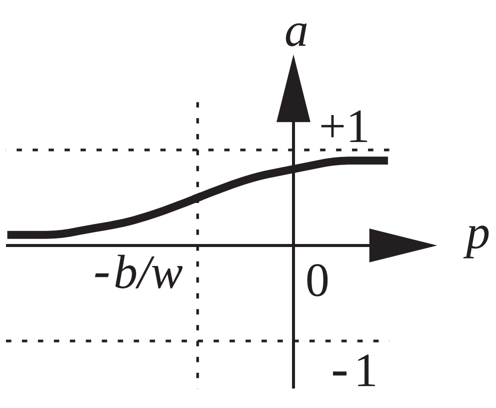
$a = purelin(wp + b)$

Single-Input *purelin* Neuron

Transfer Functions







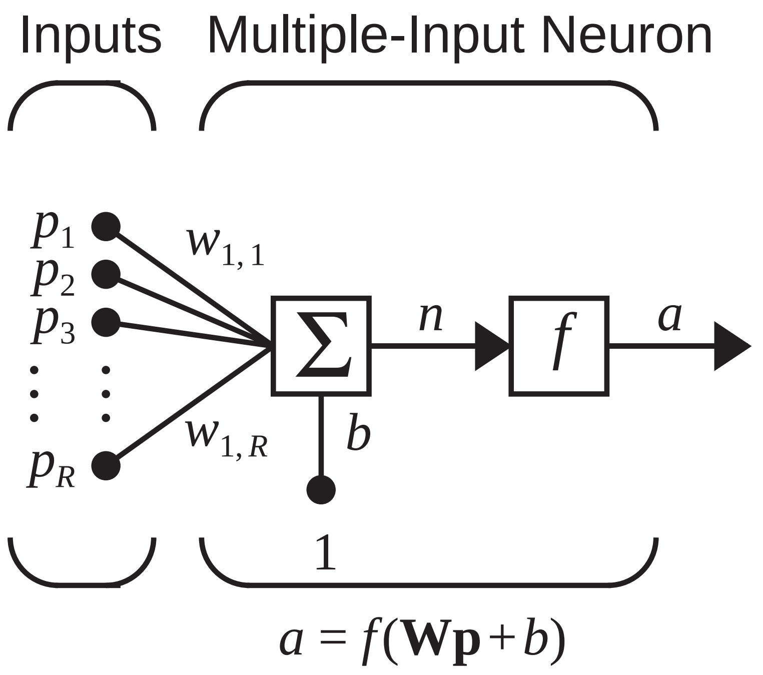
$$a = logsig(n)$$

Log-Sigmoid Transfer Function

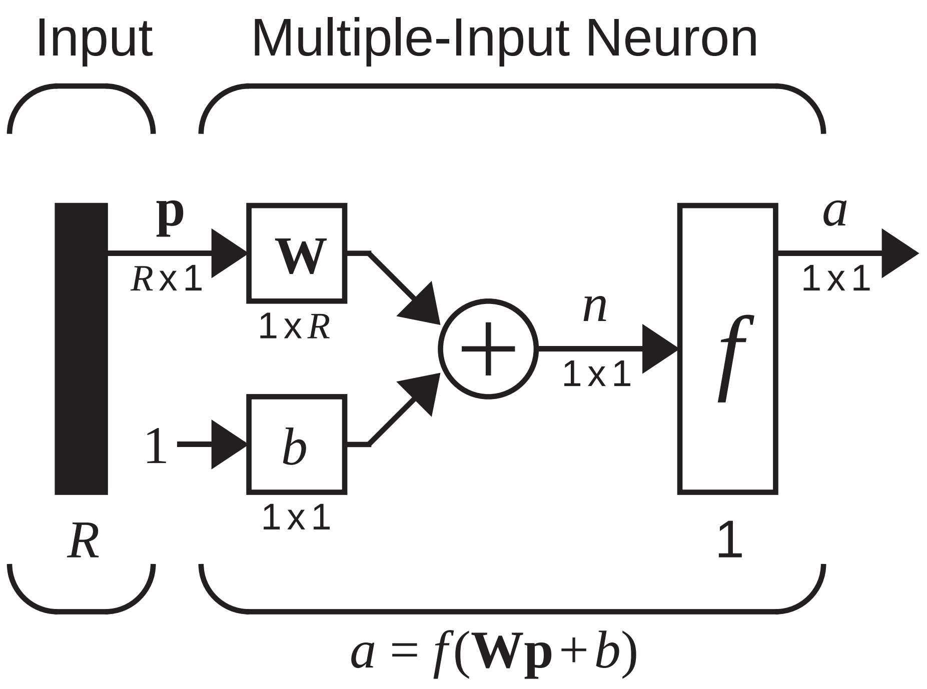
$$a = logsig(wp + b)$$

Single-Input *logsig* Neuron

Multiple-Input Neuron



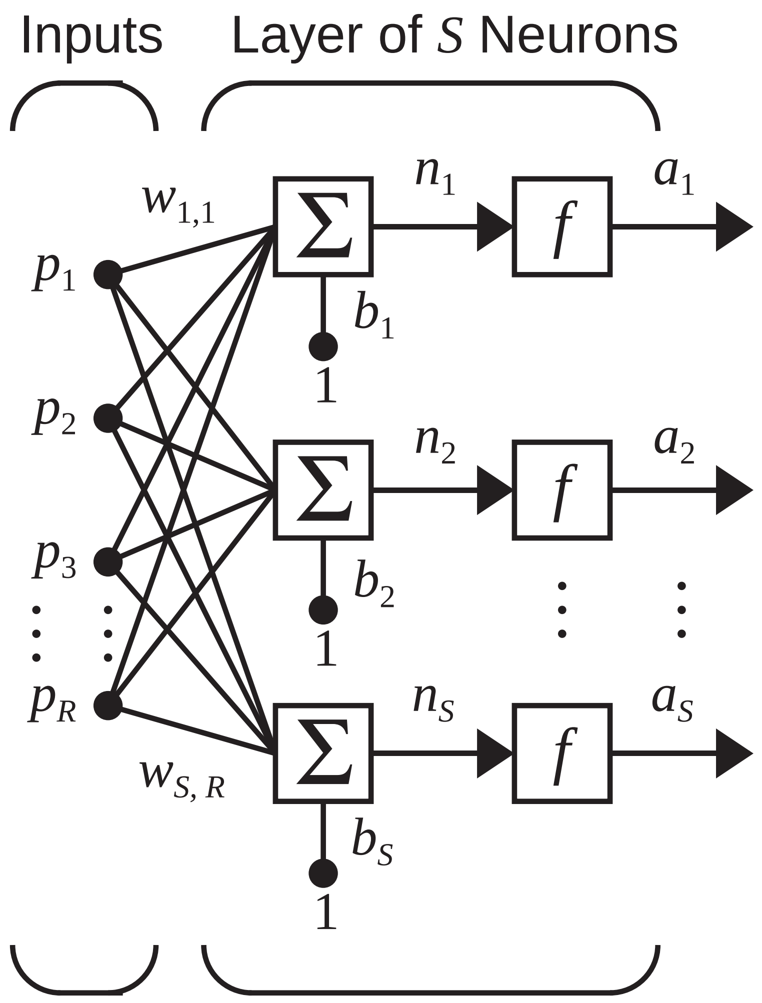
$a = f(\mathbf{Wp} + b)$



$a = f(\mathbf{Wp} + b)$

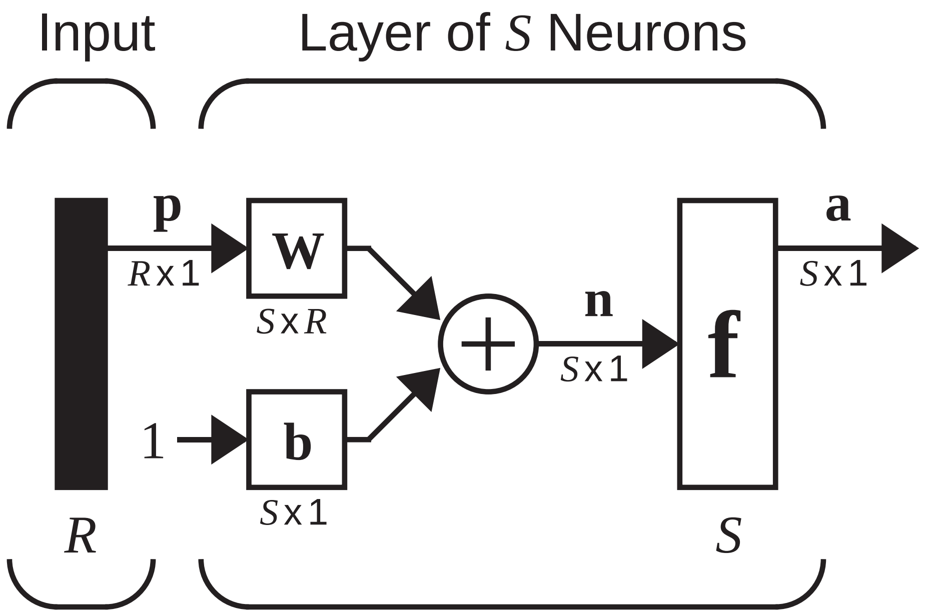
Abreviated Notation

Layer of Neurons



$$\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

Abbreviated Notation

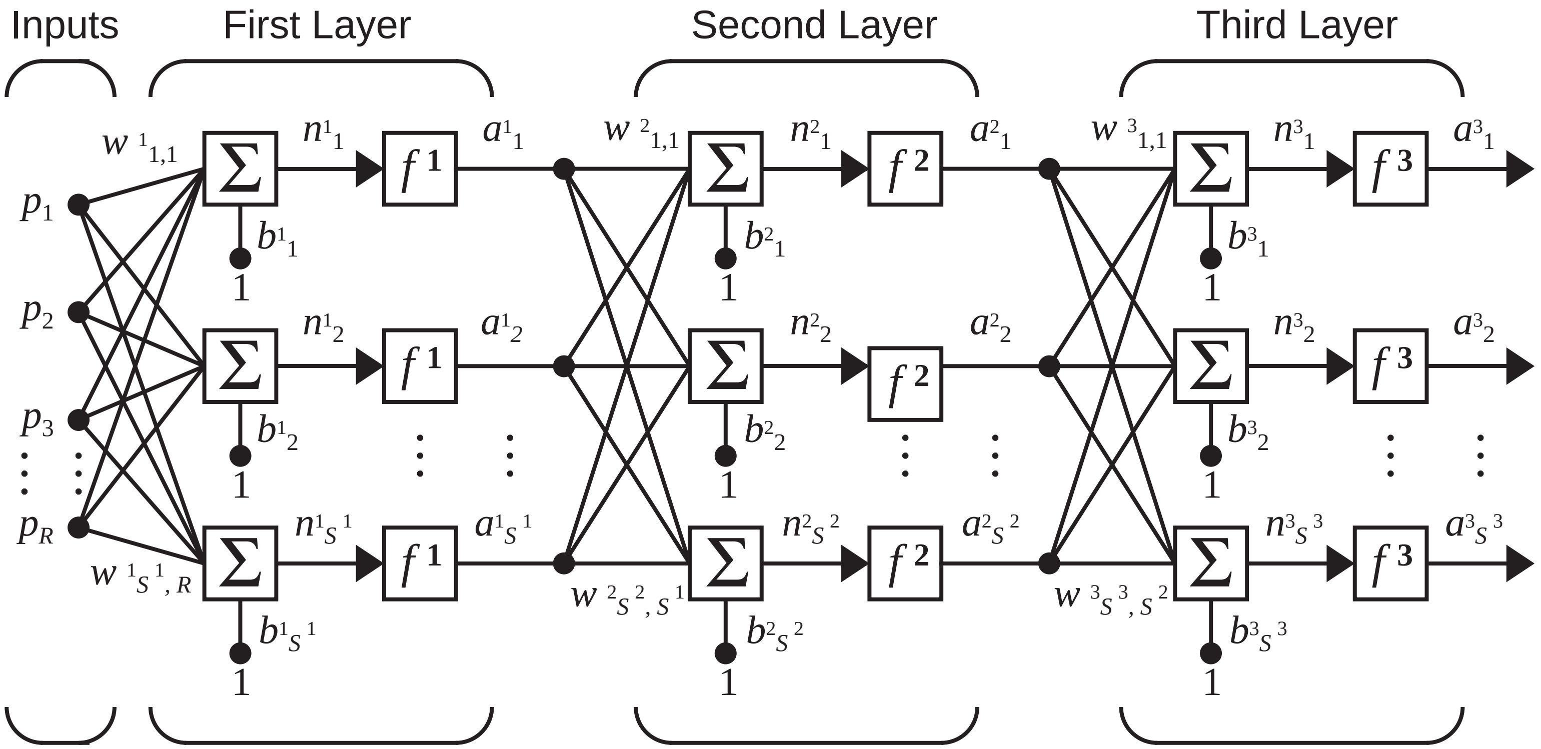


$$\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

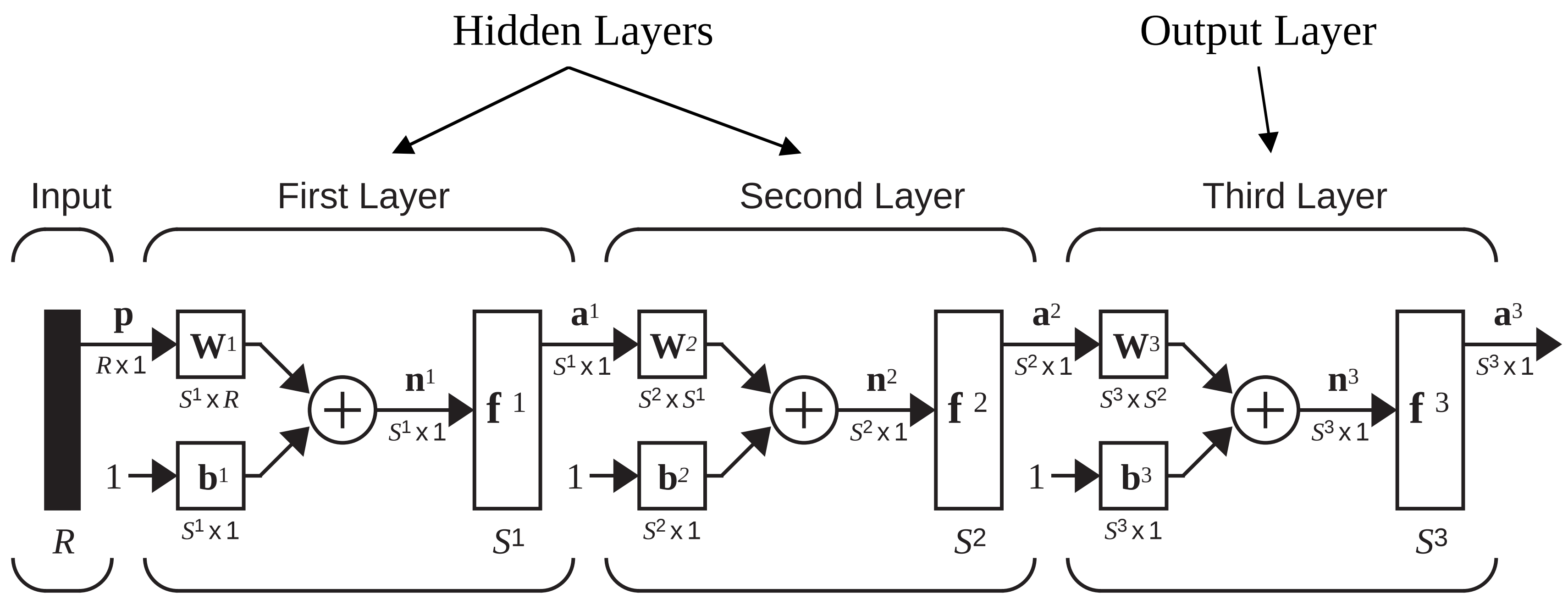
Multilayer Network



$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

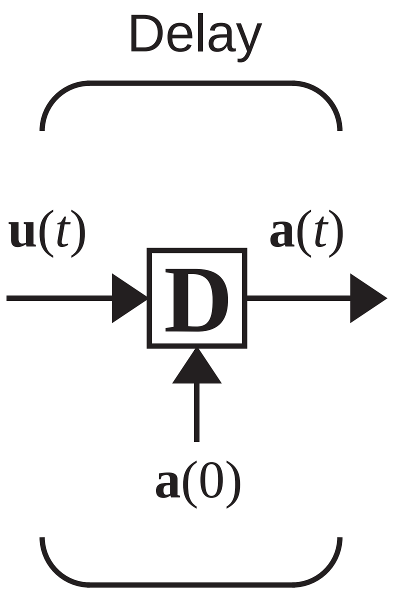
Abreviated Notation



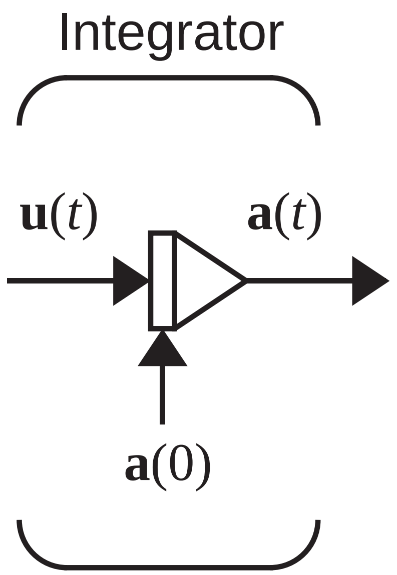
$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3)$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

Delays and Integrators

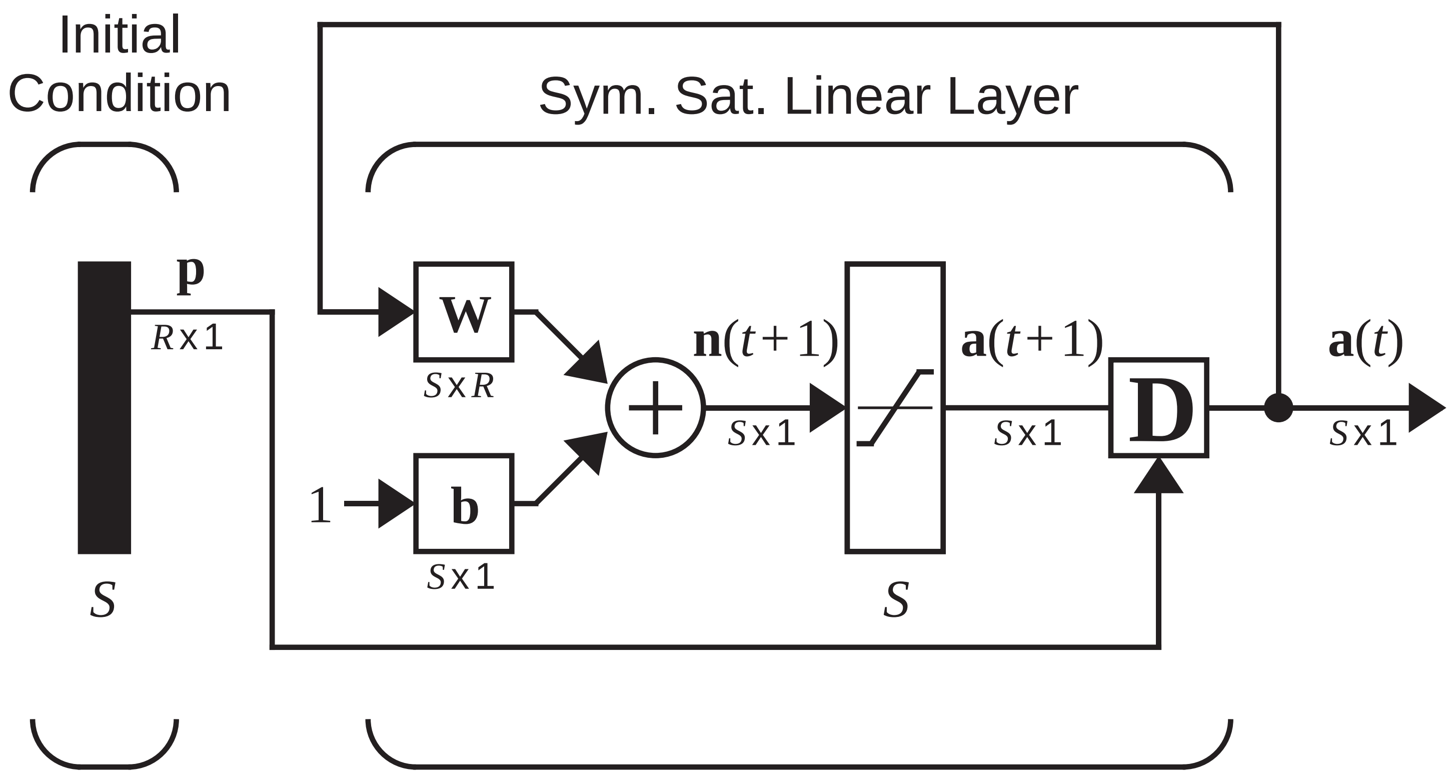


$$\mathbf{a}(t) = \mathbf{u}(t - 1)$$



$$\mathbf{a}(t) = \int_0^t \mathbf{u}(\tau)\, d\tau + \mathbf{a}(0)$$

Recurrent Network



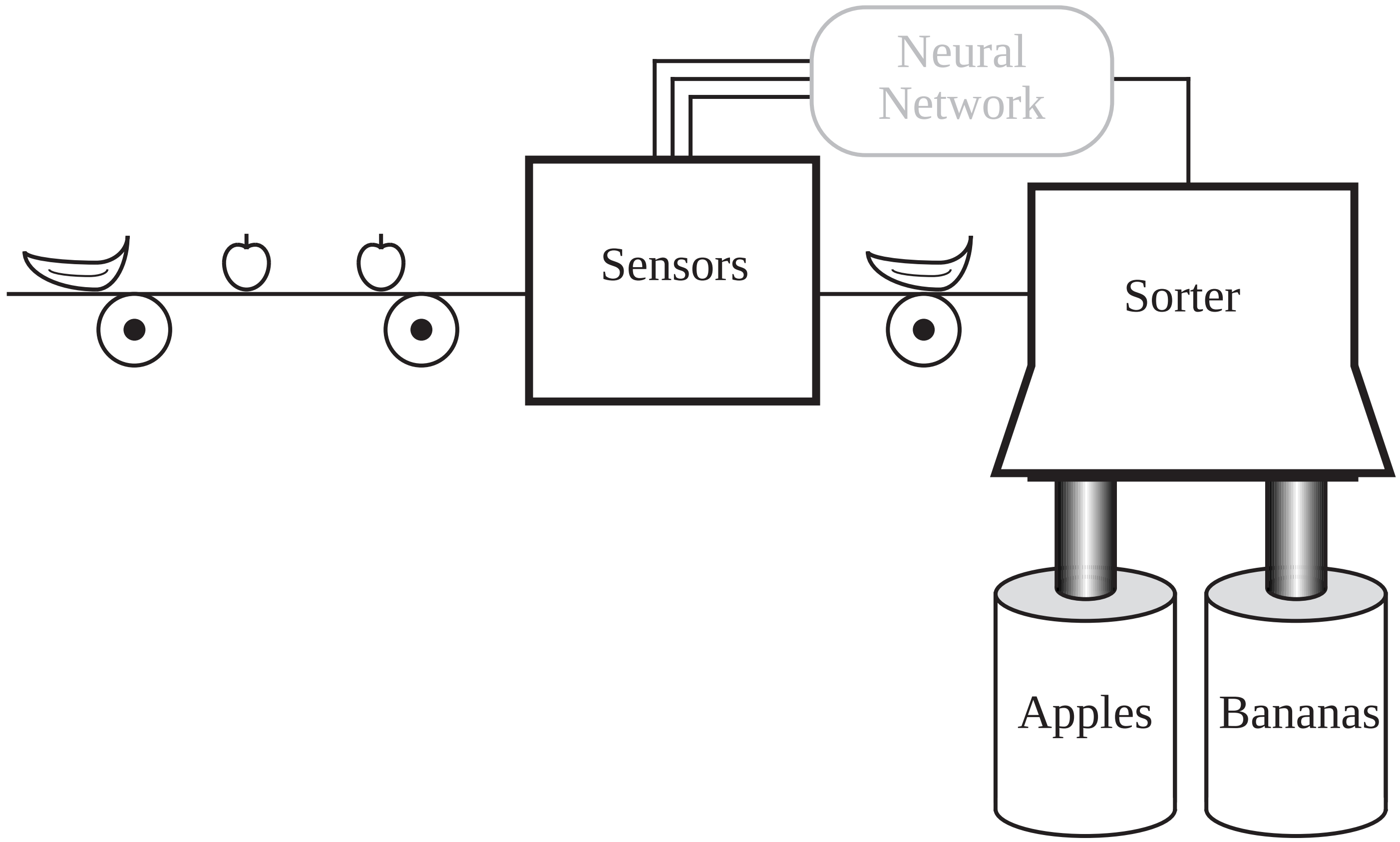
$$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlin}(\mathbf{Wa}(t) + \mathbf{b})$$

$$\mathbf{a}(1) = \mathbf{satlins}(\mathbf{Wa}(0) + \mathbf{b}) = \mathbf{satlins}(\mathbf{Wp} + \mathbf{b})$$

$$\mathbf{a}(2) = \mathbf{satlins}(\mathbf{Wa}(1) + \mathbf{b})$$

An Illustrative Example

Apple/Banana Sorter



Prototype Vectors

Measurement Vector

$$\mathbf{p} = \begin{bmatrix} \text{shape} \\ \text{texture} \\ \text{weight} \end{bmatrix}$$

Prototype Banana

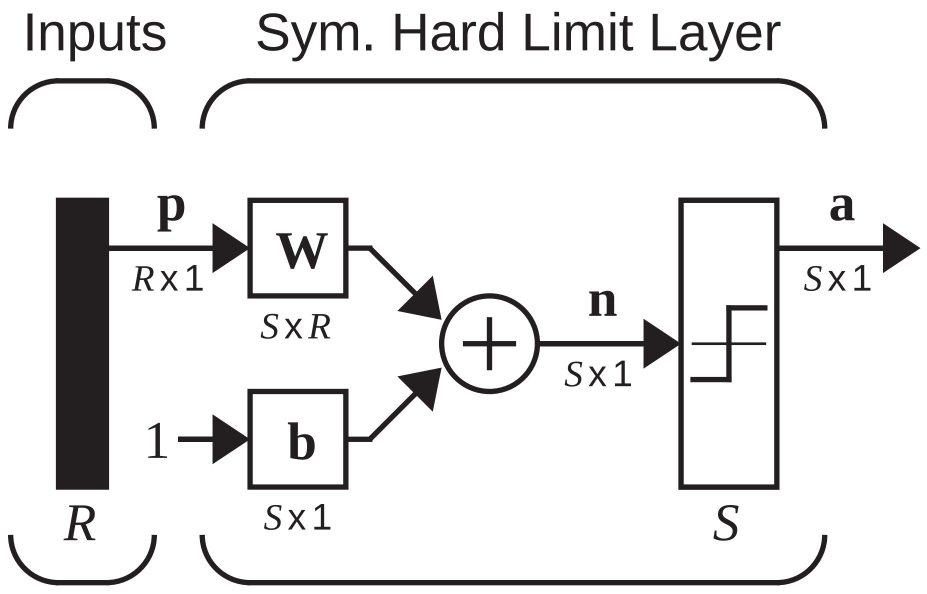
$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

Prototype Apple

$$\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

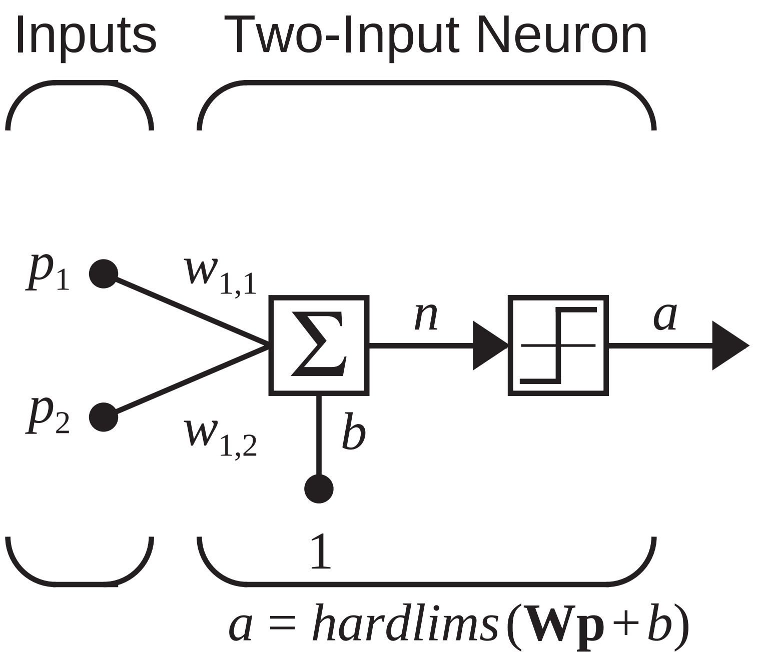
Shape: {1 : round ; -1 : eliptical}
Texture: {1 : smooth ; -1 : rough}
Weight: {1 : > 1 lb. ; -1 : < 1 lb.}

Perceptron

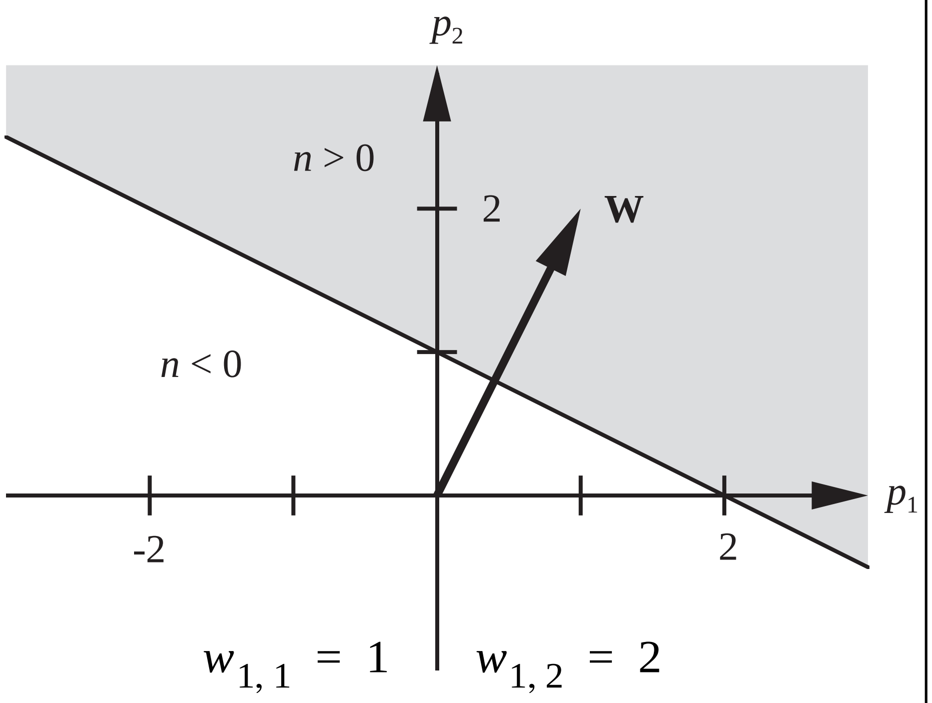


$$\mathbf{a} = \mathbf{hardlims}(\mathbf{Wp} + \mathbf{b})$$

Two-Input Case



$a = hardlims(\mathbf{Wp} + b)$



$w_{1,1} = 1 \qquad w_{1,2} = 2$

$$a = hardlims(n) = hardlims(\begin{bmatrix} 1 & 2 \end{bmatrix}\mathbf{p} + (-2))$$

Decision Boundary

$$\mathbf{Wp} + b = 0 \qquad \begin{bmatrix} 1 & 2 \end{bmatrix}\mathbf{p} + (-2) = 0$$

Apple/Banana Example

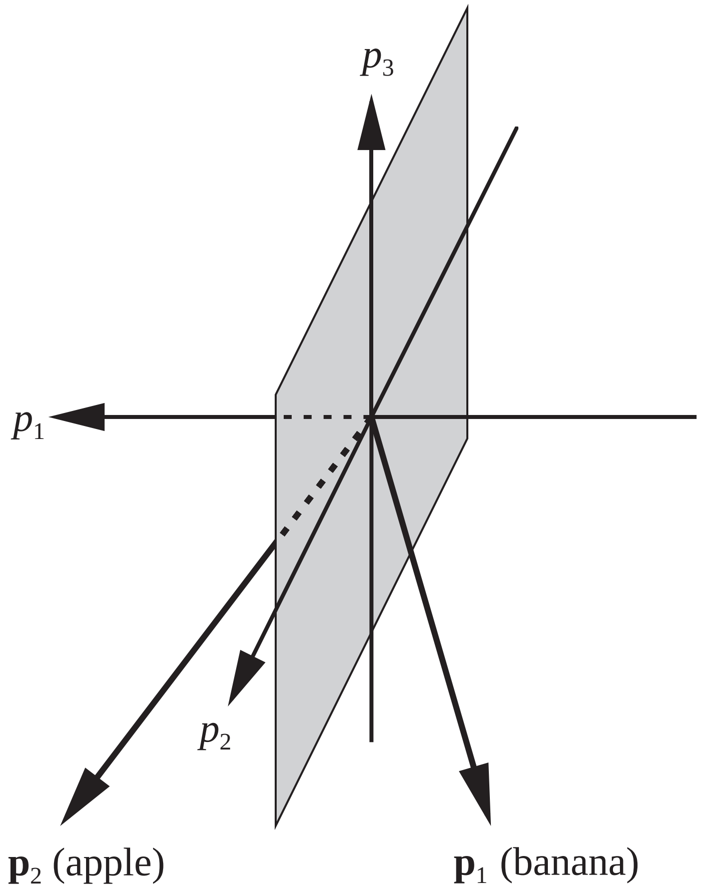
$$a = hardlims\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + b\right)$$

The decision boundary should separate the prototype vectors.

$$p_1 = 0$$

The weight vector should be orthogonal to the decision boundary, and should point in the direction of the vector which should produce an output of 1. The bias determines the position of the boundary

$$\begin{bmatrix} -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + 0 = 0$$



Testing the Network

Banana:

$$a = hardlims\left(\begin{bmatrix} -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0 \right) = 1(\text{banana})$$

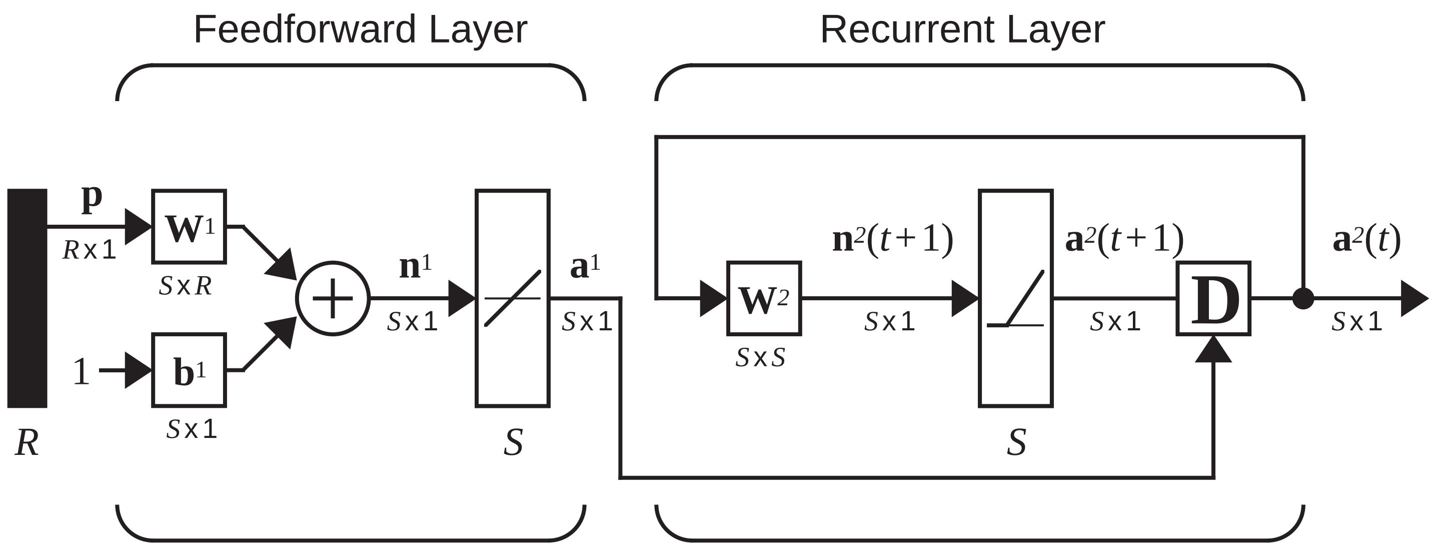
Apple:

$$a = hardlims\left(\begin{bmatrix} -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 0 \right) = -1\ (\text{apple})$$

“Rough” Banana:

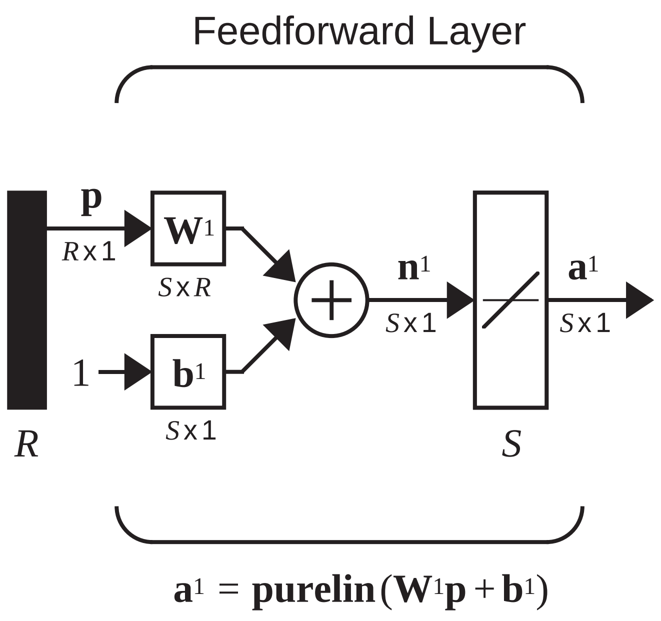
$$a = hardlims\left(\begin{bmatrix} -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + 0 \right) = 1(\text{banana})$$

Hamming Network



$\mathbf{a}^1 = \mathbf{purelin}(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1)$ $\quad\mathbf{a}^2(0) = \mathbf{a}^1 \quad \mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$

Feedforward Layer



For Banana/Apple Recognition

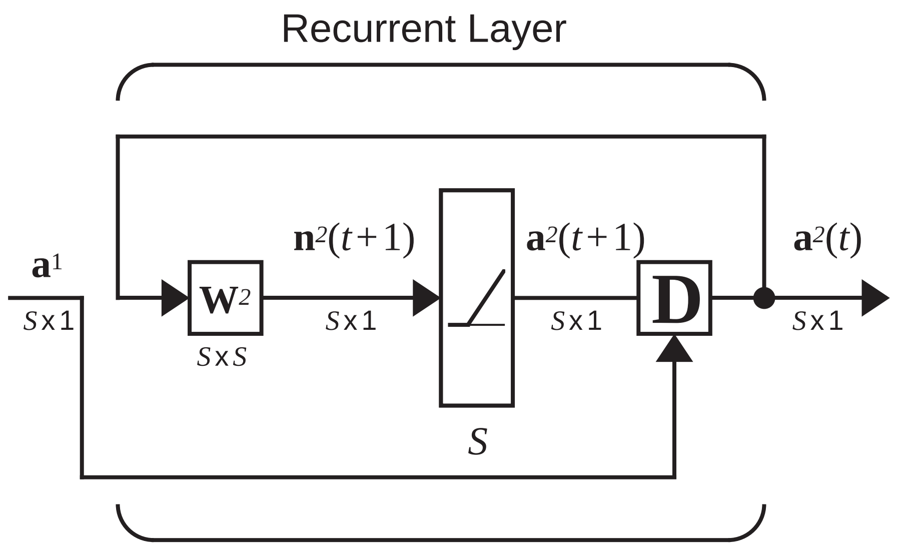
$$S = 2$$

$$\mathbf{W}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{b}^1 = \begin{bmatrix} R \\ R \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix}\mathbf{p} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + 3 \\ \mathbf{p}_2^T\mathbf{p} + 3 \end{bmatrix}$$

Recurrent Layer



$\mathbf{a}^2(0) = \mathbf{a}^1 \qquad \mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix} \qquad \varepsilon < \frac{1}{S-1}$$

$$\mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

Hamming Operation

First Layer

Input (Rough Banana)

$$\mathbf{p} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\mathbf{a}^1 = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} (1+3) \\ (-1+3) \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

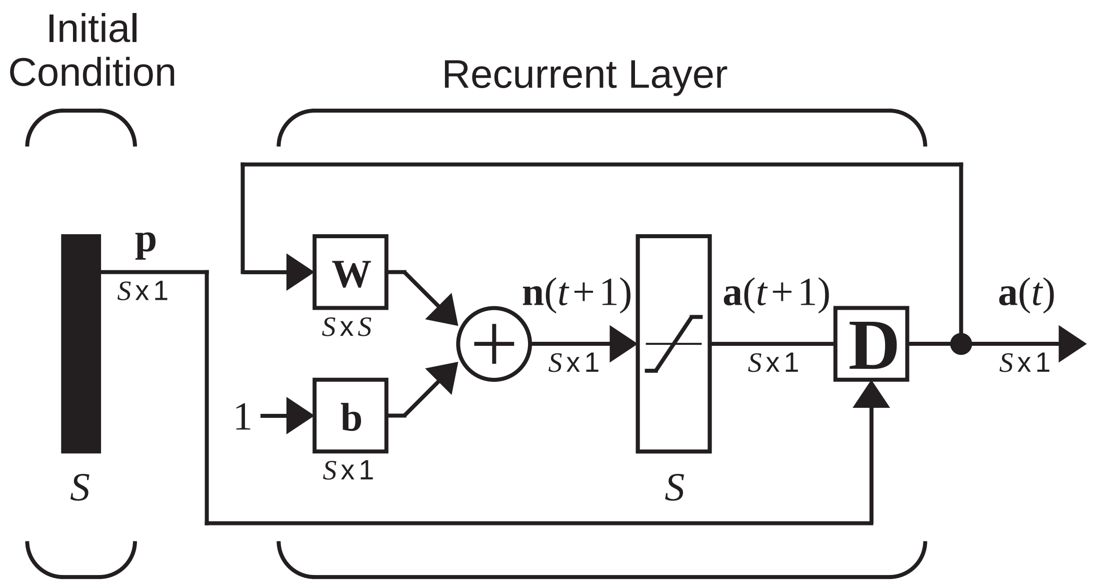
Hamming Operation

Second Layer

$$\mathbf{a}^2(1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(0)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 4 \\ 2 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

$$\mathbf{a}^2(2) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(1)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ -1.5 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

Hopfield Network



$$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(t) + \mathbf{b})$$

Apple/Banana Problem

$$\mathbf{W} = \begin{bmatrix} 1.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0.9 \\ -0.9 \end{bmatrix}$$

$$a_1(t+1) = satlins(1.2a_1(t))$$

$$a_2(t+1) = satlins(0.2a_2(t) + 0.9)$$

$$a_3(t+1) = \mathrm{satlins}(0.2a_3(t) - 0.9)$$

Test: “Rough” Banana

$$\mathbf{a}(0) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{a}(1) = \begin{bmatrix} -1 \\ 0.7 \\ -1 \end{bmatrix} \quad \mathbf{a}(2) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{a}(3) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$ (Banana)

Summary

- Perceptron
  - Feedforward Network
  - Linear Decision Boundary
  - One Neuron for Each Decision
- Hamming Network
  - Competitive Network
  - First Layer – Pattern Matching (Inner Product)
  - Second Layer – Competition (Winner-Take-All)
  - # Neurons = # Prototype Patterns
- Hopfield Network
  - Dynamic Associative Memory Network
  - Network Output Converges to a Prototype Pattern
  - # Neurons = # Elements in each Prototype Pattern

Perceptron Learning Rule

Learning Rules

- Supervised Learning

  Network is provided with a set of examples of proper network behavior (inputs/targets)

$$\{\mathbf{p}_1,\mathbf{t}_1\}, \{\mathbf{p}_2,\mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q,\mathbf{t}_Q\}$$

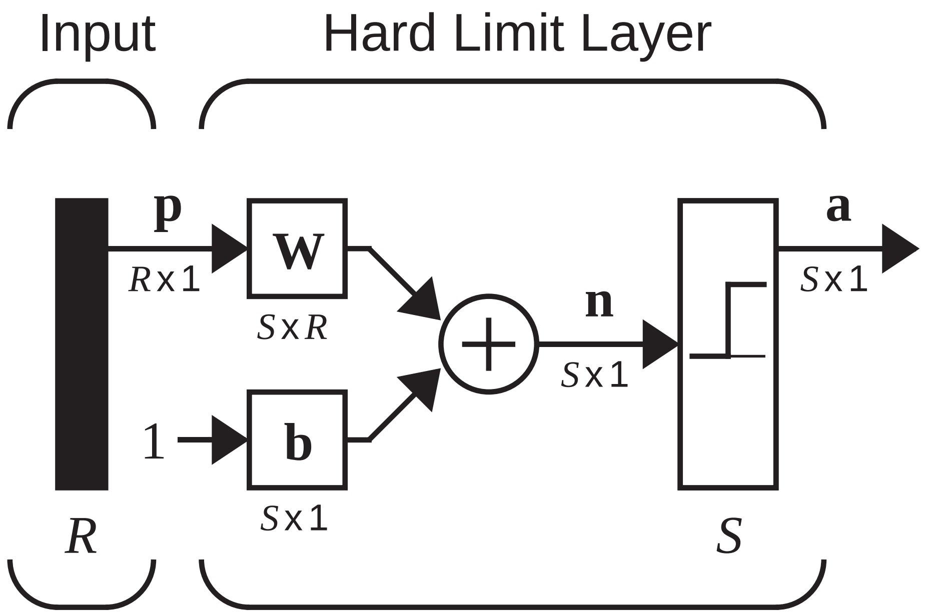
- Reinforcement Learning

  Network is only provided with a grade, or score, which indicates network performance

- Unsupervised Learning

  Only network inputs are available to the learning algorithm. Network learns to categorize (cluster) the inputs.

Perceptron Architecture



$$\mathbf{a} = \mathbf{hardlim}(\mathbf{Wp} + \mathbf{b})$$

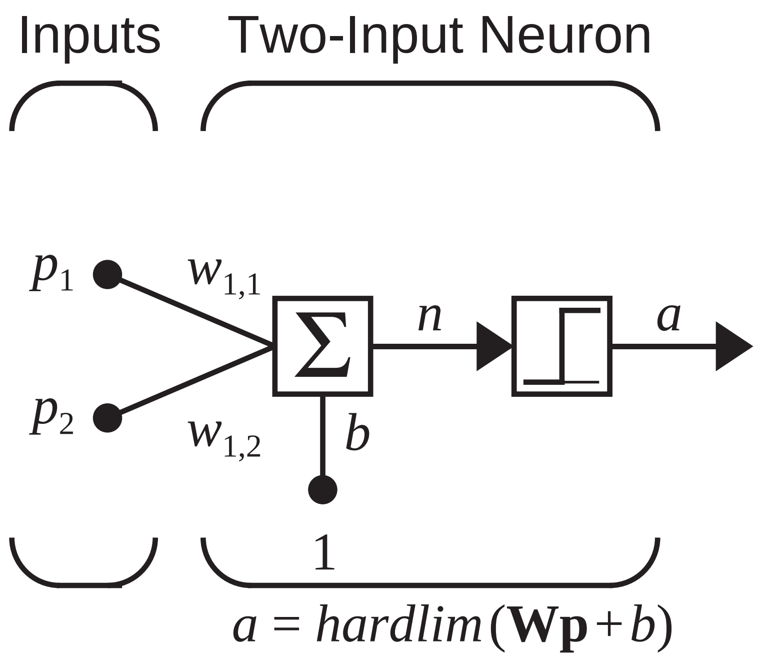
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

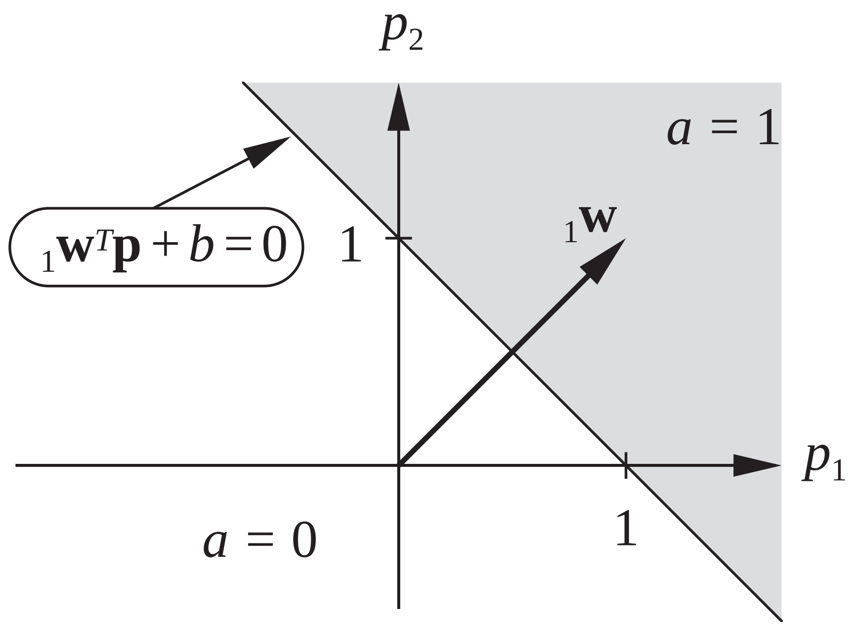
$${}_{\mathrm{i}}\mathbf{w} = \begin{bmatrix} w_{\mathrm{i},1} \\ w_{\mathrm{i},2} \\ \vdots \\ w_{\mathrm{i},R} \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^{\mathrm{T}} \\ {}_2\mathbf{w}^{\mathrm{T}} \\ \vdots \\ {}_S\mathbf{w}^{\mathrm{T}} \end{bmatrix}$$

$$a_i = hardlim(n_i) = hardlim({}_i\mathbf{w}^{\mathrm{T}}\mathbf{p} + b_i)$$

Single-Neuron Perceptron

$w_{1,1} = 1 \qquad w_{1,2} = 1 \qquad b = -1$



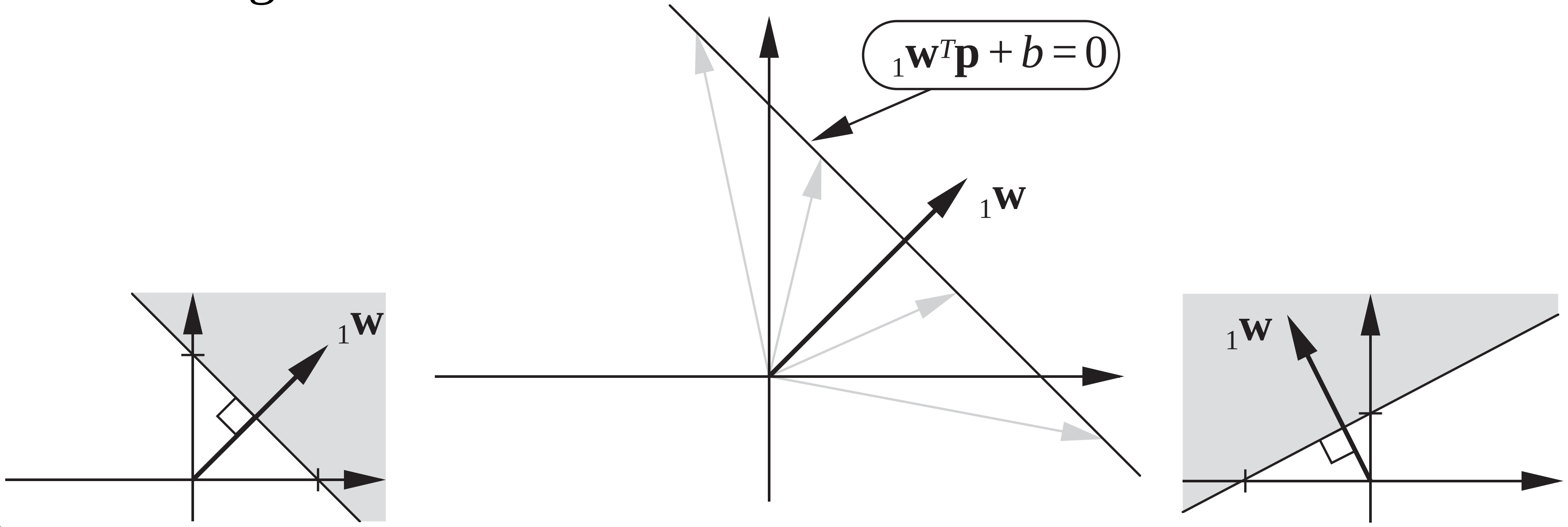


$$a = hardlim({}_1\mathbf{w}^{\mathrm{T}}\mathbf{p} + b) = hardlim(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

Decision Boundary

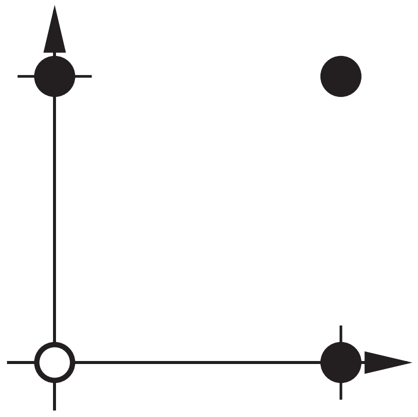
$${}_1\mathbf{w}^T\mathbf{p} + b = 0 \qquad\qquad {}_1\mathbf{w}^T\mathbf{p} = -b$$

- All points on the decision boundary have the same inner product with the weight vector.
- Therefore they have the same projection onto the weight vector, and they must lie on a line orthogonal to the weight vector

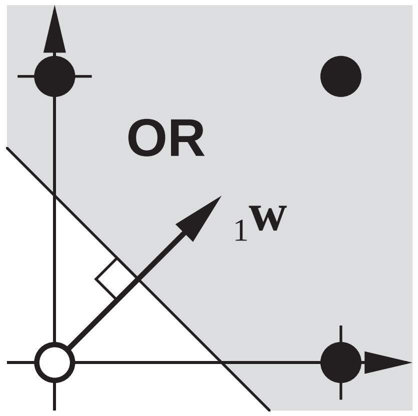


Example - OR

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 1\right\}$$



OR Solution



Weight vector should be orthogonal to the decision boundary.

$$_1\mathbf{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

Pick a point on the decision boundary to find the bias.

$$_1\mathbf{w}^T\mathbf{p} + b = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + \mathrm{b} = 0 \quad \Rightarrow \quad b = -0.25$$

Multiple-Neuron Perceptron

Each neuron will have its own decision boundary.

$${}_{i}\mathbf{w}^{T}\mathbf{p} + b_{i} = 0$$

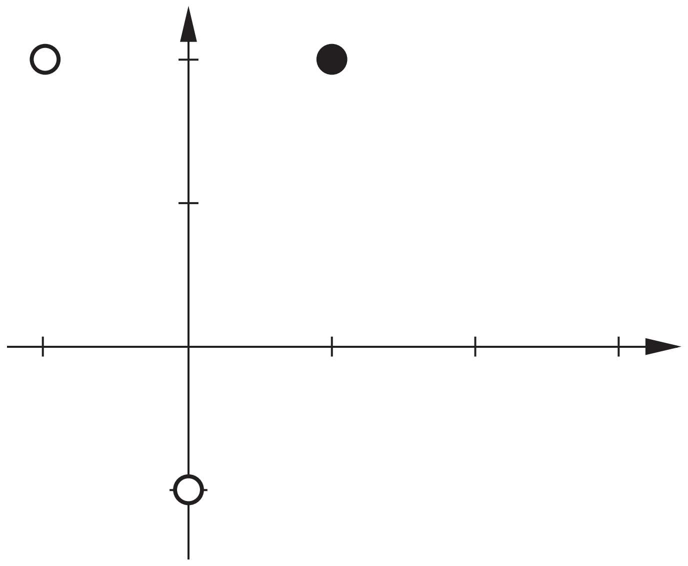
A single neuron can classify input vectors into two categories.

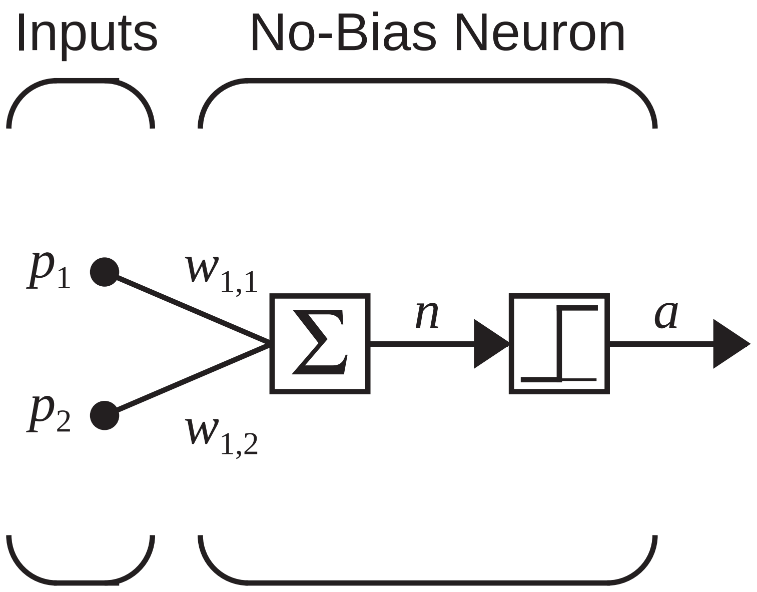
A multi-neuron perceptron can classify input vectors into 2^S categories.

Learning Rule Test Problem

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$



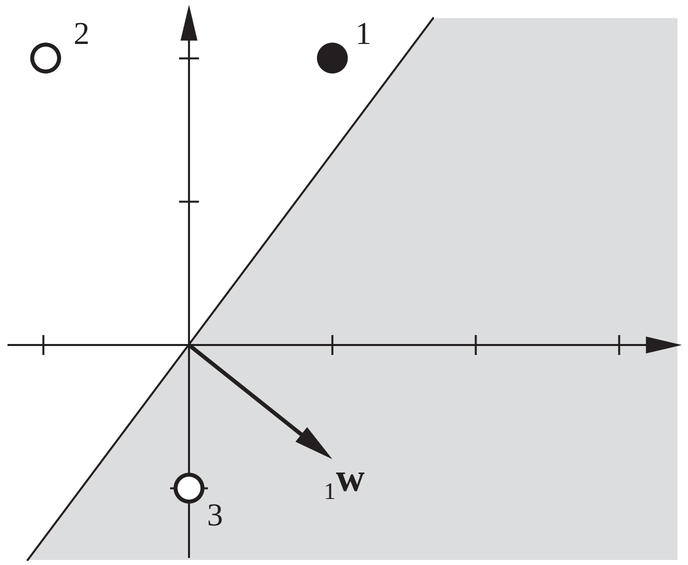


$a = hardlim(\mathbf{Wp})$

Starting Point

Random initial weight:

$${}_1\mathbf{w} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix}$$



Present $\mathbf{p}_1$ to the network:

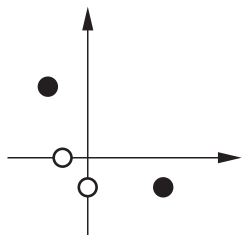
$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

$$a = hardlim(-0.6) = 0$$

Incorrect Classification.

Tentative Learning Rule

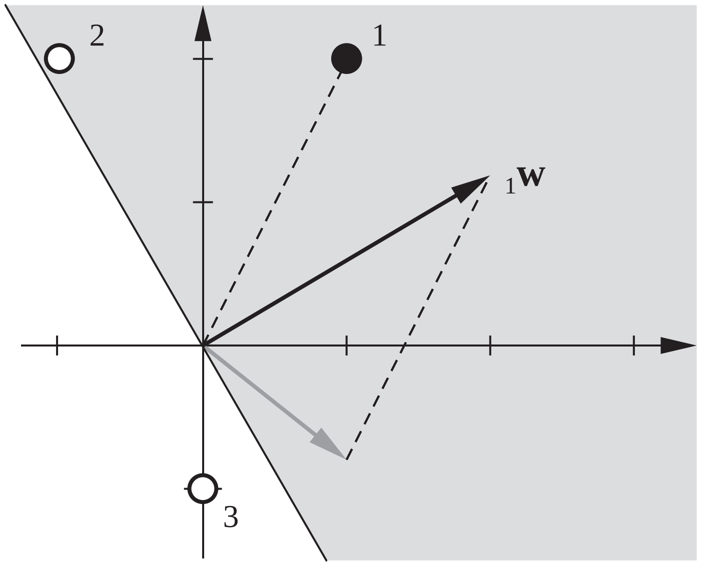
- Set ${}_1\mathbf{w}$ to $\mathbf{p}_1$ ✗
  - Not stable



- Add $\mathbf{p}_1$ to ${}_1\mathbf{w}$ ✓

Tentative Rule: If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

$$ {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} $$



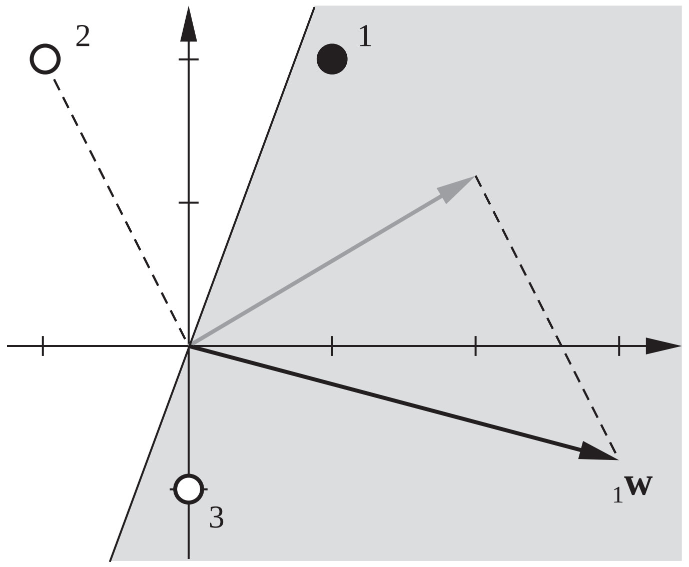
Second Input Vector

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_2) = hardlim\left(\begin{bmatrix} 2.0 & 1.2 \end{bmatrix}\begin{bmatrix} -1 \\ 2 \end{bmatrix}\right)$$

$a = hardlim(0.4) = 1$ (Incorrect Classification)

Modification to Rule: If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_2 = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix}$$

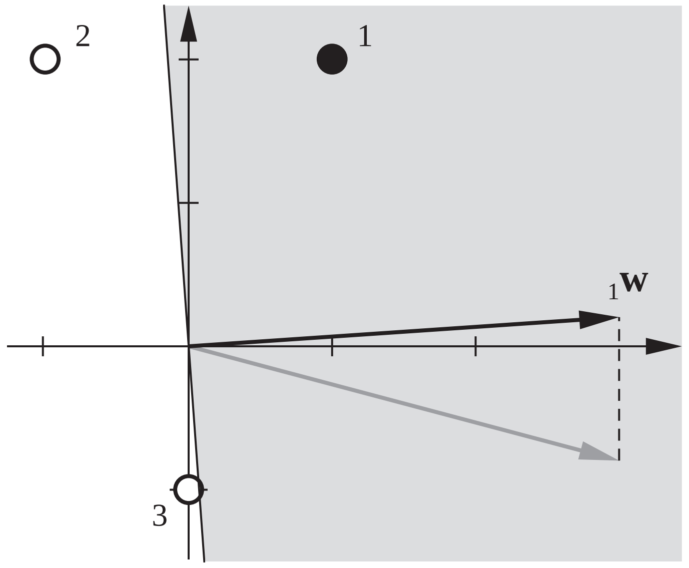


Third Input Vector

$$a = hardlim({}_1\mathbf{w}^T \mathbf{p}_3) = hardlim\left(\begin{bmatrix} 3.0 & -0.8 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

$a = \mathrm{hardlim}(0.8) = 1$ (Incorrect Classification)

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}_3 = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$



Patterns are now correctly classified.

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$.

Unified Learning Rule

If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$e = t - a$$

If $e = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $e = -1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $e = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p}$$

$$b^{new} = b^{old} + e$$

A bias is a weight with an input of 1.

Multiple-Neuron Perceptrons

To update the ith row of the weight matrix:

$${}_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\mathbf{p}$$

$$b_i^{new} = b_i^{old} + e_i$$

Matrix form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T$$

$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e}$$

Apple/Banana Example

Training Set

$$\left\{\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = \begin{bmatrix} 1 \end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \end{bmatrix}\right\}$$

Initial Weights

$$\mathbf{W} = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} \qquad b = 0.5$$

First Iteration

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$a = hardlim(-0.5) = 0 \qquad e = t_1 - a = 1 - 0 = 1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} + (1)\begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (1) = 1.5$$

Second Iteration

$$a = hardlim(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} + (1.5)\right)$$

$$a = hardlim(2.5) = 1$$

$$e = t_2 - a = 0 - 1 = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix} + (-1)\begin{bmatrix}1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-1.5 & -1 & -0.5\end{bmatrix}$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

Check

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$a = hardlim(1.5) = 1 = t_1$$

$$a = hardlim(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$a = hardlim(-1.5) = 0 = t_2$$

Perceptron Rule Capability

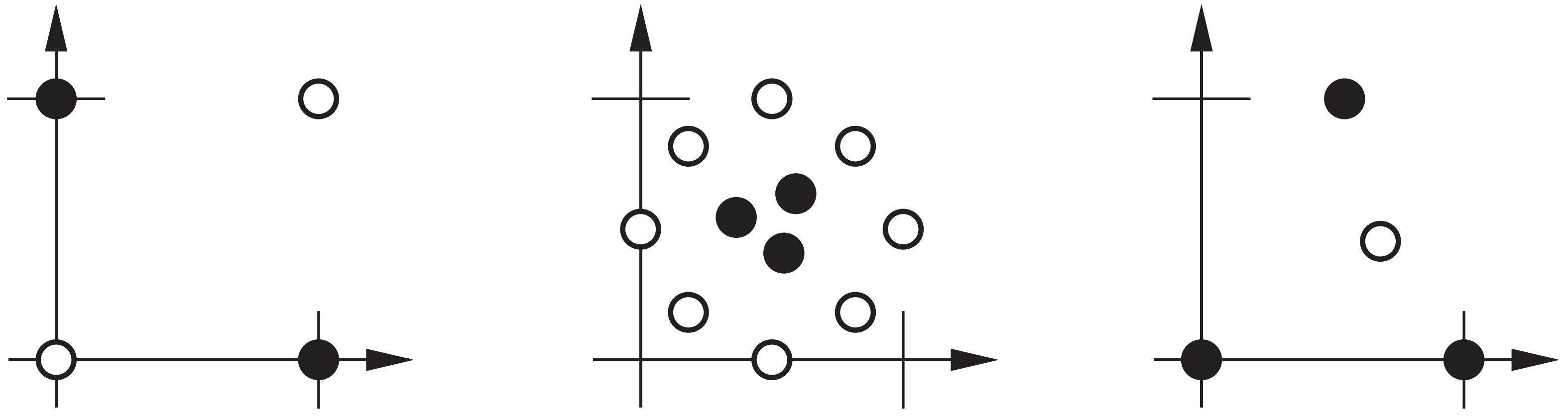
The perceptron rule will always converge to weights which accomplish the desired classification, assuming that such weights exist.

Perceptron Limitations

Linear Decision Boundary

$$ {}_1\mathbf{w}^T \mathbf{p} + b = 0 $$

Linearly Inseparable Problems



Signal & Weight Vector Spaces

Notation

Vectors in $\Re^n$.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Generalized Vectors.

$$x$$

Vector Space

1. An operation called vector addition is defined such that if $x \in X$ and $y \in X$ then $x+y \in X$.
2. $x + y = y + x$
3. $(x + y) + z = x + (y + z)$
4. There is a unique vector $0 \in X$, called the zero vector, such that $x + 0 = x$ for all $x \in X$.
5. For each vector there is a unique vector in X, to be called $(-x)$, such that $x + (-x) = 0$.

Vector Space (Cont.)

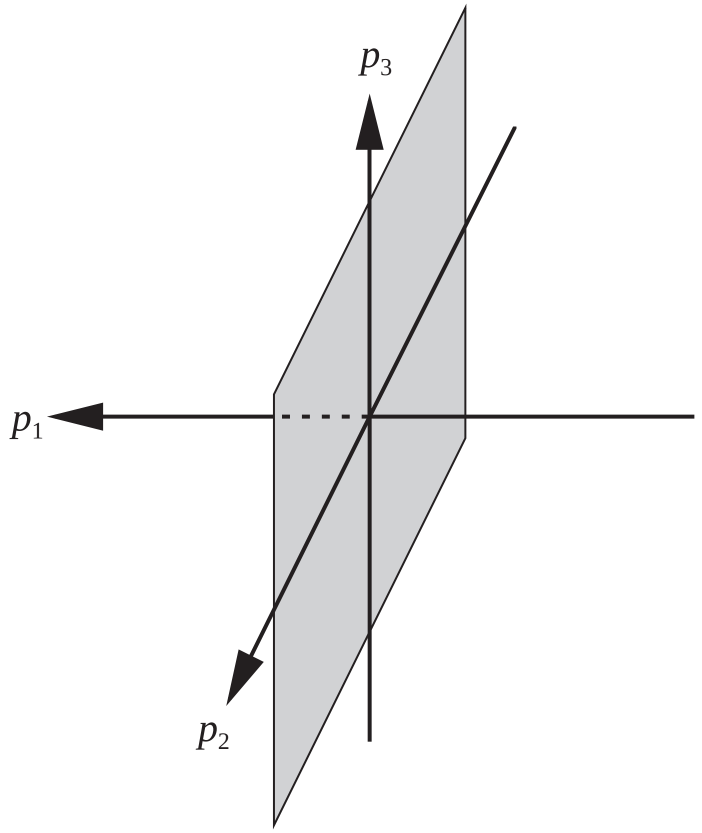
6. An operation, called multiplication, is defined such that for all scalars $a \in F$, and all vectors $x \in X$, $a\,x \in X$.
7. For any $x \in X$, $1x = x$ (for scalar 1).
8. For any two scalars $a \in F$ and $b \in F$, and any $x \in X$, $a(bx) = (ab)x$.
9. $(a + b)x = ax + bx$.
10. $a(x + y) = a\,x + a\,y$

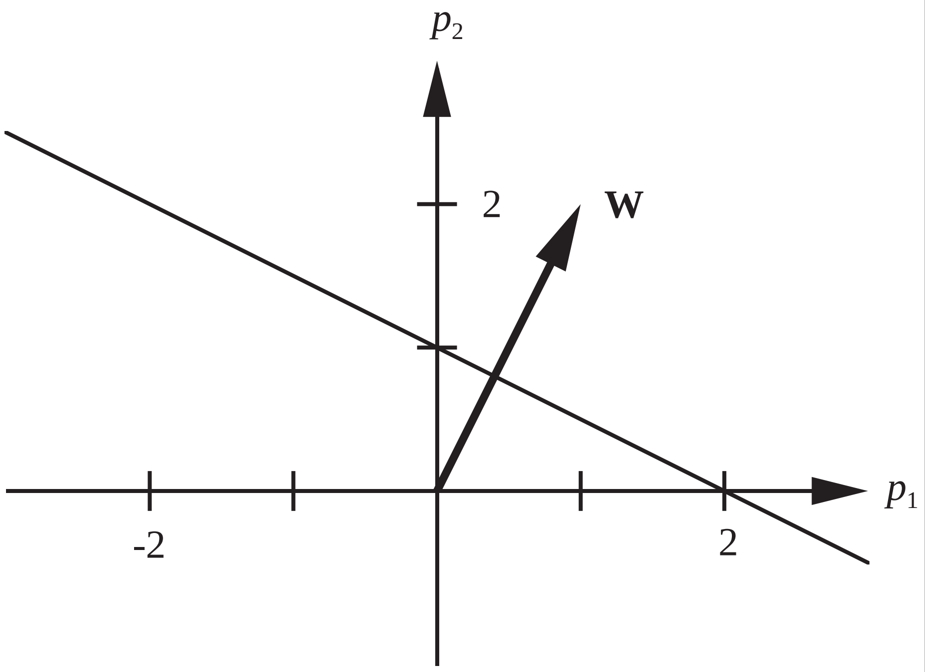
5

Examples (Decision Boundaries)

Is the p_2, p_3 plane a vector space?

Is the line $p_1 + 2p_2 - 2 = 0$ a vector space?





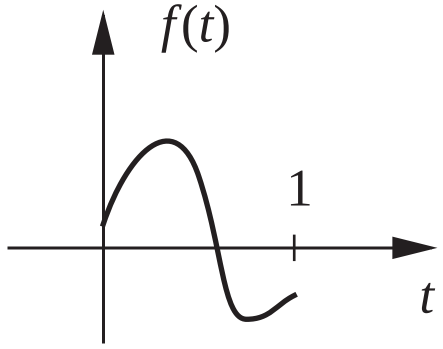
Other Vector Spaces

Polynomials of degree 2 or less.

$$x = 2 + t + 4t^2$$

$$y = 1 + 5t$$

Continuous functions in the interval [0,1].



Linear Independence

If

$$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \cdots + a_n\mathbf{x}_n = \mathbf{0}$$

implies that each

$$a_i = 0$$

then

$$\{\mathbf{x}_i\}$$

is a set of linearly independent vectors.

Example (Banana and Apple)

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \qquad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

Let

$$a_1\mathbf{p}_1 + a_2\mathbf{p}_2 = \mathbf{0}$$

$$\begin{bmatrix} -a_1 + a_2 \\ a_1 + a_2 \\ -a_1 + (-a_2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

This can only be true if

$$a_1 = a_2 = 0$$

Therefore the vectors are independent.

Spanning a Space

A subset **spans** a space if every vector in the space can be written as a linear combination of the vectors in the subspace.

$$x = x_1 u_1 + x_2 u_2 + \cdots + x_m u_m$$

Basis Vectors

- A set of basis vectors for the space X is a set of vectors which spans X and is linearly independent.
- The dimension of a vector space, Dim(X), is equal to the number of vectors in the basis set.
- Let X be a finite dimensional vector space, then every basis set of X has the same number of elements.

Example

Polynomials of degree 2 or less.

Basis A:

$u_1 = 1 \qquad u_2 = t \qquad u_3 = t^2$

Basis B:

$u_1 = 1 - t \qquad u_2 = 1 + t \qquad u_3 = 1 + t + t^2$

(Any three linearly independent vectors in the space will work.)

How can you represent the vector $x = 1 + 2t$ using both basis sets?

5

Inner Product / Norm

A scalar function of vectors x and y can be defined as an **inner product**, (x,y), provided the following are satisfied (for real inner products):

- $(x,y) = (y,x)$.
- $(x,\mathrm{a}y_1+\mathrm{b}y_2) = \mathrm{a}(x,y_1) + \mathrm{b}(x,y_2)$.
- $(x, x) \geq 0$, where equality holds iff $x = 0$.

A scalar function of a vector x is called a **norm**, $||x||$, provided the following are satisfied:

- $||x|| \geq 0$.
- $||x|| = 0$ iff $x = 0$.
- $||a\, x|| = |a|\; ||x||$ for scalar a .
- $||x + y|| \leq ||x|| + ||y||$.

5

Example

Standard Euclidean Inner Product

$$\mathbf{x}^{\mathrm{T}}\mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

Standard Euclidean Norm

$$\|x\| = (x\ ,\ x)^{1/2}$$

$$\|\mathbf{x}\| = (\mathbf{x}^T\mathbf{x})^{1/2} = (x_1{}^2 + x_2{}^2 + \ldots + x_\mathrm{n}{}^2)^{\ 1/2}$$

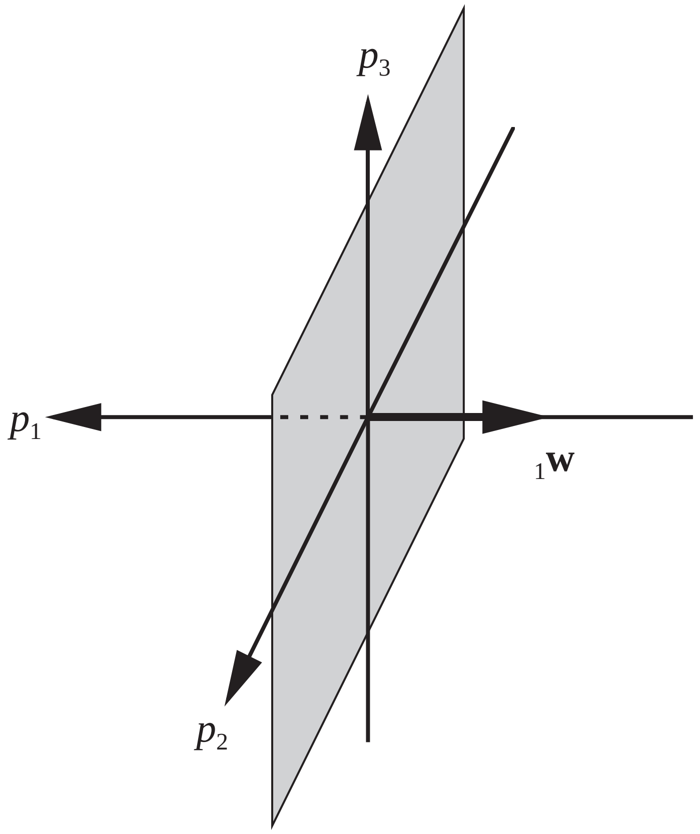
Angle

$$\cos(\theta) = (x,y)/(\|x\|\ \|y\|)$$

Orthogonality

Two vectors $x, y \in X$ are orthogonal if $(x, y) = 0$.

Example



Any vector in the p_2, p_3 plane is orthogonal to the weight vector.

5

Gram-Schmidt Orthogonalization

Independent Vectors $\Rightarrow$ Orthogonal Vectors

$y_1, y_2, \ldots, y_n \Rightarrow v_1, v_2, \ldots, v_n$

Step 1: Set first orthogonal vector to first independent vector.

$$v_1 = y_1$$

Step 2: Subtract the portion of y_2 that is in the direction of v_1.

$$v_2 = y_2 - a v_1$$

Where a is chosen so that v_2 is orthogonal to v_1:

$$(v_1, v_2) = (v_1, y_2 - a v_1) = (v_1, y_2) - a(v_1, v_1) = 0$$

$$a = \frac{(v_1, y_2)}{(v_1, v_1)}$$

Gram-Schmidt (Cont.)

Projection of y_2 on v_1:

$$\frac{(v_1, y_2)}{(v_1, v_1)} v_1$$

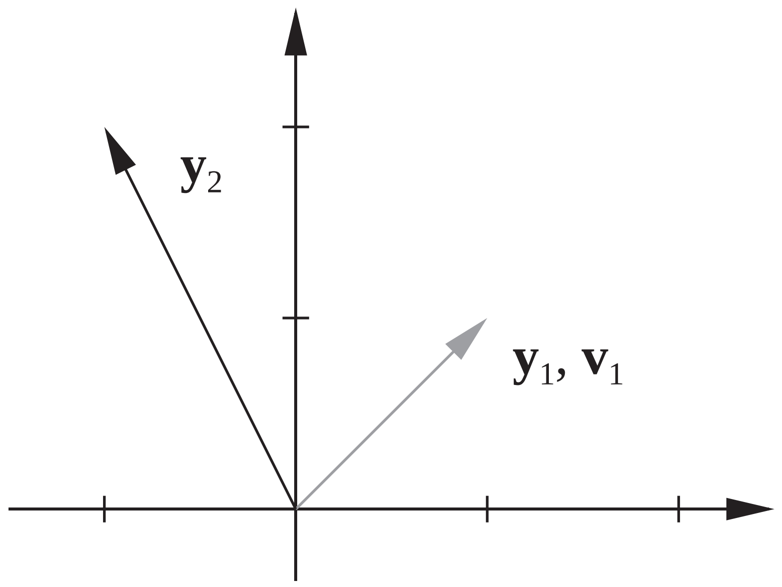
Step k: Subtract the portion of y_k that is in the direction of all previous v_i .

$$v_k = y_k - \sum_{i=1}^{k-1} \frac{(v_i, y_k)}{(v_i, v_i)} v_i$$

Example

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{y}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

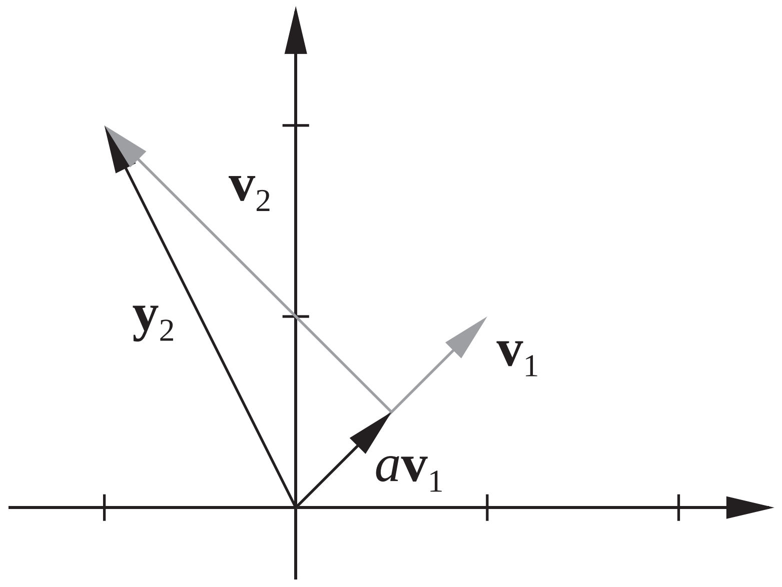
Step 1. $\mathbf{v}_1 = \mathbf{y}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



Example (Cont.)

Step 2.

$$\mathbf{v}_2 = \mathbf{y}_2 - \frac{\mathbf{v}_1^T\mathbf{y}_2}{\mathbf{v}_1^T\mathbf{v}_1}\mathbf{v}_1 = \begin{bmatrix} -1 \\ 2 \end{bmatrix} - \frac{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} -1 \\ 2 \end{bmatrix}}{\begin{bmatrix} 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \end{bmatrix}$$



Vector Expansion

If a vector space X has a basis set $\{v_1, v_2, ..., v_n\}$, then any $x \in X$ has a unique vector expansion:

$$x = \sum_{i=1}^{n} x_i v_i = x_1 v_1 + x_2 v_2 + \cdots + x_n v_n$$

If the basis vectors are **orthogonal**, and we take the inner product of v_j and x :

$$(v_j, x) = (v_j, \sum_{i=1}^{n} x_i v_i) = \sum_{i=1}^{n} x_i (v_j, v_i) = x_j (v_j, v_j)$$

Therefore the coefficients of the expansion can be computed:

$$x_j = \frac{(v_j, x)}{(v_j, v_j)}$$

Column of Numbers

The vector expansion provides a meaning for writing a vector as a column of numbers.

$$\chi = \sum_{i=1}^{n} x_i \mathrm{v}_i = x_1 \mathrm{v}_1 + x_2 \mathrm{v}_2 + \cdots + x_n \mathrm{v}_n$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

To interpret **x**, we need to know what basis was used for the expansion.

Reciprocal Basis Vectors

Definition of reciprocal basis vectors, r_i:

$$(r_i, v_j) = 0 \qquad i \neq j$$
$$= 1 \qquad i = j$$

where the basis vectors are $\{v_1, v_2, ..., v_n\}$, and the reciprocal basis vectors are $\{r_1, r_2, ..., r_n\}$.

For vectors in $\Re^n$ we can use the following inner product:

$$(r_i, v_j) = \mathbf{r}_i^T \mathbf{v}_j$$

Therefore, the equations for the reciprocal basis vectors become:

$$\mathbf{R}^T\mathbf{B} = \mathbf{I} \quad \Rightarrow \quad \mathbf{R}^T = \mathbf{B}^{-1}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_n \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \dots & \mathbf{r}_n \end{bmatrix}$$

Vector Expansion

$$x = x_1 v_1 + x_2 v_2 + \cdots + x_n v_n$$

Take the inner product of the first reciprocal basis vector with the vector to be expanded:

$$(r_1, x) = x_1 (r_1, v_1) + x_2 (r_1, v_2) + \cdots + x_n (r_1, v_n)$$

By definition of the reciprocal basis vectors:

$$(r_1, v_2) = (r_1, v_3) = \cdots = (r_1, v_n) = 0$$

$$(r_1, v_1) = 1$$

Therefore, the first coefficient in the expansion is:

$$x_1 = (r_1, x)$$

In general, we then have (even for nonorthogonal basis vectors):

$$x_j = (r_j, x)$$

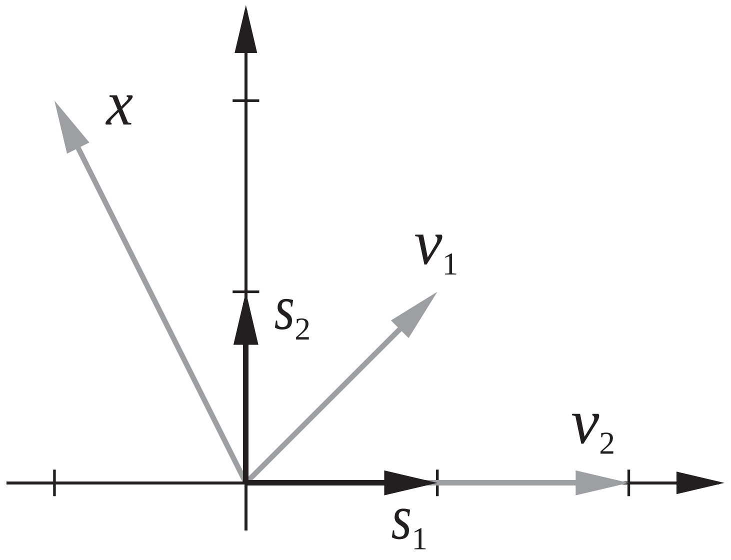
Example

Basis Vectors:

$$\mathbf{v}_1^s = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{v}_2^s = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Vector to Expand:

$$\mathbf{x}^s = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$



Example (Cont.)

Reciprocal Basis Vectors:

$$\mathbf{R}^T = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 0 & 1 \\ 0.5 & -0.5 \end{bmatrix} \qquad \mathbf{r}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \mathbf{r}_2 = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix}$$

Expansion Coefficients:

$$x_1^v = \mathbf{r}_1^T \mathbf{x}^s = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = 2$$

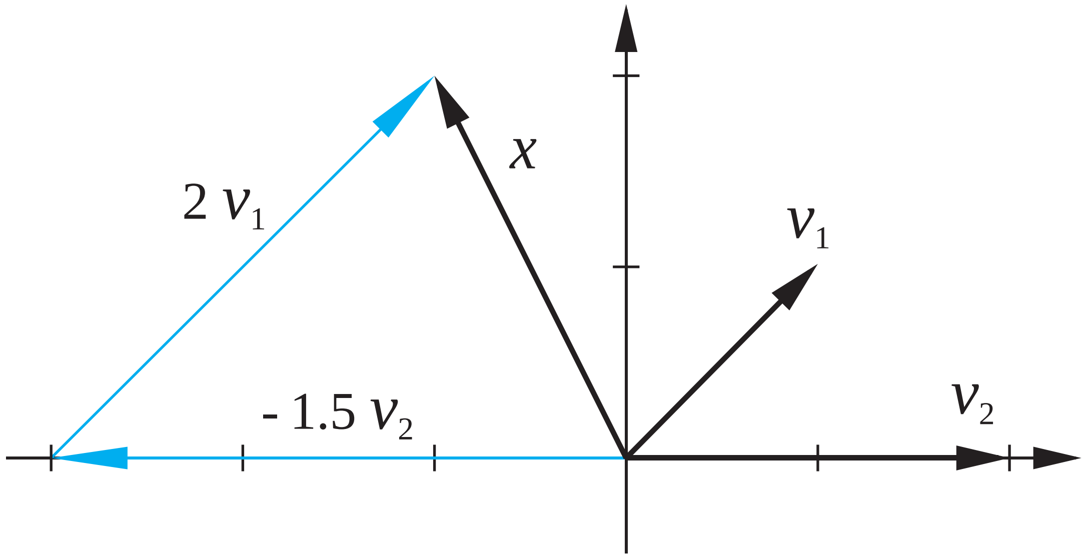
$$x_2^v = \mathbf{r}_2^T \mathbf{x}^s = \begin{bmatrix} 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = -1.5$$

Matrix Form:

$$\mathbf{x}^v = \mathbf{R}^T \mathbf{x}^s = \mathbf{B}^{-1} \mathbf{x}^s = \begin{bmatrix} 0 & 1 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ -1.5 \end{bmatrix}$$

Example (Cont.)

$$x = (-1)s_1 + 2s_2 = 2v_1 - 1.5v_2$$

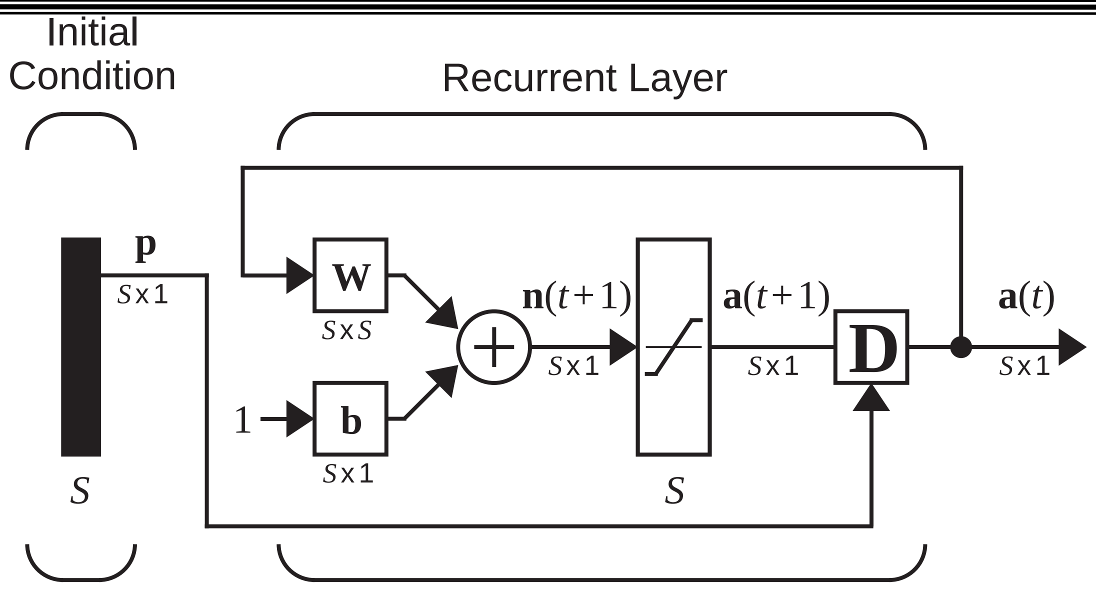


$$\mathbf{x}^s = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \qquad \mathbf{x}^v = \begin{bmatrix} 2 \\ -1.5 \end{bmatrix}$$

The interpretation of the column of numbers depends on the basis set used for the expansion.

Linear Transformations

Hopfield Network Questions



$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(t)+\mathbf{b})$

- The network output is repeatedly multiplied by the weight matrix W.
- What is the effect of this repeated operation?
- Will the output converge, go to infinity, oscillate?
- In this chapter we want to investigate matrix multiplication, which represents a general linear transformation.

Linear Transformations

A **transformation** consists of three parts:

1. A set of elements $X = \{x_i\}$, called the domain,
2. A set of elements $Y = \{y_i\}$, called the range, and
3. A rule relating each $x_i \in X$ to an element $y_i \in Y$.

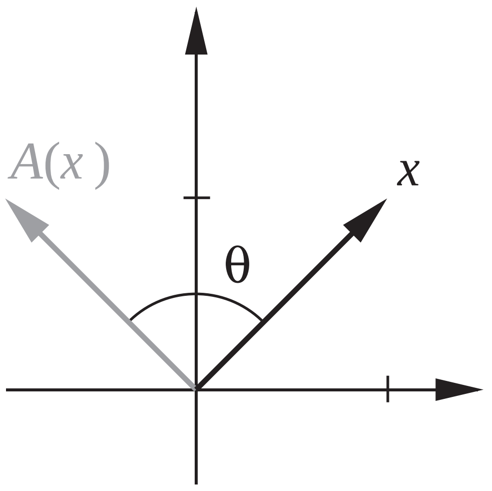
A transformation is **linear** if:

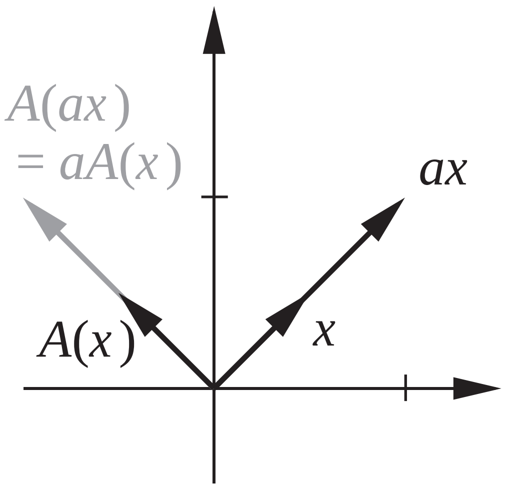
1. For all $x_1, x_2 \in X$, $A(x_1 + x_2) = A(x_1) + A(x_2)$,
2. For all $x \in X$, $a \in \Re$, $A(ax) = aA(x)$.

Example - Rotation

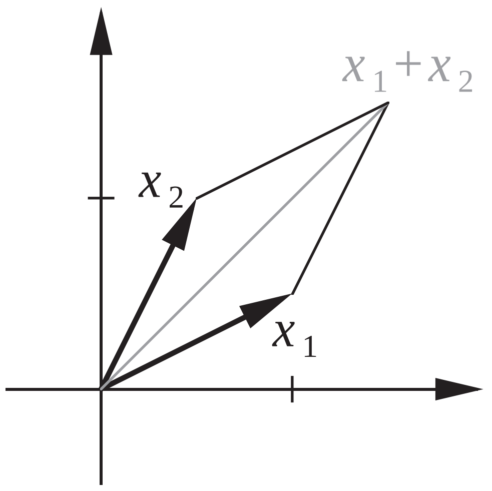
Is rotation linear?

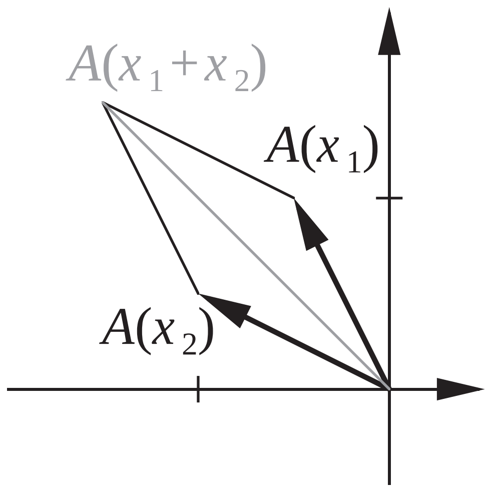
1.





2.





6

Matrix Representation - (1)

Any linear transformation between two finite-dimensional vector spaces can be represented by matrix multiplication.

Let $\{v_1, v_2, ..., v_n\}$ be a basis for X, and let $\{u_1, u_2, ..., u_m\}$ be a basis for Y.

$$x = \sum_{i=1}^{n} x_i v_i \qquad\qquad y = \sum_{i=1}^{m} y_i u_i$$

Let $A:X\rightarrow Y$

$$A(x) = y$$

$$A\left(\sum_{j=1}^{n} x_j v_j\right) = \sum_{i=1}^{m} y_i u_i$$

Matrix Representation - (2)

Since A is a linear operator,

$$\sum_{j=1}^{n} x_j A(v_j) = \sum_{i=1}^{m} y_i u_i$$

Since the u_i are a basis for Y,

$$A(v_j) = \sum_{i=1}^{m} a_{ij} u_i$$

(The coefficients a_{ij} will make up the matrix representation of the transformation.)

$$\sum_{j=1}^{n} x_\mathrm{j} \sum_{i=1}^{m} a_{ij} u_i = \sum_{i=1}^{m} y_i u_i$$

Matrix Representation - (3)

$$\sum_{i=1}^{m} u_i \sum_{j=1}^{n} a_{ij} x_j = \sum_{i=1}^{m} y_i u_i$$

$$\sum_{i=1}^{m} u_i \left(\sum_{j=1}^{n} a_{ij} x_j - y_i \right) = 0$$

Because the u_i are independent,

$$\sum_{j=1}^{n} a_{ij} x_j = y_i \quad \Rightarrow \quad \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

This is equivalent to matrix multiplication.

Summary

- A linear transformation can be represented by matrix multiplication.
- To find the matrix which represents the transformation we must transform each basis vector for the domain and then expand the result in terms of the basis vectors of the range.

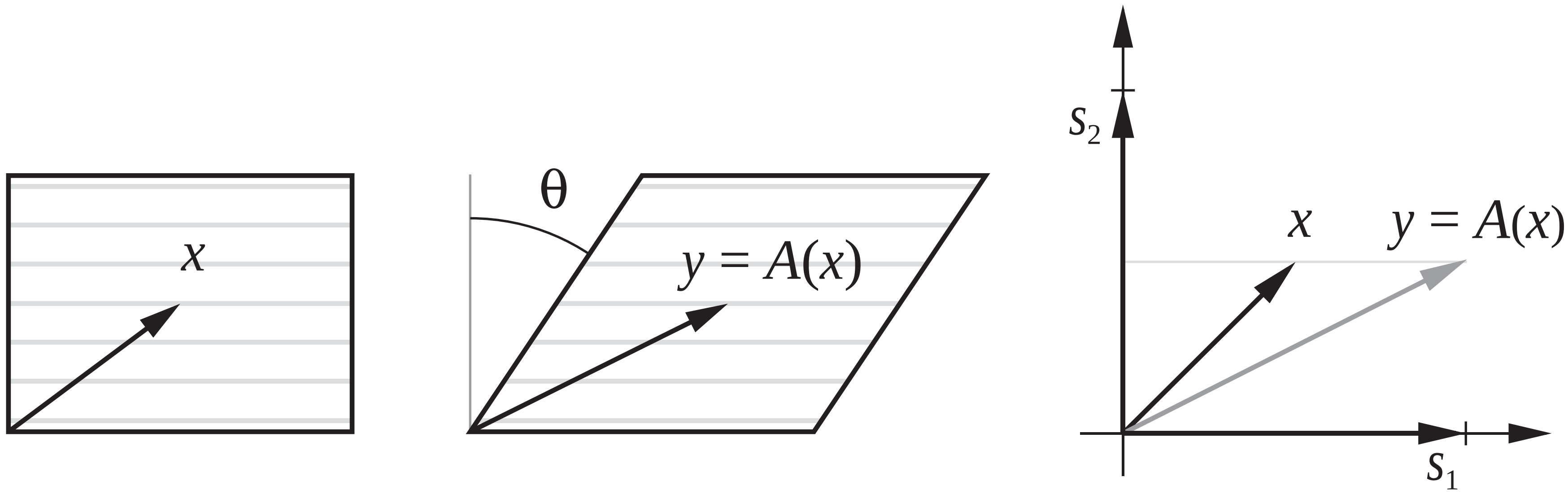
$$\boxed{A(v_j) = \sum_{i=1}^{m} a_{ij} u_i}$$

Each of these equations gives us one column of the matrix.

6

Example - (1)

Stand a deck of playing cards on edge so that you are looking at the deck sideways. Draw a vector *x* on the edge of the deck. Now "skew" the deck by an angle θ, as shown below, and note the new vector $y = A(x)$. What is the matrix of this transformation in terms of the standard basis set?



Example - (2)

To find the matrix we need to transform each of the basis vectors.

$$A(v_j) = \sum_{i=1}^{m} a_{ij} u_i$$

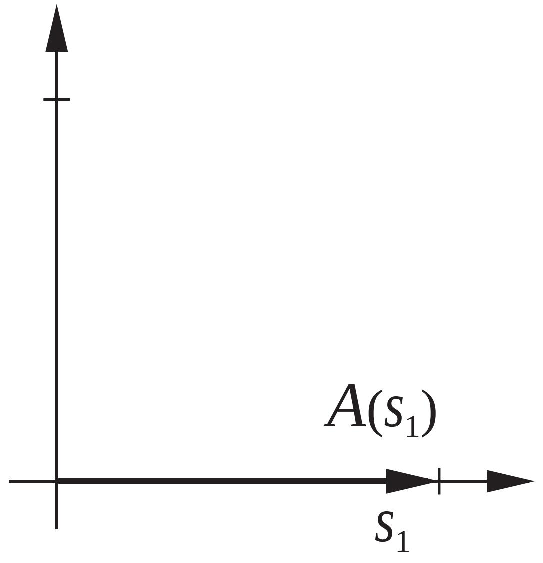
We will use the standard basis vectors for both the domain and the range.

$$A(s_j) = \sum_{i=1}^{2} a_{ij} s_i = a_{1j} s_1 + a_{2j} s_2$$

Example - (3)

We begin with s_1:

If we draw a line on the bottom card and then skew the deck, the line will not change.

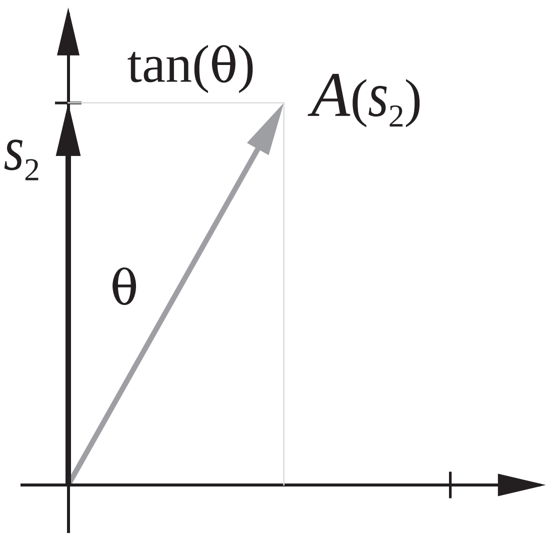


$$A(s_1) = 1s_1 + 0s_2 = \sum_{i=1}^{2} a_{i1}s_i = a_{11}s_1 + a_{21}s_2$$

This gives us the first column of the matrix.

Example - (4)

Next, we skew s_2:



$$A(s_2) = \tan(\theta)s_1 + 1s_2 = \sum_{i=1}^{2} a_{i2}s_i = a_{12}s_1 + a_{22}s_2$$

This gives us the second column of the matrix.

Example - (5)

The matrix of the transformation is:

$$\mathbf{A} = \begin{bmatrix} 1 & \tan(\theta) \\ 0 & 1 \end{bmatrix}$$

Change of Basis

Consider the linear transformation $A:X \rightarrow Y$. Let $\{v_1, v_2, ..., v_n\}$ be a basis for X, and let $\{u_1, u_2, ..., u_m\}$ be a basis for Y.

$$x = \sum_{i=1}^{n} x_i v_i \qquad y = \sum_{i=1}^{m} y_i u_i$$

$$A(x) = y$$

The matrix representation is:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\mathbf{Ax} = \mathbf{y}$$

New Basis Sets

Now let's consider different basis sets. Let $\{t_1, t_2, ..., t_n\}$ be a basis for X, and let $\{w_1, w_2, ..., w_m\}$ be a basis for Y.

$$x = \sum_{i=1}^{n} x'_i t_i \qquad y = \sum_{i=1}^{m} y'_i w_i$$

The new matrix representation is:

$$\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ a'_{21} & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & & \vdots \\ a'_{m1} & a'_{m2} & \cdots & a'_{mn} \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{bmatrix} = \begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_m \end{bmatrix}$$

$$\mathbf{A}'\mathbf{x}' = \mathbf{y}'$$

How are **A** and **A**' related?

Expand t_i in terms of the original basis vectors for X.

$$t_i = \sum_{j=1}^{n} t_{ji} v_j \qquad \mathbf{t}_i = \begin{bmatrix} t_{1i} \\ t_{2i} \\ \vdots \\ t_{ni} \end{bmatrix}$$

Expand w_i in terms of the original basis vectors for Y.

$$w_i = \sum_{j=1}^{m} w_{ji} u_j \qquad \mathbf{w}_i = \begin{bmatrix} w_{1i} \\ w_{2i} \\ \vdots \\ w_{mi} \end{bmatrix}$$

How are **A** and **A**' related?

$$\mathbf{B}_t = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \dots & \mathbf{t}_n \end{bmatrix} \qquad \mathbf{x} = x'_1\mathbf{t}_1 + x'_2\mathbf{t}_2 + \dots + x'_n\mathbf{t}_n = \mathbf{B}_t\mathbf{x}'$$

$$\mathbf{B}_w = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \dots & \mathbf{w}_m \end{bmatrix} \qquad \mathbf{y} = \mathbf{B}_w\mathbf{y}'$$

$$\mathbf{A}\mathbf{x} = \mathbf{y} \quad \Rightarrow \quad \mathbf{A}\mathbf{B}_t\mathbf{x}' = \mathbf{B}_w\mathbf{y}'$$

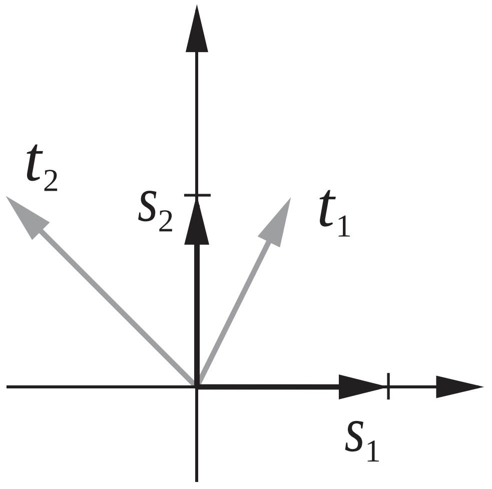
$$\begin{aligned} [\mathbf{B}_w^{-1}\mathbf{A}\mathbf{B}_t]\mathbf{x}' &= \mathbf{y}' \\ \mathbf{A}'\mathbf{x}' &= \mathbf{y}' \end{aligned} \quad \Rightarrow \quad \boxed{\mathbf{A}' = [\mathbf{B}_w^{-1}\mathbf{A}\mathbf{B}_t]}$$

Similarity Transform

6

Example - (1)

Take the skewing problem described previously, and find the new matrix representation using the basis set $\{s_1, s_2\}$.



$$t_1 = 0.5s_1 + s_2$$

$$t_2 = -s_1 + s_2$$

$$\mathbf{t}_1 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \qquad \mathbf{t}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\Rightarrow \quad \mathbf{B}_t = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 \end{bmatrix} = \begin{bmatrix} 0.5 & -1 \\ 1 & 1 \end{bmatrix} \qquad \mathbf{B}_w = \mathbf{B}_t = \begin{bmatrix} 0.5 & -1 \\ 1 & 1 \end{bmatrix}$$

(Same basis for domain and range.)

Example - (2)

$$\mathbf{A'} = [\mathbf{B}_w^{-1}\mathbf{A}\mathbf{B}_t] = \begin{bmatrix} 2/3 & 2/3 \\ -2/3 & 1/3 \end{bmatrix} \begin{bmatrix} 1 & \tan\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & -1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{A'} = \begin{bmatrix} (2/3)\tan\theta + 1 & (2/3)\tan\theta \\ (-2/3)\tan\theta & (-2/3)\tan\theta + 1 \end{bmatrix}$$

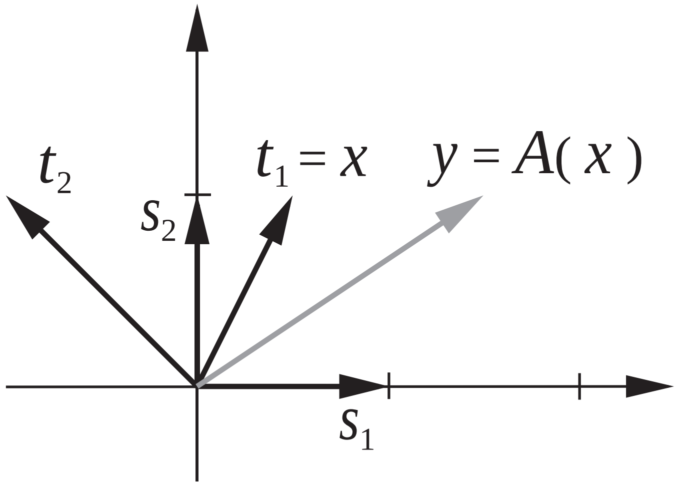
For $\theta = 45°$:

$$\mathbf{A'} = \begin{bmatrix} 5/3 & 2/3 \\ -2/3 & 1/3 \end{bmatrix} \qquad \mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Example - (3)

Try a test vector: $\mathbf{x} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$ $\mathbf{x}' = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$\mathbf{y} = \mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 1 \end{bmatrix} \qquad \mathbf{y}' = \mathbf{A}'\mathbf{x}' = \begin{bmatrix} 5/3 & 2/3 \\ -2/3 & 1/3 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 5/3 \\ -2/3 \end{bmatrix}$$



Check using reciprocal basis vectors:

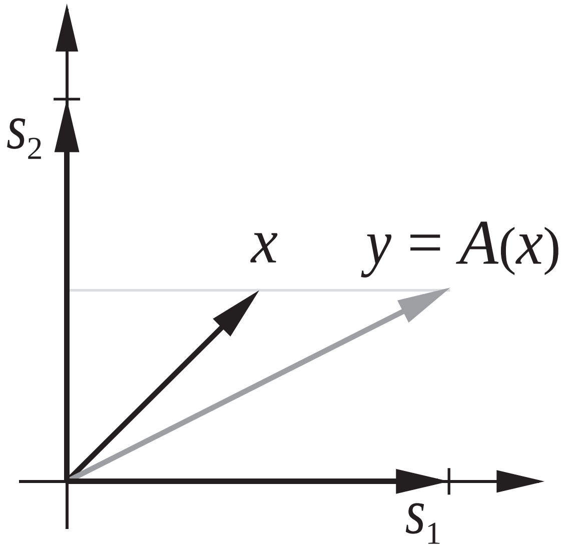
$$\mathbf{y}' = \mathbf{B}^{-1}\mathbf{y} = \begin{bmatrix} 0.5 & -1 \\ 1 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 1.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 2/3 & 2/3 \\ -2/3 & 1/3 \end{bmatrix}\begin{bmatrix} 1.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 5/3 \\ -2/3 \end{bmatrix}$$

Eigenvalues and Eigenvectors

Let $A:X \rightarrow X$ be a linear transformation. Those vectors $z \in X$, which are not equal to zero, and those scalars λ which satisfy

$$A(z) = \lambda z$$

are called eigenvectors and eigenvalues, respectively.



Can you find an eigenvector for this transformation?

Computing the Eigenvalues

$$\mathbf{Az} = \lambda\mathbf{z}$$

$$[\mathbf{A} - \lambda\mathbf{I}]\mathbf{z} = \mathbf{0} \quad \Rightarrow \quad |[\mathbf{A} - \lambda\mathbf{I}]| = 0$$

Skewing example (45°):

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad \left|\begin{bmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{bmatrix}\right| = 0 \qquad (1-\lambda)^2 = 0 \qquad \begin{matrix} \lambda_1 = 1 \\ \lambda_2 = 1 \end{matrix}$$

$$\begin{bmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{bmatrix}\mathbf{z} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\mathbf{z}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} z_{11} \\ z_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad z_{21} = 0 \qquad \mathbf{z}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

For this transformation there is only one eigenvector.

Diagonalization

Perform a change of basis (similarity transformation) using the eigenvectors as the basis vectors. If the eigenvalues are distinct, the new matrix will be diagonal.

$$\mathbf{B} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_n \end{bmatrix}$$

$\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$ Eigenvectors

$\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ Eigenvalues

$$[\mathbf{B}^{-1}\mathbf{A}\mathbf{B}] = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

Example

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\left| \begin{bmatrix} 1-\lambda & 1 \\ 1 & 1-\lambda \end{bmatrix} \right| = 0 \qquad \lambda^2 - 2\lambda = (\lambda)(\lambda-2) = 0 \qquad \begin{matrix} \lambda_1 = 0 \\ \lambda_2 = 2 \end{matrix} \qquad \begin{bmatrix} 1-\lambda & 1 \\ 1 & 1-\lambda \end{bmatrix} \mathbf{z} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\lambda_1 = 0 \quad \Longrightarrow \quad \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \mathbf{z}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} z_{11} \\ z_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad z_{21} = -z_{11} \qquad \mathbf{z}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$\lambda_2 = 2 \quad \Longrightarrow \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \mathbf{z}_1 = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} z_{12} \\ z_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad z_{22} = z_{12} \qquad \mathbf{z}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Diagonal Form: $\mathbf{A}' = [\mathbf{B}^{-1}\mathbf{A}\mathbf{B}] = \begin{bmatrix} 1/2 & -1/2 \\ 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$

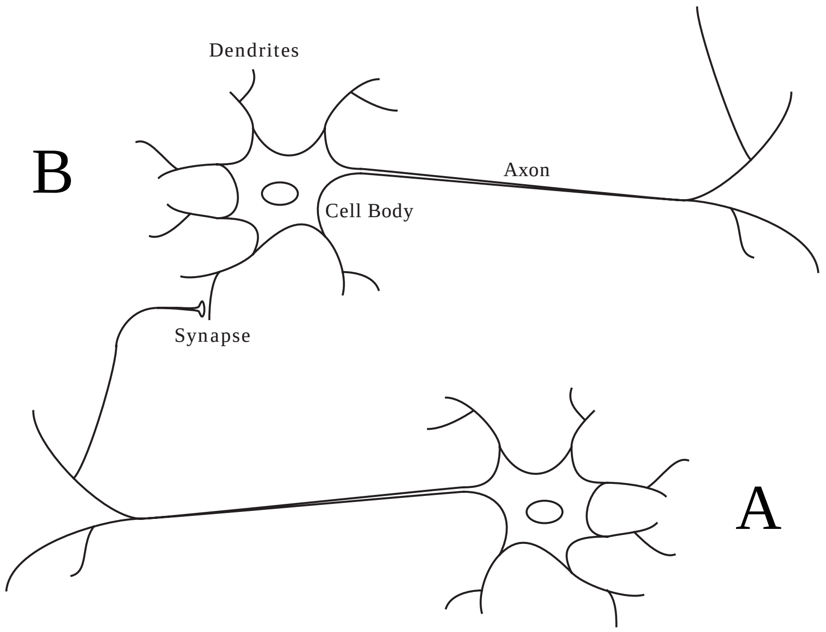
Supervised Hebbian Learning

7

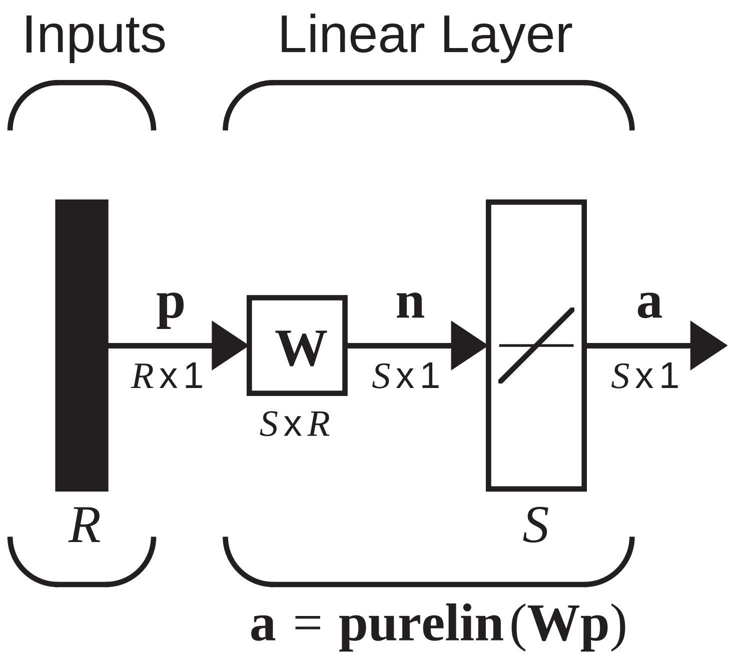
Hebb’s Postulate

“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A’s efficiency, as one of the cells firing B, is increased.”

D. O. Hebb, 1949



Linear Associator



$$\mathbf{a} = \mathbf{W}\mathbf{p} \qquad a_i = \sum_{j=1}^{R} w_{ij} p_j$$

Training Set:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Hebb Rule

$$w_{ij}^{new} = w_{ij}^{old} + \alpha \; \underbrace{f_i(a_{iq})}_{\text{Postsynaptic Signal}} \underbrace{g_j(p_{jq})}_{\text{Presynaptic Signal}}$$

Simplified Form:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{jq}$$

Supervised Form:

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq} p_{jq}$$

Matrix Form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T$$

Batch Operation

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T = \sum_{q=1}^{Q} \mathbf{t}_q\mathbf{p}_q^T$$

(Zero Initial Weights)

Matrix Form:

$$\mathbf{W} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \ldots & \mathbf{t}_Q \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \ldots & \mathbf{t}_Q \end{bmatrix}$$

Performance Analysis

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \left(\sum_{q=1}^{Q} \mathbf{t}_q \mathbf{p}_q^T\right)\mathbf{p}_k = \sum_{q=1}^{Q} \mathbf{t}_q (\mathbf{p}_q^T \mathbf{p}_k)$$

Case I, input patterns are orthogonal.

$$(\mathbf{p}_q^T \mathbf{p}_k) = 1 \qquad q = k$$

$$= 0 \qquad q \neq k$$

Therefore the network output equals the target:

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k$$

Case II, input patterns are normalized, but not orthogonal.

$$\mathbf{a} = \mathbf{W}\mathbf{p}_k = \mathbf{t}_k + \boxed{\sum_{q \neq k} \mathbf{t}_q (\mathbf{p}_q^T \mathbf{p}_k)}$$

Error

Example

Banana Apple Normalized Prototype Patterns

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \qquad \left\{ \mathbf{p}_1 = \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

Weight Matrix (Hebb Rule):

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -0.5774 & 0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix} = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix}$$

Tests:

Banana

$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} -0.6668 \end{bmatrix}$$

Apple

$$\mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 0 & 1.1548 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} 0.6668 \end{bmatrix}$$

Pseudoinverse Rule - (1)

Performance Index: $\mathbf{W}\mathbf{p}_q = \mathbf{t}_q \qquad q = 1, 2, \ldots, Q$

$$F(\mathbf{W}) = \sum_{q=1}^{Q} \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2$$

Matrix Form:

$$\mathbf{W}\mathbf{P} = \mathbf{T}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix}$$

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

$$\|\mathbf{E}\|^2 = \sum_i \sum_j e_{ij}^2$$

Pseudoinverse Rule - (2)

$$\mathbf{W}\mathbf{P} = \mathbf{T}$$

Minimize:

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

If an inverse exists for **P**, *F*(**W**) can be made zero:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{-1}$$

When an inverse does not exist *F*(**W**) can be minimized using the pseudoinverse:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+}$$

$$\mathbf{P}^{+} = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$$

Relationship to the Hebb Rule

Hebb Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T$$

Pseudoinverse Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$$

If the prototype patterns are orthonormal:

$$\mathbf{P}^T\mathbf{P} = \mathbf{I}$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T = \mathbf{P}^T$$

Example

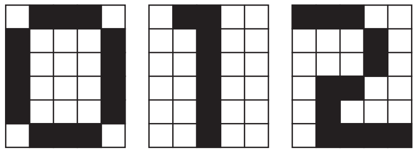
$$\left\{\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix}\right\} \qquad \mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \begin{bmatrix} -1 & 1 \end{bmatrix}\left(\begin{bmatrix} -1 & 1 \\ 1 & 1 \\ -1 & -1 \end{bmatrix}\right)^{+}$$

$$\mathbf{P}^{+} = (\mathbf{P}^{T}\mathbf{P})^{-1}\mathbf{P}^{T} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}^{-1}\begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.25 & -0.25 \\ 0.5 & 0.25 & -0.25 \end{bmatrix}$$

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{+} = \begin{bmatrix} -1 & 1 \end{bmatrix}\begin{bmatrix} -0.5 & 0.25 & -0.25 \\ 0.5 & 0.25 & -0.25 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

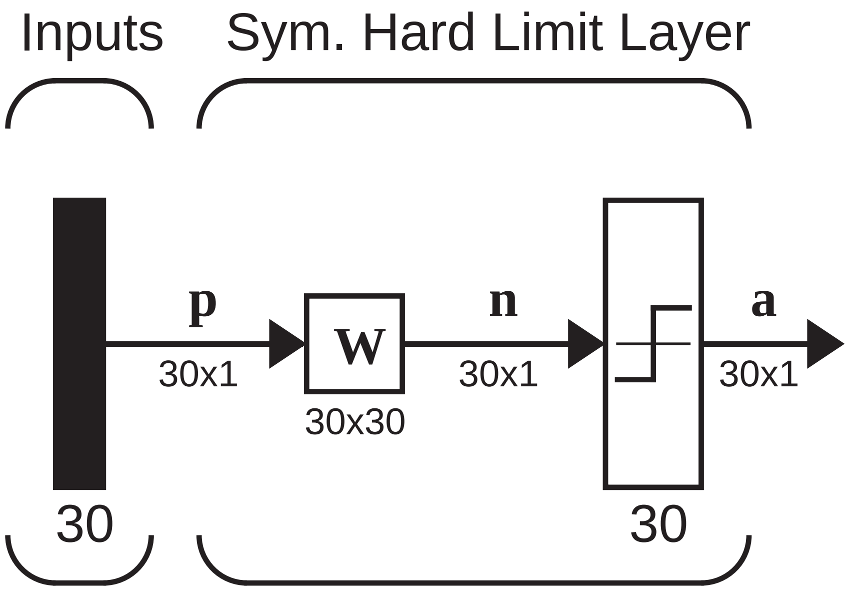
$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix} \qquad \mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}$$

Autoassociative Memory



$\mathbf{p}_1, \mathbf{t}_1 \quad \mathbf{p}_2, \mathbf{t}_2 \quad \mathbf{p}_3, \mathbf{t}_3$

$$\mathbf{p}_1 = \begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & \dots & 1 & -1 \end{bmatrix}^T$$

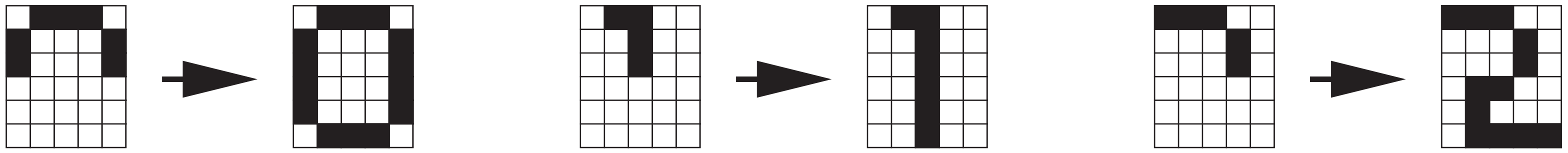


$$\mathbf{a} = \mathbf{hardlims}(\mathbf{Wp})$$

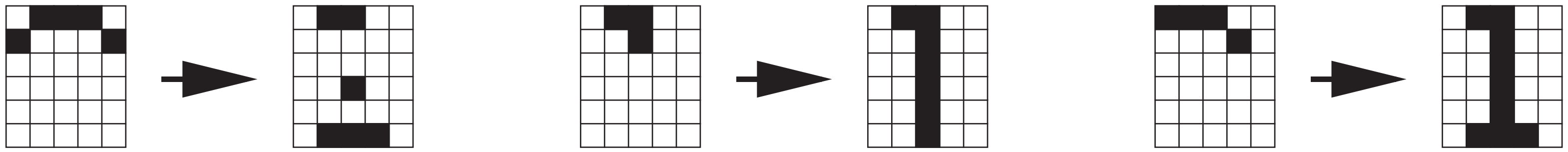
$$\mathbf{W} = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \mathbf{p}_3\mathbf{p}_3^T$$

Tests

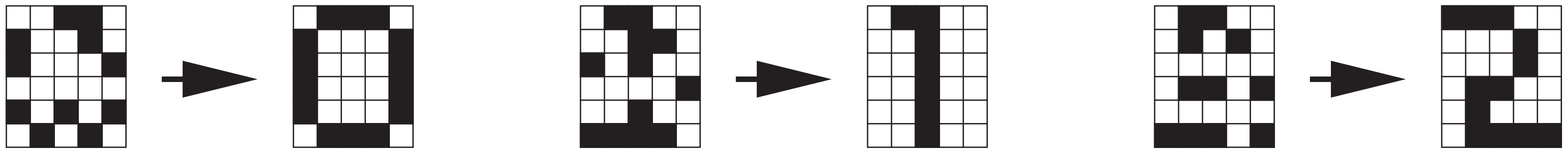
50% Occluded



67% Occluded



Noisy Patterns (7 pixels)



7

Variations of Hebbian Learning

Basic Rule: $\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q \mathbf{p}_q^T$

Learning Rate: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

Smoothing: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T - \gamma \mathbf{W}^{old} = (1 - \gamma)\mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

Delta Rule: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T$

Unsupervised: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{a}_q \mathbf{p}_q^T$

Performance Surfaces

Taylor Series Expansion

$$F(x) = F(x^*) + \frac{d}{dx}F(x)\bigg|_{x = x^*}(x - x^*)$$

$$+ \frac{1}{2}\frac{d^2}{dx^2}F(x)\bigg|_{x = x^*}(x - x^*)^2 + \cdots$$

$$+ \frac{1}{n!}\frac{d^n}{dx^n}F(x)\bigg|_{x = x^*}(x - x^*)^n + \cdots$$

Example

$$F(x) = e^{-x}$$

Taylor series of $F(x)$ about $x^* = 0$:

$$F(x) = e^{-x} = e^{-0} - e^{-0}(x-0) + \frac{1}{2}e^{-0}(x-0)^2 - \frac{1}{6}e^{-0}(x-0)^3 + \ldots$$

$$F(x) = 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \ldots$$

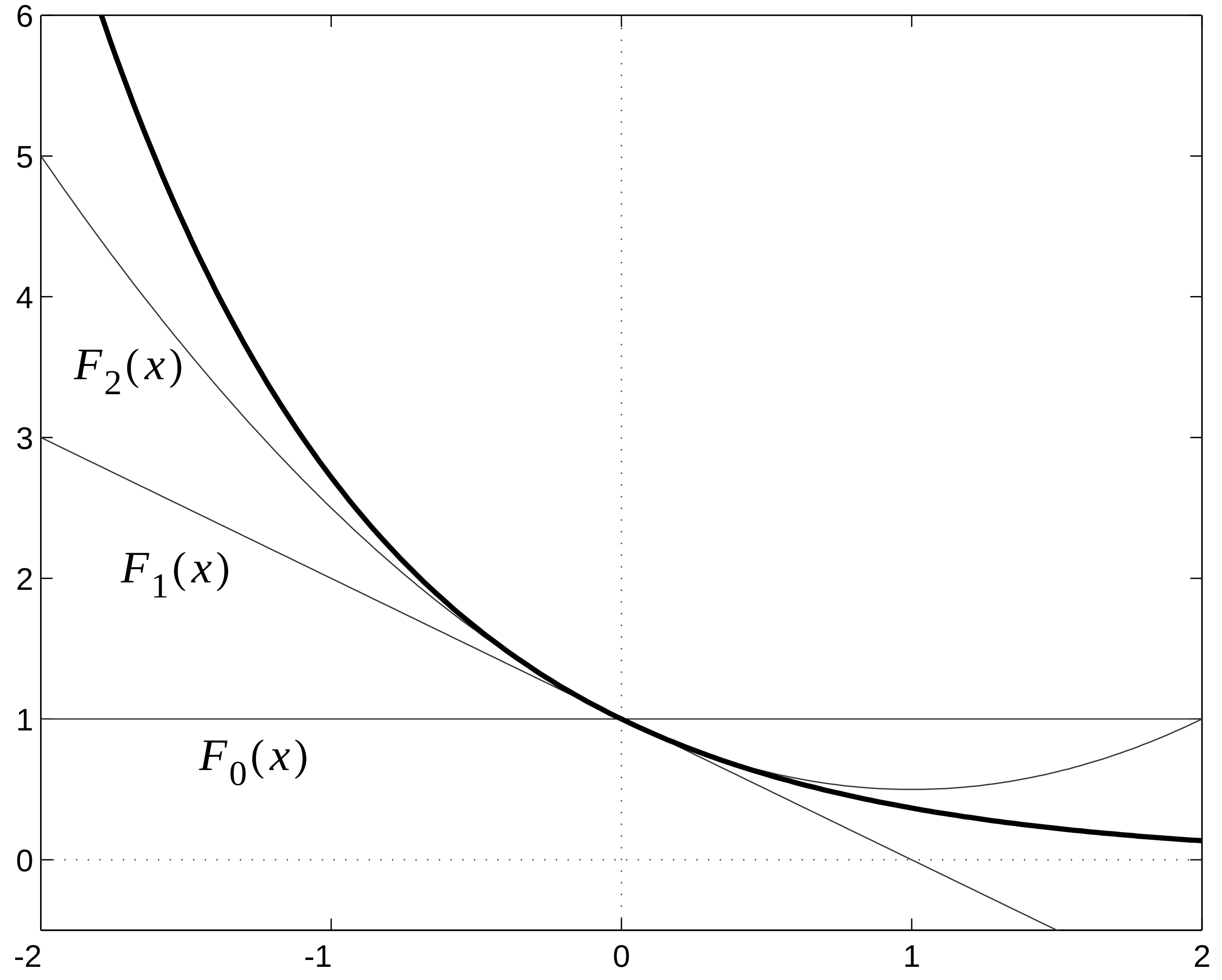
Taylor series approximations:

$$F(x) \approx F_0(x) = 1$$

$$F(x) \approx F_1(x) = 1 - x$$

$$F(x) \approx F_2(x) = 1 - x + \frac{1}{2}x^2$$

Plot of Approximations



Vector Case

$$F(\mathbf{x}) = F(x_1, x_2, \dots, x_n)$$

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \frac{\partial}{\partial x_1}F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(x_1 - x_1^*) + \frac{\partial}{\partial x_2}F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(x_2 - x_2^*)$$

$$+ \cdots + \frac{\partial}{\partial x_n}F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(x_n - x_n^*) + \frac{1}{2}\frac{\partial^2}{\partial x_1^2}F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(x_1 - x_1^*)^2$$

$$+ \frac{1}{2}\frac{\partial^2}{\partial x_1 \partial x_2}F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(x_1 - x_1^*)(x_2 - x_2^*) + \cdots$$

Matrix Form

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x}-\mathbf{x}^*)$$

$$+ \frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^T \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*) + \cdots$$

Gradient

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n}F(\mathbf{x}) \end{bmatrix}$$

Hessian

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2}F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2}F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_{\mathrm{n}}}F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1}F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2}F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_2 \partial x_{\mathrm{n}}}F(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2}{\partial x_{\mathrm{n}} \partial x_1}F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2}F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_n^2}F(\mathbf{x}) \end{bmatrix}$$

Directional Derivatives

First derivative (slope) of $F(\mathbf{x})$ along x_i axis: $\partial F(\mathbf{x}) / \partial x_i$

(ith element of gradient)

Second derivative (curvature) of $F(\mathbf{x})$ along x_i axis: $\partial^2 F(\mathbf{x}) / \partial x_i^2$

(i,i element of Hessian)

First derivative (slope) of $F(\mathbf{x})$ along vector $\mathbf{p}$: $$\frac{\mathbf{p}^T \nabla F(\mathbf{x})}{\|\mathbf{p}\|}$$

Second derivative (curvature) of $F(\mathbf{x})$ along vector $\mathbf{p}$: $$\frac{\mathbf{p}^T \nabla^2 F(\mathbf{x}) \mathbf{p}}{\|\mathbf{p}\|^2}$$

Example

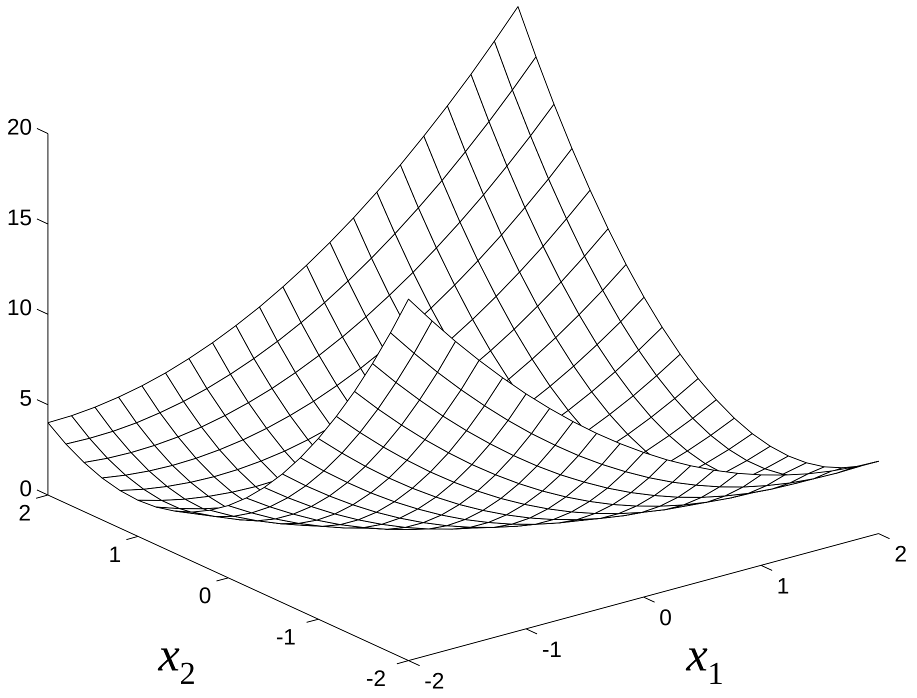
$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2$$

$$\mathbf{x}^* = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

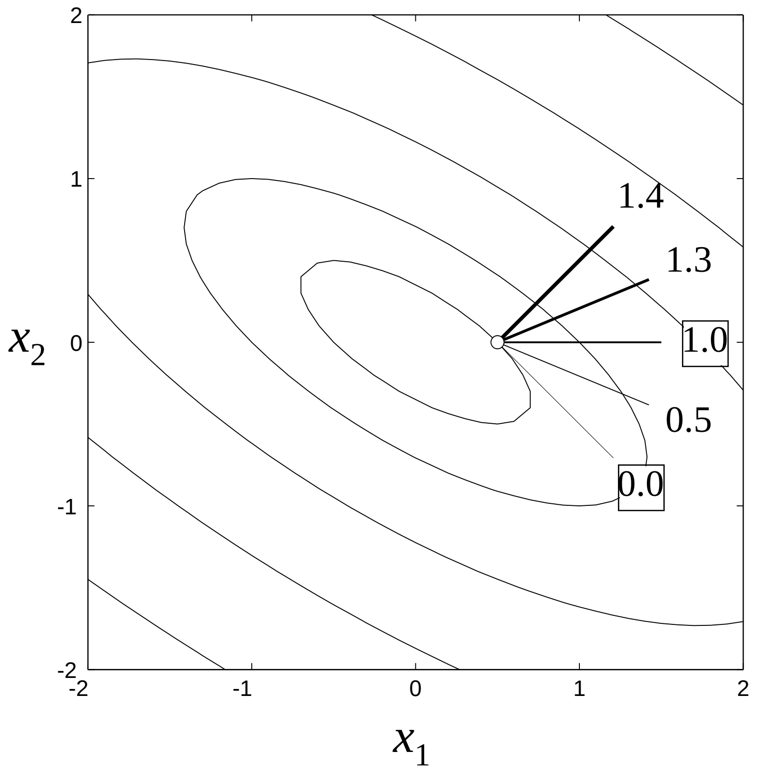
$$\nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} = \left.\begin{bmatrix} \dfrac{\partial}{\partial x_1}F(\mathbf{x}) \\ \dfrac{\partial}{\partial x_2}F(\mathbf{x}) \end{bmatrix}\right|_{\mathbf{x}=\mathbf{x}^*} = \left.\begin{bmatrix} 2x_1 + 2x_2 \\ 2x_1 + 4x_2 \end{bmatrix}\right|_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\frac{\mathbf{p}^T \nabla F(\mathbf{x})}{\|\mathbf{p}\|} = \frac{\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}}{\left\| \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\|} = \frac{\begin{bmatrix} 0 \end{bmatrix}}{\sqrt{2}} = 0$$

Plots



Directional Derivatives



Minima

Strong Minimum

The point $\mathbf{x}^*$ is a strong minimum of $F(\mathbf{x})$ if a scalar $\delta > 0$ exists, such that $F(\mathbf{x}^*) < F(\mathbf{x}^* + \Delta\mathbf{x})$ for all $\Delta\mathbf{x}$ such that $\delta > \|\Delta\mathbf{x}\| > 0$.

Global Minimum

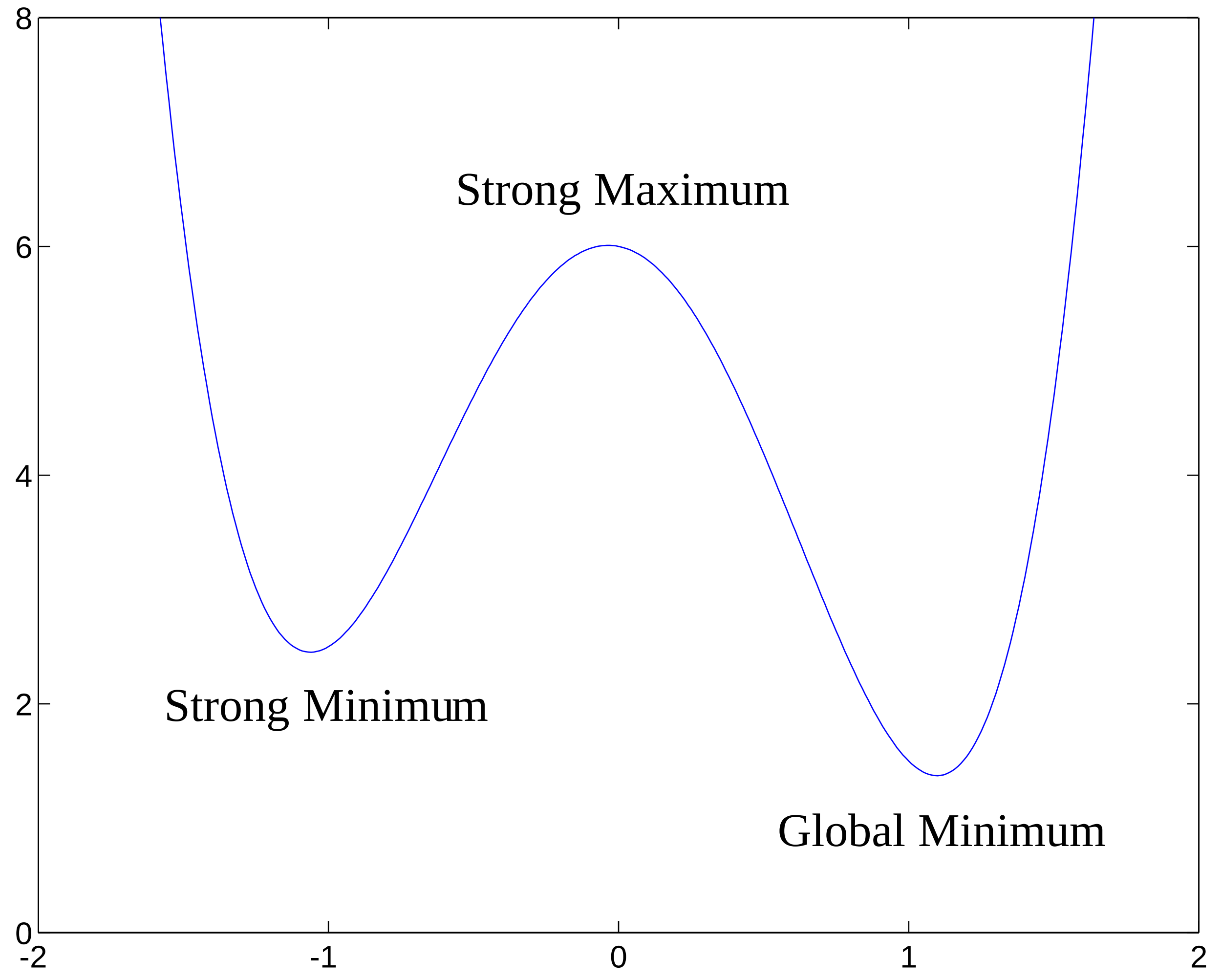
The point $\mathbf{x}^*$ is a unique global minimum of $F(\mathbf{x})$ if $F(\mathbf{x}^*) < F(\mathbf{x}^* + \Delta\mathbf{x})$ for all $\Delta\mathbf{x} \neq 0$.

Weak Minimum

The point $\mathbf{x}^*$ is a weak minimum of $F(\mathbf{x})$ if it is not a strong minimum, and a scalar $\delta > 0$ exists, such that $F(\mathbf{x}^*) \leq F(\mathbf{x}^* + \Delta\mathbf{x})$ for all $\Delta\mathbf{x}$ such that $\delta > \|\Delta\mathbf{x}\| > 0$.

Scalar Example

$$F(x) = 3x^4 - 7x^2 - \frac{1}{2}x + 6$$

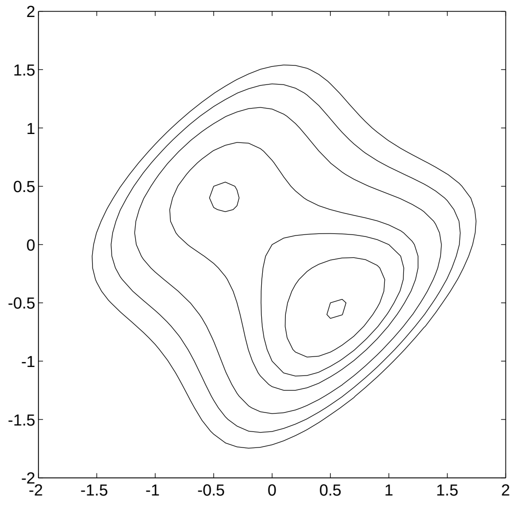
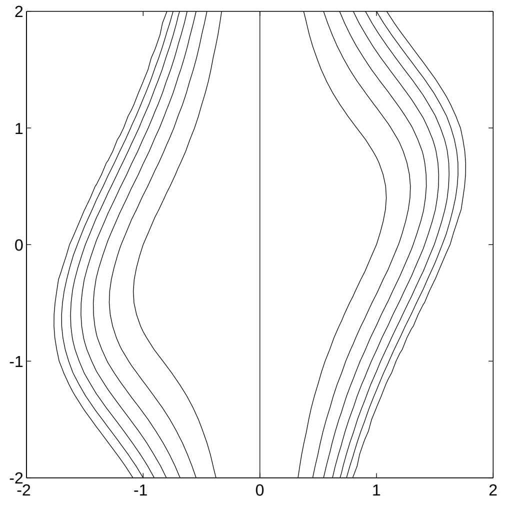
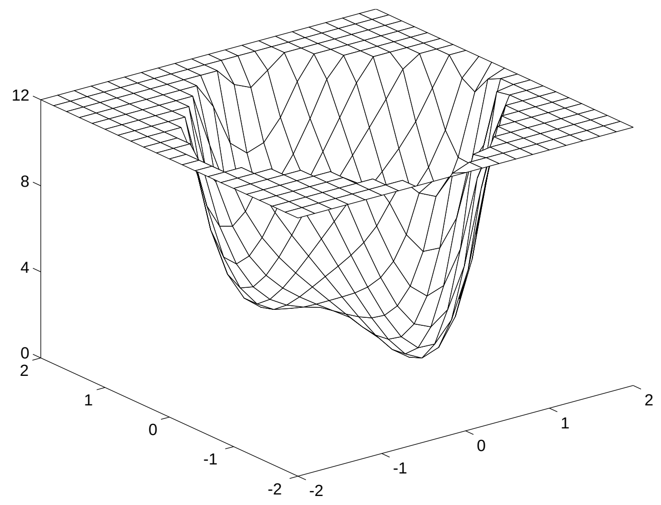
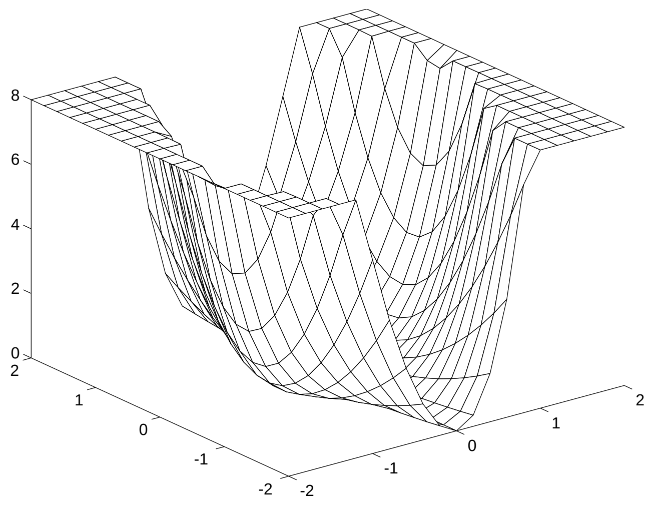


8

Vector Example

$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$

$F(\mathbf{x}) = (x_1^2 - 1.5x_1x_2 + 2x_2^2)x_1^2$

First-Order Optimality Condition

$$F(\mathbf{x}) = F(\mathbf{x}^* + \Delta\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x} + \cdots$$

$$\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$$

For small $\Delta\mathbf{x}$:

$$F(\mathbf{x}^* + \Delta\mathbf{x}) \cong F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x}$$

If **x*** is a minimum, this implies:

$$\nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x} \geq 0$$

If $\nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x} > 0$ then $F(\mathbf{x}^* - \Delta\mathbf{x}) \cong F(\mathbf{x}^*) - \nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x} < F(\mathbf{x}^*)$

But this would imply that **x*** is not a minimum. Therefore $\nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} \Delta\mathbf{x} = 0$

Since this must be true for every $\Delta\mathbf{x}$, $\boxed{\nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}}$

Second-Order Condition

If the first-order condition is satisfied (zero gradient), then

$$F(\mathbf{x}^* + \Delta\mathbf{x}) = F(\mathbf{x}^*) + \frac{1}{2}\Delta\mathbf{x}^T \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}^*} \Delta\mathbf{x} + \cdots$$

A strong minimum will exist at $\mathbf{x}^*$ if $\Delta\mathbf{x}^T \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}^*} \Delta\mathbf{x} > 0$ for any $\Delta\mathbf{x} \neq \mathbf{0}$.

Therefore the Hessian matrix must be positive definite. A matrix **A** is positive definite if:

$$\boxed{\mathbf{z}^T \mathbf{A} \mathbf{z} > 0} \qquad \text{for any } \mathbf{z} \neq 0.$$

This is a **sufficient** condition for optimality.

A **necessary** condition is that the Hessian matrix be positive semidefinite. A matrix A is positive semidefinite if:

$$\boxed{\mathbf{z}^T \mathbf{A} \mathbf{z} \geq 0} \qquad \text{for any } \mathbf{z}.$$

Example

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} = \mathbf{0} \quad \Rightarrow \quad \mathbf{x}^* = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

(Not a function of **x** in this case.)

To test the definiteness, check the eigenvalues of the Hessian. If the eigenvalues are all greater than zero, the Hessian is positive definite.

$$\left|\nabla^2 F(\mathbf{x}) - \lambda \mathbf{I}\right| = \left|\begin{bmatrix} 2-\lambda & 2 \\ 2 & 4-\lambda \end{bmatrix}\right| = \lambda^2 - 6\lambda + 4 = (\lambda - 0.76)(\lambda - 5.24)$$

$\lambda = 0.76, 5.24$ Both eigenvalues are positive, therefore <u>strong minimum</u>.

Quadratic Functions

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c \qquad \text{(Symmetric } \mathbf{A}\text{)}$$

Gradient and Hessian:

Useful properties of gradients:

$$\nabla(\mathbf{h}^T\mathbf{x}) = \nabla(\mathbf{x}^T\mathbf{h}) = \mathbf{h}$$

$$\nabla\mathbf{x}^T\mathbf{Q}\mathbf{x} = \mathbf{Q}\mathbf{x} + \mathbf{Q}^T\mathbf{x} = 2\mathbf{Q}\mathbf{x} \quad \text{(for symmetric } \mathbf{Q}\text{)}$$

Gradient of Quadratic Function:

$$\boxed{\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}}$$

Hessian of Quadratic Function:

$$\boxed{\nabla^2 F(\mathbf{x}) = \mathbf{A}}$$

Eigensystem of the Hessian

Consider a quadratic function which has a stationary point at the origin, and whose value there is zero.

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$$

Perform a similarity transform on the Hessian matrix, using the eigenvalues as the new basis vectors.

$$\mathbf{B} = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_n \end{bmatrix}$$

Since the Hessian matrix is symmetric, its eigenvectors are orthogonal.

$$\mathbf{B}^{-1} = \mathbf{B}^T$$

$$\mathbf{A}' = [\mathbf{B}^T\mathbf{A}\mathbf{B}] = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix} = \Lambda \qquad \mathbf{A} = \mathbf{B}\Lambda\mathbf{B}^T$$

Second Directional Derivative

$$\frac{\mathbf{p}^T \nabla^2 F(\mathbf{x}) \mathbf{p}}{\|\mathbf{p}\|^2} = \frac{\mathbf{p}^T \mathbf{A} \mathbf{p}}{\|\mathbf{p}\|^2}$$

Represent **p** with respect to the eigenvectors (new basis):

$$\mathbf{p} = \mathbf{B}\mathbf{c}$$

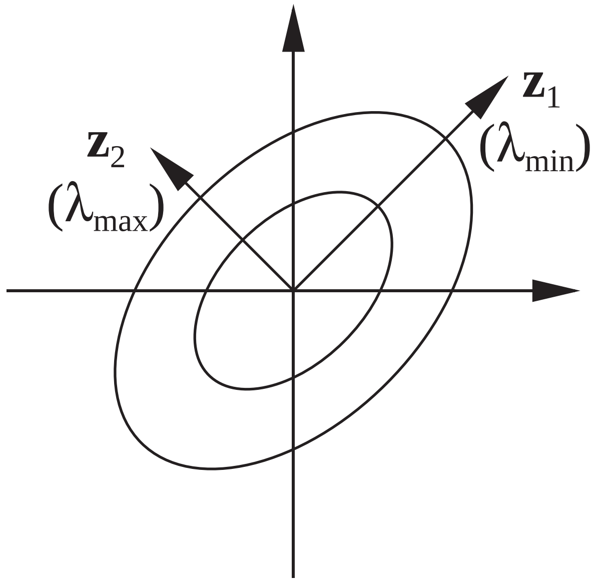
$$\frac{\mathbf{p}^T \mathbf{A} \mathbf{p}}{\|\mathbf{p}\|^2} = \frac{\mathbf{c}^T \mathbf{B}^T (\mathbf{B} \mathbf{\Lambda} \mathbf{B}^T) \mathbf{B} \mathbf{c}}{\mathbf{c}^T \mathbf{B}^T \mathbf{B} \mathbf{c}} = \frac{\mathbf{c}^T \mathbf{\Lambda} \mathbf{c}}{\mathbf{c}^T \mathbf{c}} = \frac{\sum_{i=1}^{n} \lambda_i c_i^2}{\sum_{i=1}^{n} c_i^2}$$

$$\lambda_{min} \le \frac{\mathbf{p}^T \mathbf{A} \mathbf{p}}{\|\mathbf{p}\|^2} \le \lambda_{max}$$

Eigenvector (Largest Eigenvalue)

$$\mathbf{p} = \mathbf{z}_{max} \qquad \mathbf{c} = \mathbf{B}^T\mathbf{p} = \mathbf{B}^T\mathbf{z}_{max} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\frac{\mathbf{z}_{max}{}^T\mathbf{A}\mathbf{z}_{max}}{\|\mathbf{z}_{max}\|^2} = \frac{\sum_{i=1}^{n} \lambda_i c_i^2}{\sum_{i=1}^{n} c_i^2} = \lambda_{max}$$



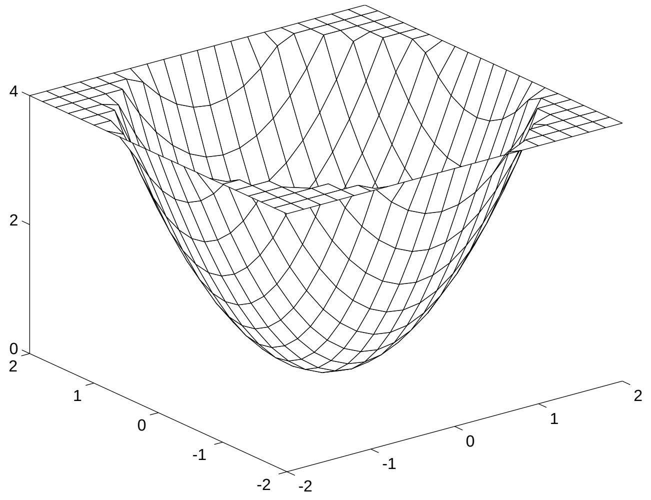
The eigenvalues represent curvature (second derivatives) along the eigenvectors (the principal axes).

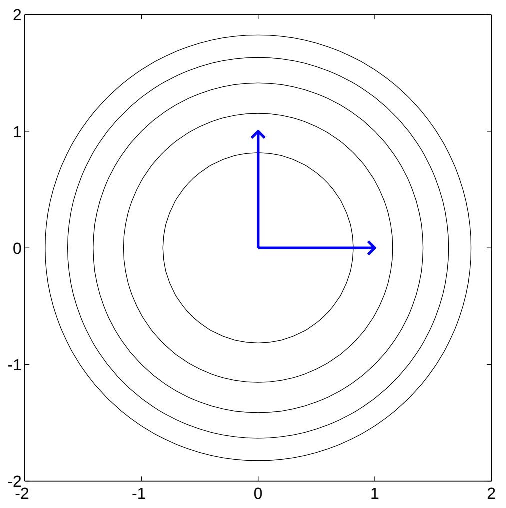
Circular Hollow

$$F(\mathbf{x}) = x_1^2 + x_2^2 = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \mathbf{x}$$

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad \lambda_1 = 2 \qquad \mathbf{z}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \lambda_2 = 2 \qquad \mathbf{z}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(Any two independent vectors in the plane would work.)

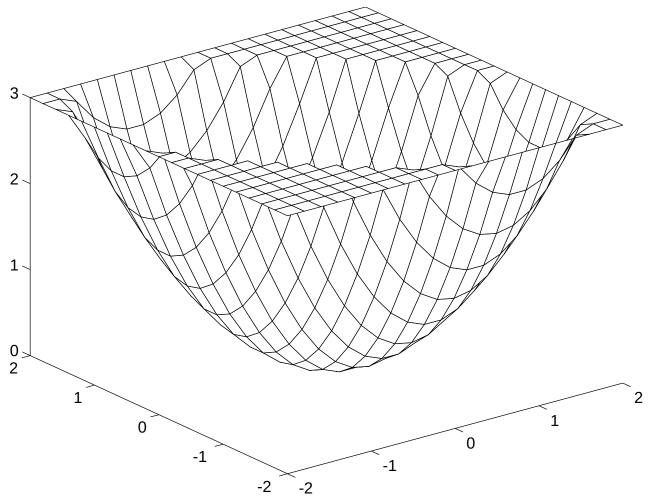


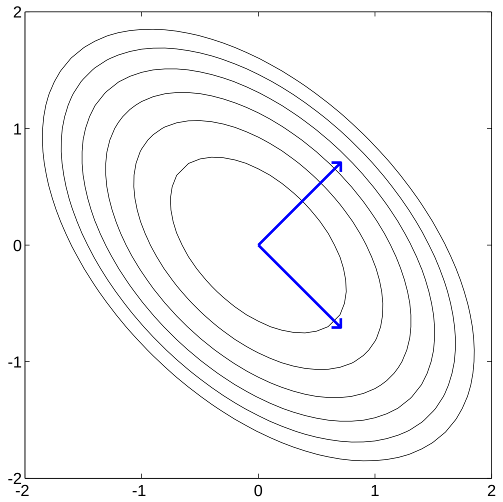


Elliptical Hollow

$$F(\mathbf{x}) = x_1^2 + x_1 x_2 + x_2^2 = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{x}$$

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \qquad \lambda_1 = 1 \qquad \mathbf{z}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad \lambda_2 = 3 \qquad \mathbf{z}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



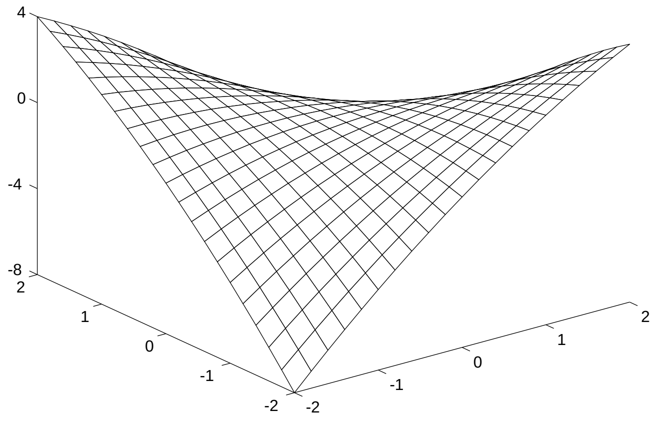


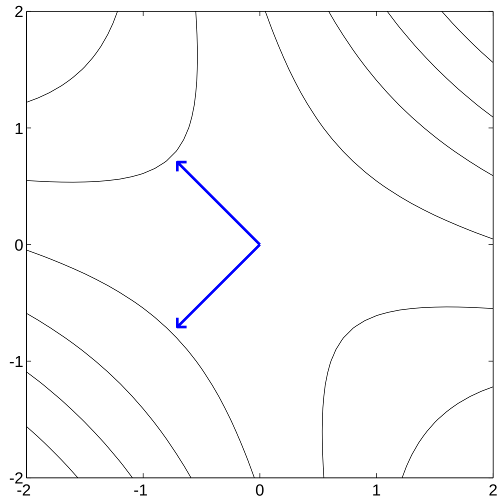
8

Elongated Saddle

$$F(\mathbf{x}) = -\frac{1}{4}x_1^2 - \frac{3}{2}x_1x_2 - \frac{1}{4}x_2^2 = \frac{1}{2}\mathbf{x}^T\begin{bmatrix}-0.5 & -1.5\\ -1.5 & -0.5\end{bmatrix}\mathbf{x}$$

$$\nabla^2F(\mathbf{x}) = \begin{bmatrix}-0.5 & -1.5\\ -1.5 & -0.5\end{bmatrix} \qquad \lambda_1 = 1 \qquad \mathbf{z}_1 = \begin{bmatrix}-1\\ 1\end{bmatrix} \qquad \lambda_2 = -2 \qquad \mathbf{z}_2 = \begin{bmatrix}-1\\ -1\end{bmatrix}$$

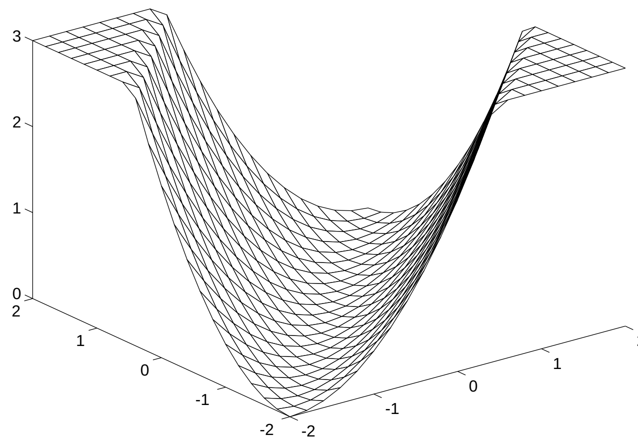


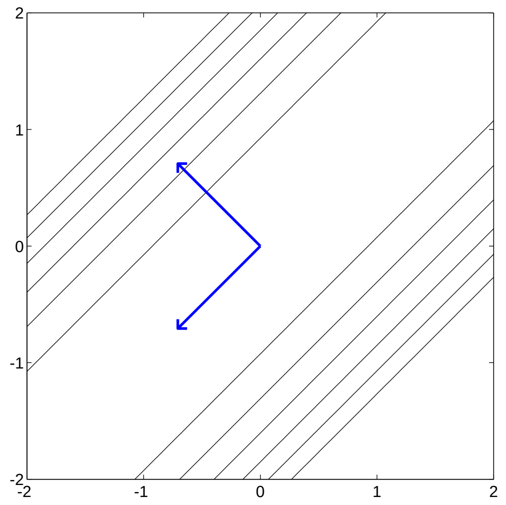


Stationary Valley

$$F(\mathbf{x}) = \frac{1}{2}x_1^2 - x_1 x_2 + \frac{1}{2}x_2^2 = \frac{1}{2}\mathbf{x}^T\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\mathbf{x}$$

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad \lambda_1 = 1 \qquad \mathbf{z}_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \lambda_2 = 0 \qquad \mathbf{z}_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$





Quadratic Function Summary

- If the eigenvalues of the Hessian matrix are all positive, the function will have a single strong minimum.
- If the eigenvalues are all negative, the function will have a single strong maximum.
- If some eigenvalues are positive and other eigenvalues are negative, the function will have a single saddle point.
- If the eigenvalues are all nonnegative, but some eigenvalues are zero, then the function will either have a weak minimum or will have no stationary point.
- If the eigenvalues are all nonpositive, but some eigenvalues are zero, then the function will either have a weak maximum or will have no stationary point.

Stationary Point: $\mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{d}$

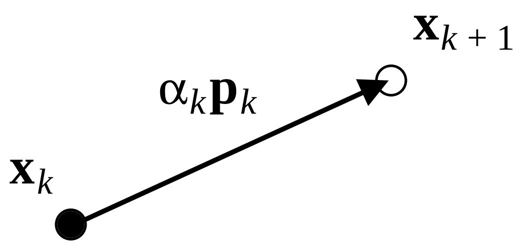
Performance Optimization

Basic Optimization Algorithm

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k}$$

or

$$\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$



$\mathbf{p}_k$ - Search Direction

α_k - Learning Rate

Steepest Descent

Choose the next step so that the function decreases:

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

For small changes in **x** we can approximate *F*(**x**):

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k$$

where

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_k}$$

If we want the function to decrease:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0$$

We can maximize the decrease by choosing:

$$\mathbf{p}_k = -\mathbf{g}_k$$

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k}$$

Example

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

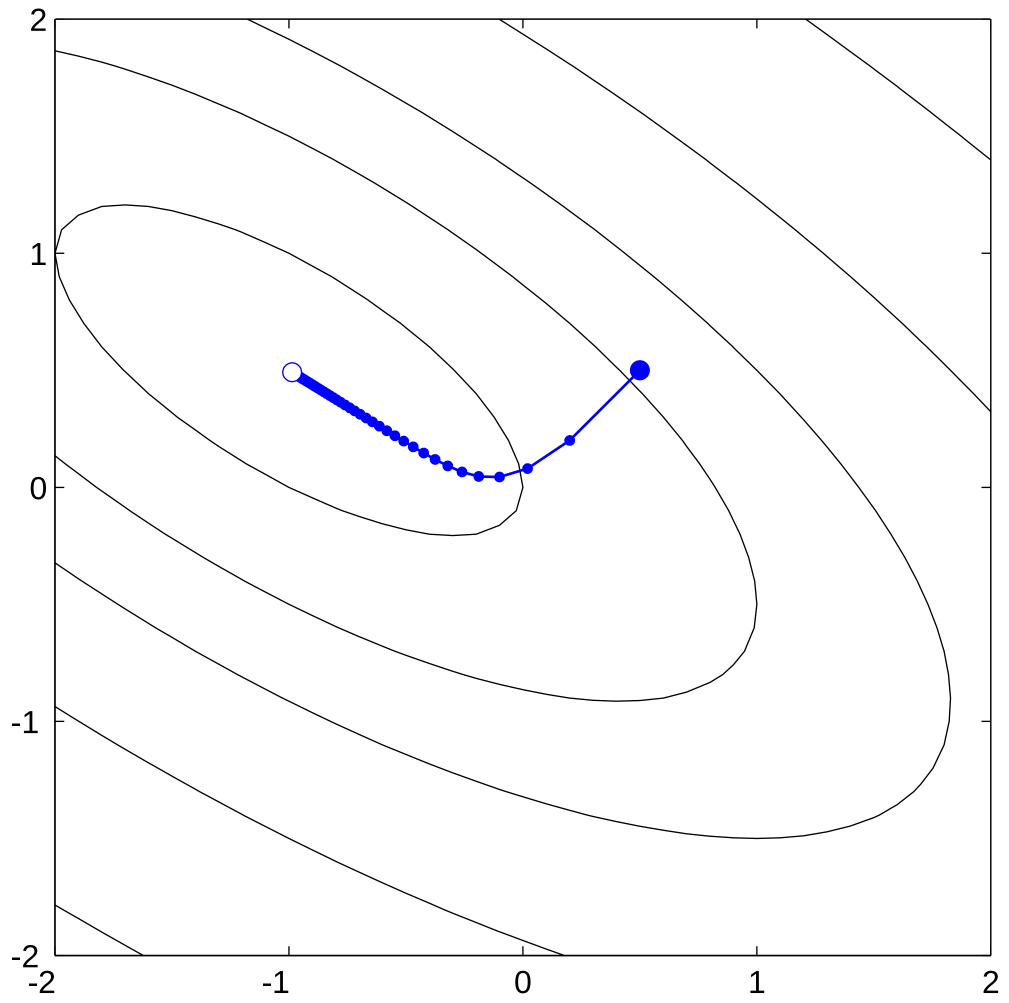
$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \alpha = 0.1$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \qquad \mathbf{g}_0 = \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$$

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} - 0.1 \begin{bmatrix} 1.8 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.08 \end{bmatrix}$$

Plot



Stable Learning Rates (Quadratic)

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c$$

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{g}_k = \mathbf{x}_k - \alpha(\mathbf{A}\mathbf{x}_k + \mathbf{d}) \Longrightarrow \mathbf{x}_{k+1} = \underbrace{[\mathbf{I} - \alpha\mathbf{A}]}_{\text{Stability is determined by the eigenvalues of this matrix.}}\mathbf{x}_k - \alpha\mathbf{d}$$

$$[\mathbf{I} - \alpha\mathbf{A}]\mathbf{z}_i = \mathbf{z}_i - \alpha\mathbf{A}\mathbf{z}_i = \mathbf{z}_i - \alpha\lambda_i\mathbf{z}_i = \underbrace{(1 - \alpha\lambda_i)}_{\text{Eigenvalues of } [\mathbf{I} - \alpha\mathbf{A}].}\mathbf{z}_i$$

(λ_i - eigenvalue of **A**)

Stability Requirement:

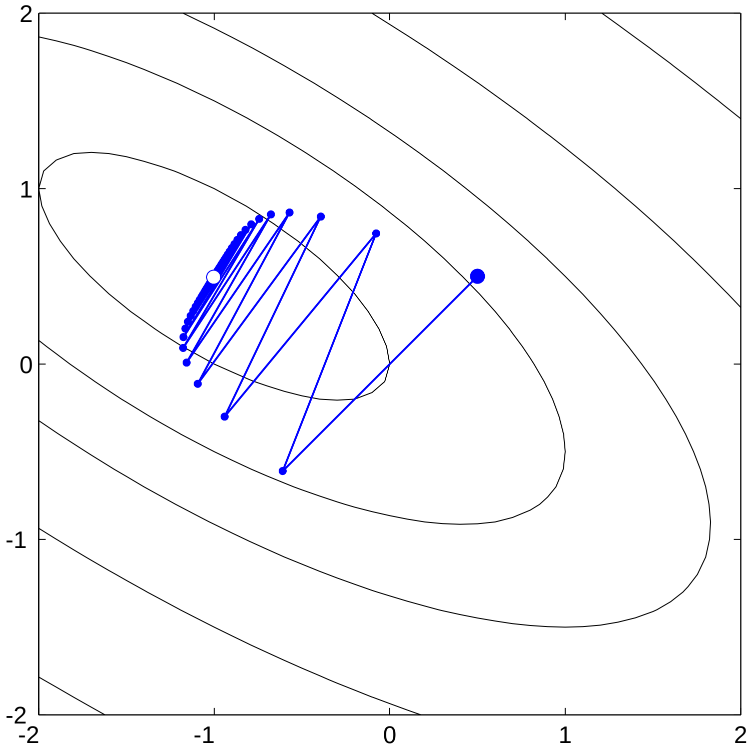
$$|(1 - \alpha\lambda_i)| < 1 \qquad \alpha < \frac{2}{\lambda_i} \qquad \boxed{\alpha < \frac{2}{\lambda_{max}}}$$

Example

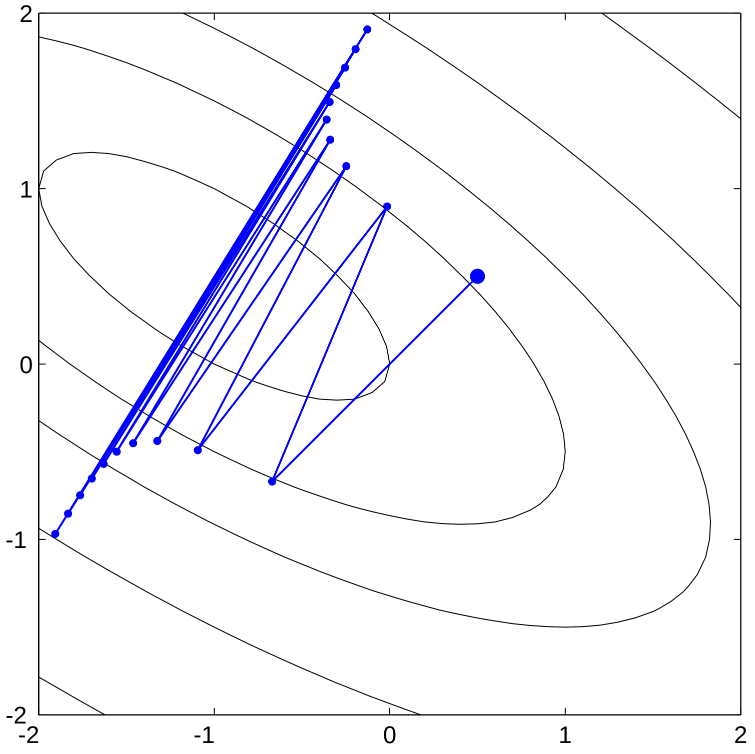
$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \qquad \left\{ (\lambda_1 = 0.764), \left(\mathbf{z}_1 = \begin{bmatrix} 0.851 \\ -0.526 \end{bmatrix} \right) \right\}, \left\{ \lambda_2 = 5.24, \left(\mathbf{z}_2 = \begin{bmatrix} 0.526 \\ 0.851 \end{bmatrix} \right) \right\}$$

$$\alpha < \frac{2}{\lambda_{max}} = \frac{2}{5.24} = 0.38$$

$\alpha = 0.37$



$\alpha = 0.39$



Minimizing Along a Line

Choose α_k to minimize $F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$

$$\frac{d}{d\alpha_k}(F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)) = \nabla F(\mathbf{x})^T \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k + \alpha_k \mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k$$

$$\alpha_k = - \frac{\nabla F(\mathbf{x})^T \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k} = - \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k}$$

where

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k}$$

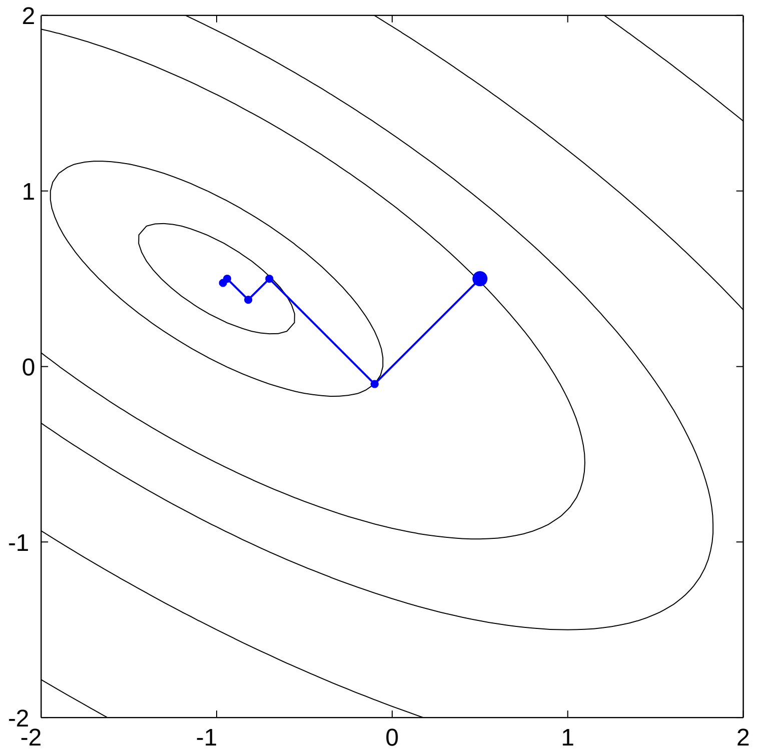
Example

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\begin{bmatrix}2 & 2\\2 & 4\end{bmatrix}\mathbf{x} + \begin{bmatrix}1 & 0\end{bmatrix}\mathbf{x} \qquad \mathbf{x}_0 = \begin{bmatrix}0.5\\0.5\end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix}\frac{\partial}{\partial x_1}F(\mathbf{x})\\ \frac{\partial}{\partial x_2}F(\mathbf{x})\end{bmatrix} = \begin{bmatrix}2x_1 + 2x_2 + 1\\ 2x_1 + 4x_2\end{bmatrix} \qquad \mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix}-3\\-3\end{bmatrix}$$

$$\alpha_0 = -\frac{\begin{bmatrix}3 & 3\end{bmatrix}\begin{bmatrix}-3\\-3\end{bmatrix}}{\begin{bmatrix}-3 & -3\end{bmatrix}\begin{bmatrix}2 & 2\\2 & 4\end{bmatrix}\begin{bmatrix}-3\\-3\end{bmatrix}} = 0.2 \qquad \mathbf{x}_1 = \mathbf{x}_0 - \alpha_0\mathbf{g}_0 = \begin{bmatrix}0.5\\0.5\end{bmatrix} - 0.2\begin{bmatrix}3\\3\end{bmatrix} = \begin{bmatrix}-0.1\\-0.1\end{bmatrix}$$

Plot



Successive steps are orthogonal.

$$\frac{d}{d\alpha_k}F(\mathbf{x}_k + \alpha_k\mathbf{p}_k) = \frac{d}{d\alpha_k}F(\mathbf{x}_{k+1}) = \nabla F(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_{k+1}}\frac{d}{d\alpha_k}[\mathbf{x}_k + \alpha_k\mathbf{p}_k]$$

$$= \nabla F(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_{k+1}}\mathbf{p}_k = \mathbf{g}_{k+1}^T\mathbf{p}_k$$

Newton's Method

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T\Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T\mathbf{A}_k\Delta\mathbf{x}_k$$

Take the gradient of this second-order approximation and set it equal to zero to find the stationary point:

$$\mathbf{g}_k + \mathbf{A}_k\Delta\mathbf{x}_k = \mathbf{0}$$

$$\Delta\mathbf{x}_k = -\mathbf{A}_k^{-1}\mathbf{g}_k$$

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k}$$

Example

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

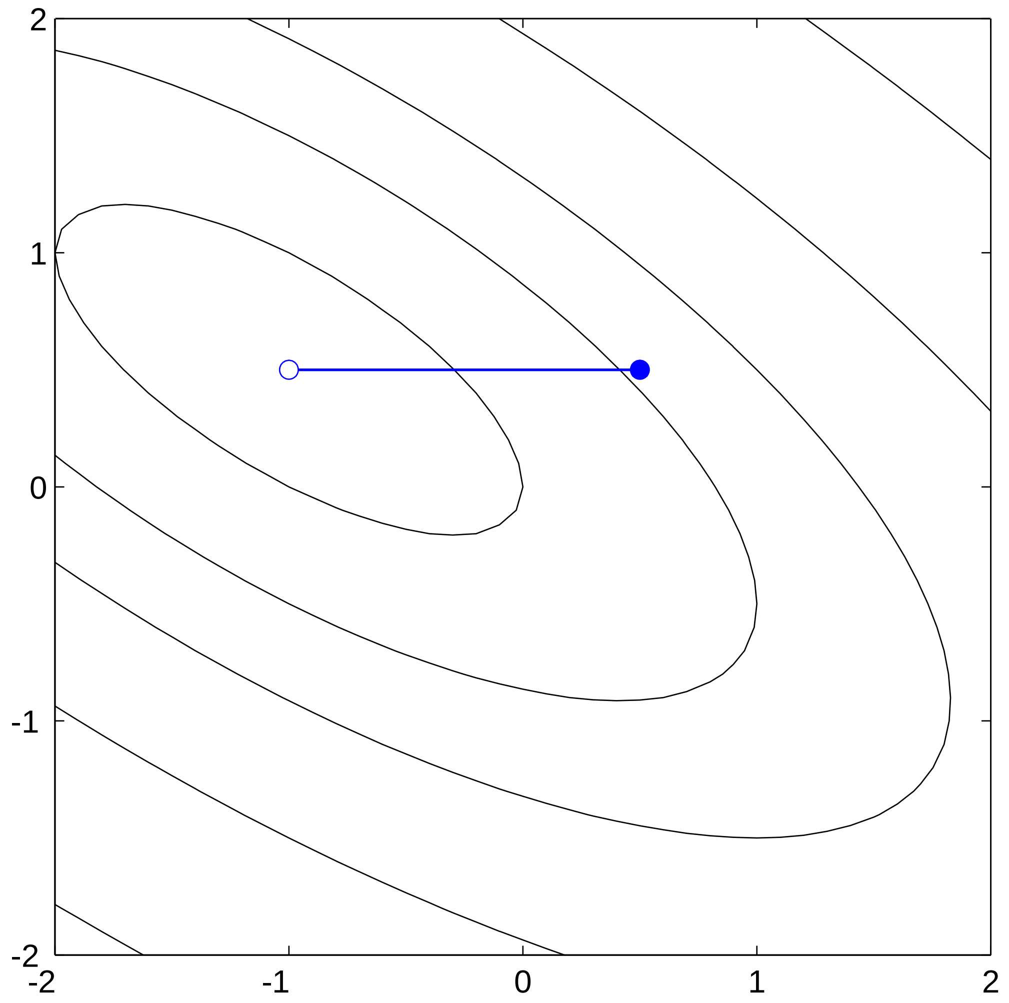
$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix}$$

$$\mathbf{g}_0 = \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

Plot

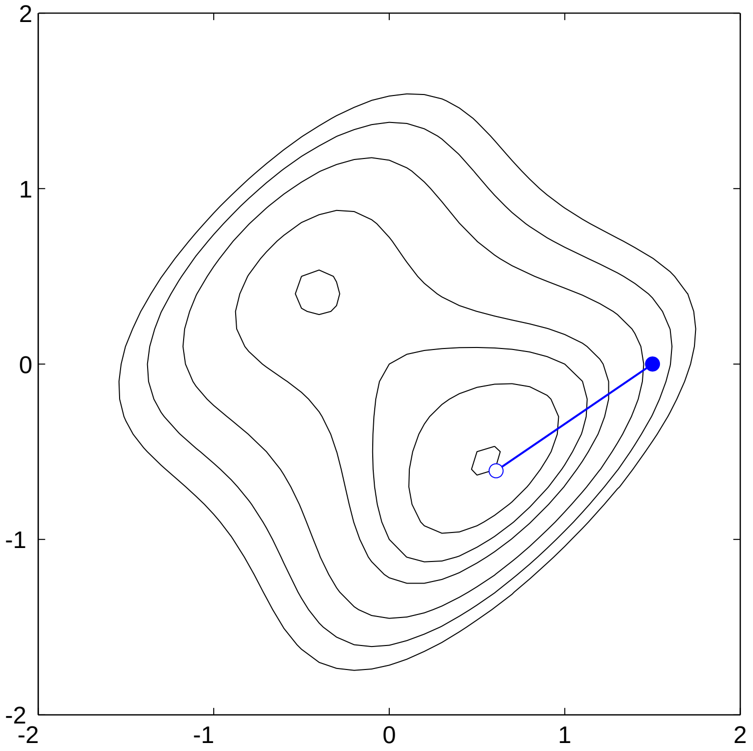


Non-Quadratic Example

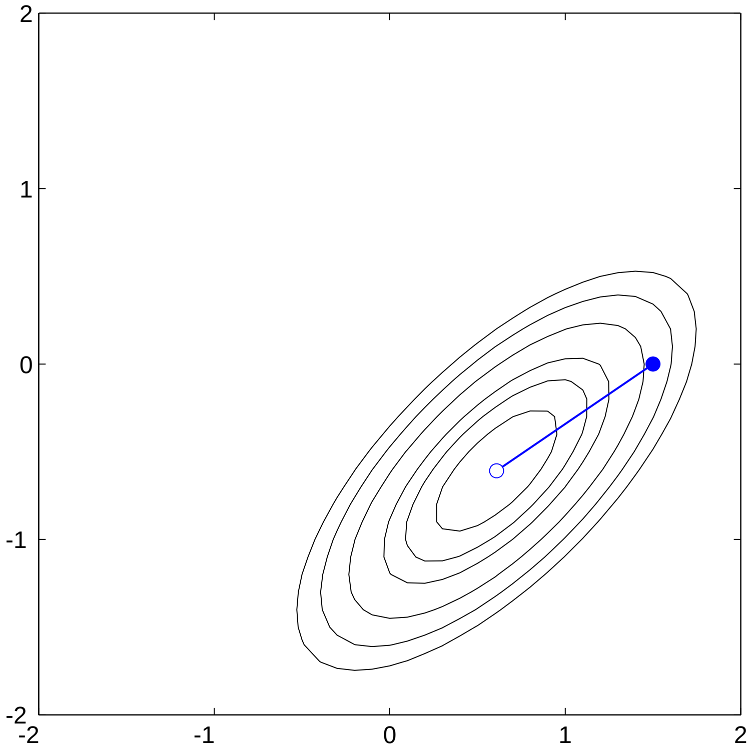
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

Stationary Points: $\mathbf{x}^1 = \begin{bmatrix} -0.42 \\ 0.42 \end{bmatrix}$ $\mathbf{x}^2 = \begin{bmatrix} -0.13 \\ 0.13 \end{bmatrix}$ $\mathbf{x}^3 = \begin{bmatrix} 0.55 \\ -0.55 \end{bmatrix}$

$F(\mathbf{x})$

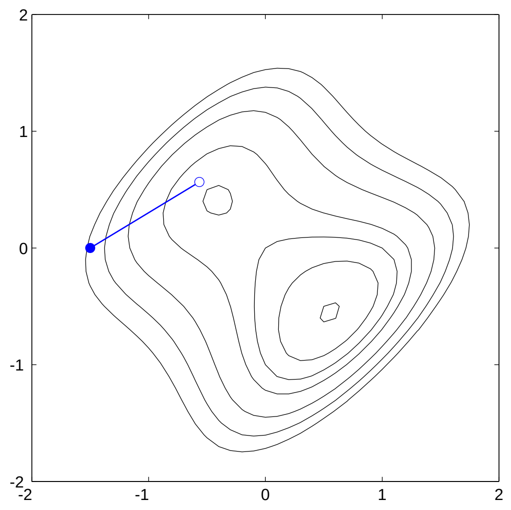


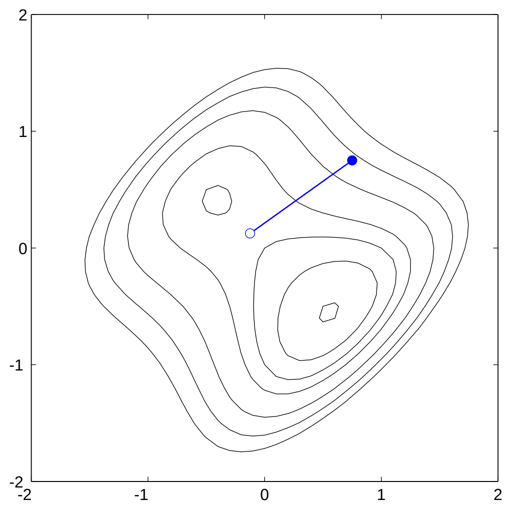
$F_2(\mathbf{x})$

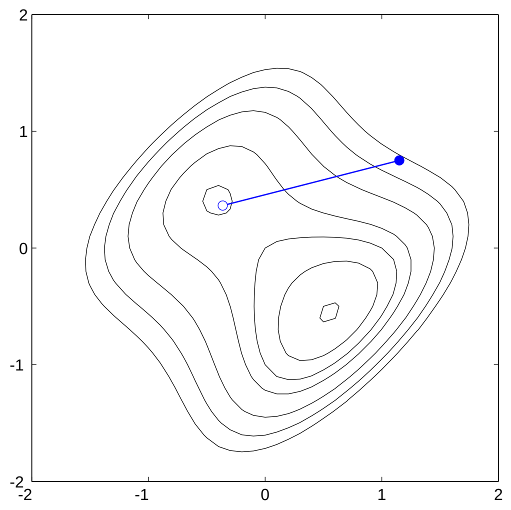


Different Initial Conditions

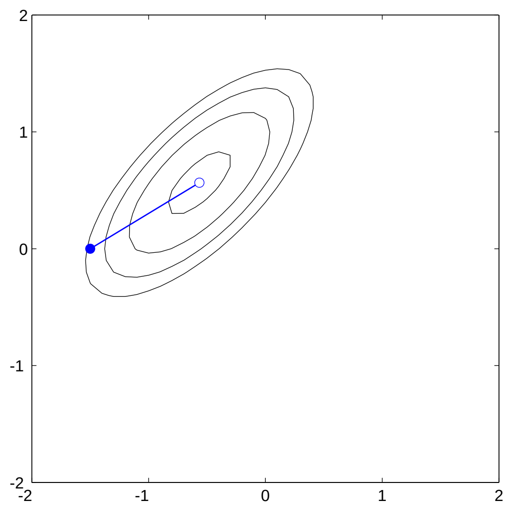
$F(\mathbf{x})$

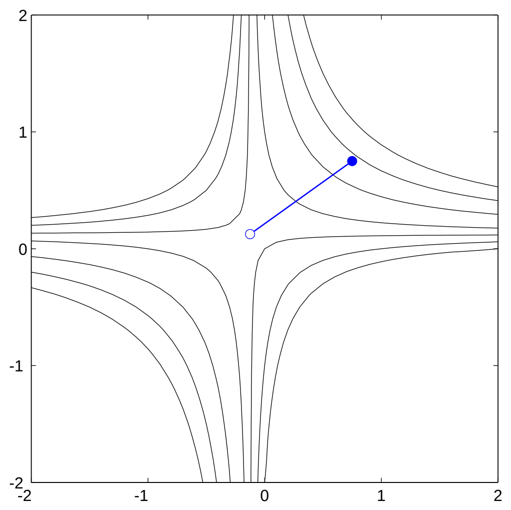


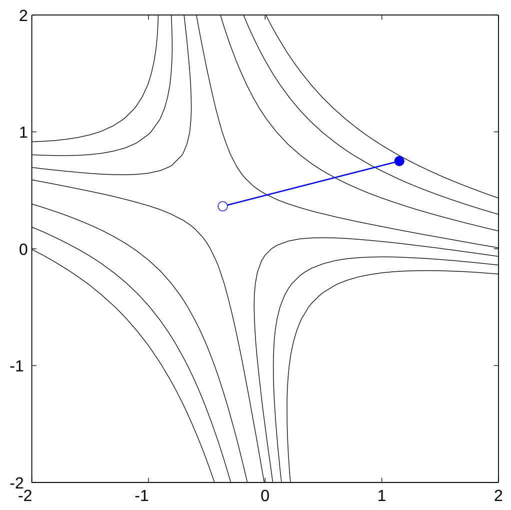




$F_2(\mathbf{x})$







Conjugate Vectors

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c$$

A set of vectors is mutually <u>conjugate</u> with respect to a positive definite Hessian matrix **A** if

$$\mathbf{p}_k^T\mathbf{A}\mathbf{p}_j = 0 \qquad k \neq j$$

One set of conjugate vectors consists of the eigenvectors of **A**.

$$\mathbf{z}_k^T\mathbf{A}\mathbf{z}_j = \lambda_j\mathbf{z}_k^T\mathbf{z}_j = 0 \qquad k \neq j$$

(The eigenvectors of symmetric matrices are orthogonal.)

For Quadratic Functions

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}$$

$$\nabla^2 F(\mathbf{x}) = \mathbf{A}$$

The change in the gradient at iteration *k* is

$$\Delta\mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k = (\mathbf{A}\mathbf{x}_{k+1} + \mathbf{d}) - (\mathbf{A}\mathbf{x}_k + \mathbf{d}) = \mathbf{A}\Delta\mathbf{x}_k$$

where

$$\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$

The conjugacy conditions can be rewritten

$$\alpha_k \mathbf{p}_k^T \mathbf{A}\mathbf{p}_j = \Delta\mathbf{x}_k^T \mathbf{A}\mathbf{p}_j = \Delta\mathbf{g}_k^T \mathbf{p}_j = 0 \qquad k \neq j$$

This does not require knowledge of the Hessian matrix.

Forming Conjugate Directions

Choose the initial search direction as the negative of the gradient.

$$\mathbf{p}_0 = -\mathbf{g}_0$$

Choose subsequent search directions to be conjugate.

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

where

$$\beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta\mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Conjugate Gradient algorithm

- The first search direction is the negative of the gradient.

$$\mathbf{p}_0 = -\mathbf{g}_0$$

- Select the learning rate to minimize along the line.

$$\alpha_k = -\frac{\nabla F(\mathbf{x})^T \big|_{\mathbf{x}=\mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x}=\mathbf{x}_k} \mathbf{p}_k} = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k}$$

(For quadratic functions.)

- Select the next search direction using

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

- If the algorithm has not converged, return to second step.

- A quadratic function will be minimized in *n* steps.

Example

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\begin{bmatrix}2 & 2\\2 & 4\end{bmatrix}\mathbf{x} + \begin{bmatrix}1 & 0\end{bmatrix}\mathbf{x} \qquad \mathbf{x}_0 = \begin{bmatrix}0.5\\0.5\end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix}\frac{\partial}{\partial x_1}F(\mathbf{x})\\ \frac{\partial}{\partial x_2}F(\mathbf{x})\end{bmatrix} = \begin{bmatrix}2x_1 + 2x_2 + 1\\ 2x_1 + 4x_2\end{bmatrix} \qquad \mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix}-3\\-3\end{bmatrix}$$

$$\alpha_0 = -\frac{\begin{bmatrix}3 & 3\end{bmatrix}\begin{bmatrix}-3\\-3\end{bmatrix}}{\begin{bmatrix}-3 & -3\end{bmatrix}\begin{bmatrix}2 & 2\\2 & 4\end{bmatrix}\begin{bmatrix}-3\\-3\end{bmatrix}} = 0.2 \qquad \mathbf{x}_1 = \mathbf{x}_0 - \alpha_0\mathbf{g}_0 = \begin{bmatrix}0.5\\0.5\end{bmatrix} - 0.2\begin{bmatrix}3\\3\end{bmatrix} = \begin{bmatrix}-0.1\\-0.1\end{bmatrix}$$

Example

$$\mathbf{g}_1 = \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_1} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}$$

$$\beta_1 = \frac{\mathbf{g}_1^T\mathbf{g}_1}{\mathbf{g}_0^T\mathbf{g}_0} = \frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix}\begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}}{\begin{bmatrix} 3 & 3 \end{bmatrix}\begin{bmatrix} 3 \\ 3 \end{bmatrix}} = \frac{0.72}{18} = 0.04$$

$$\mathbf{p}_1 = -\mathbf{g}_1 + \beta_1\mathbf{p}_0 = \begin{bmatrix} -0.6 \\ 0.6 \end{bmatrix} + 0.04\begin{bmatrix} -3 \\ -3 \end{bmatrix} = \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}$$

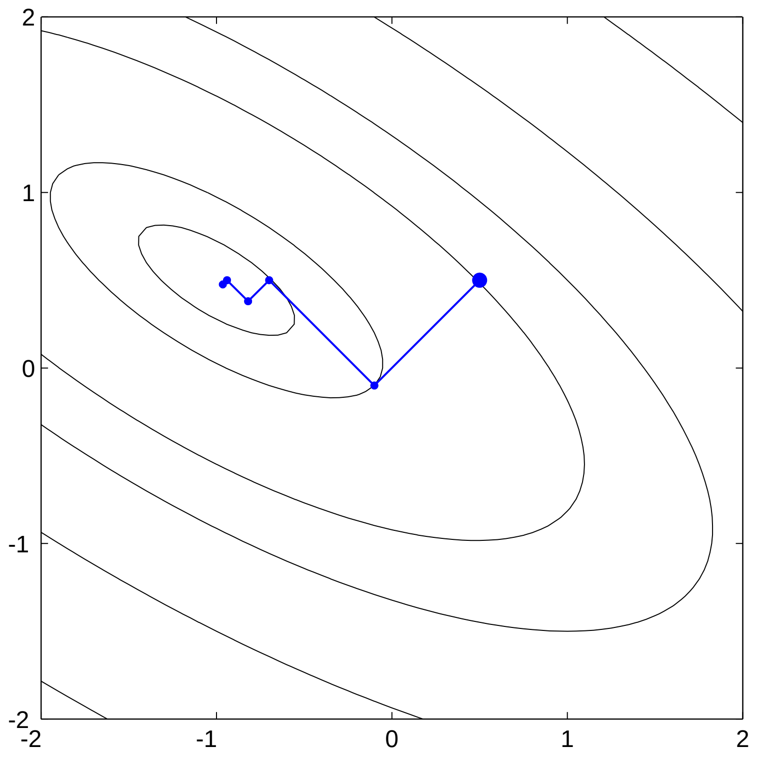
$$\alpha_1 = -\frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix}\begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}}{\begin{bmatrix} -0.72 & 0.48 \end{bmatrix}\begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}} = -\frac{-0.72}{0.576} = 1.25$$

Plots

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix} + 1.25 \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

Conjugate Gradient

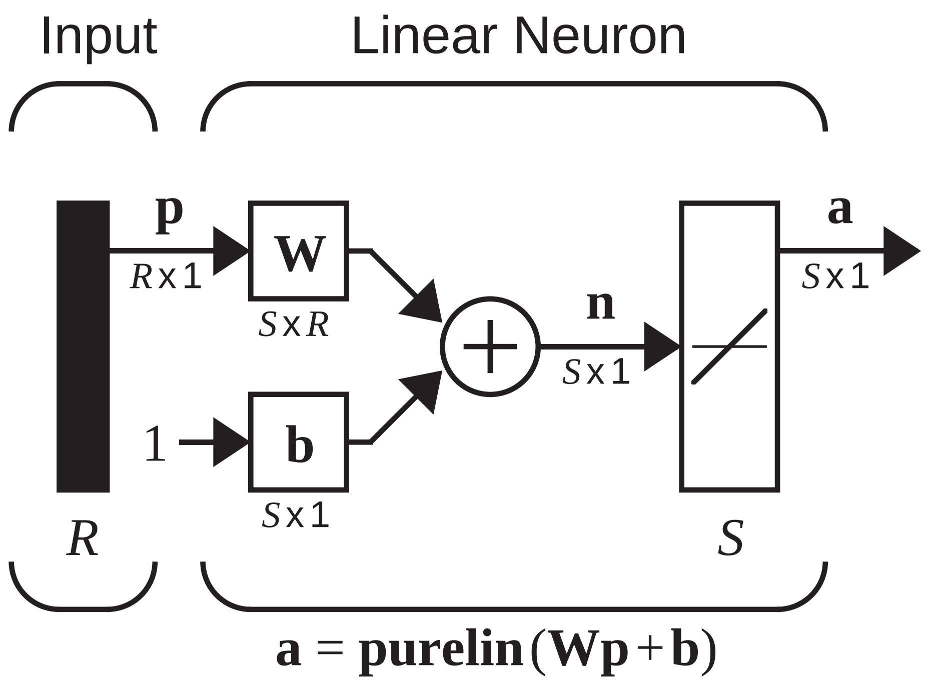
Steepest Descent



Widrow-Hoff Learning

(LMS Algorithm)

10 ADALINE Network



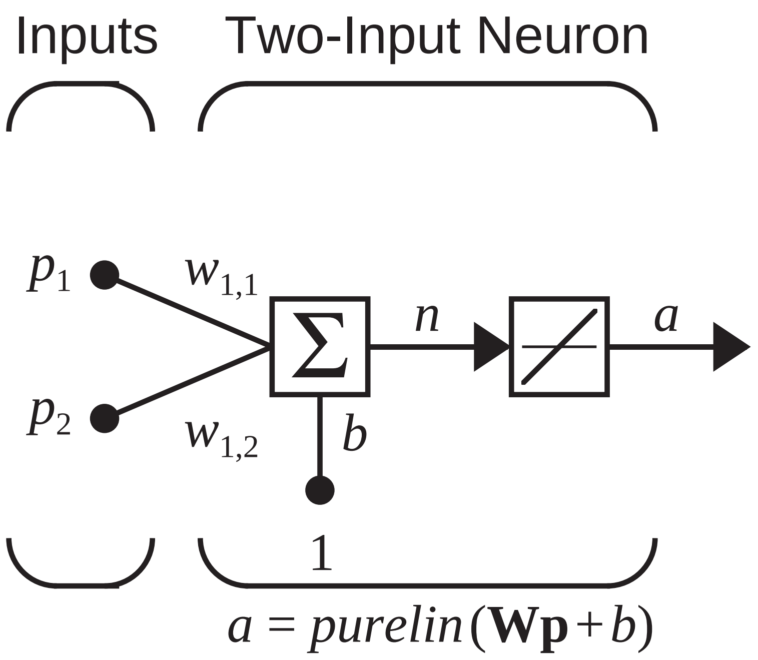
$$\mathbf{a} = \mathbf{purelin}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

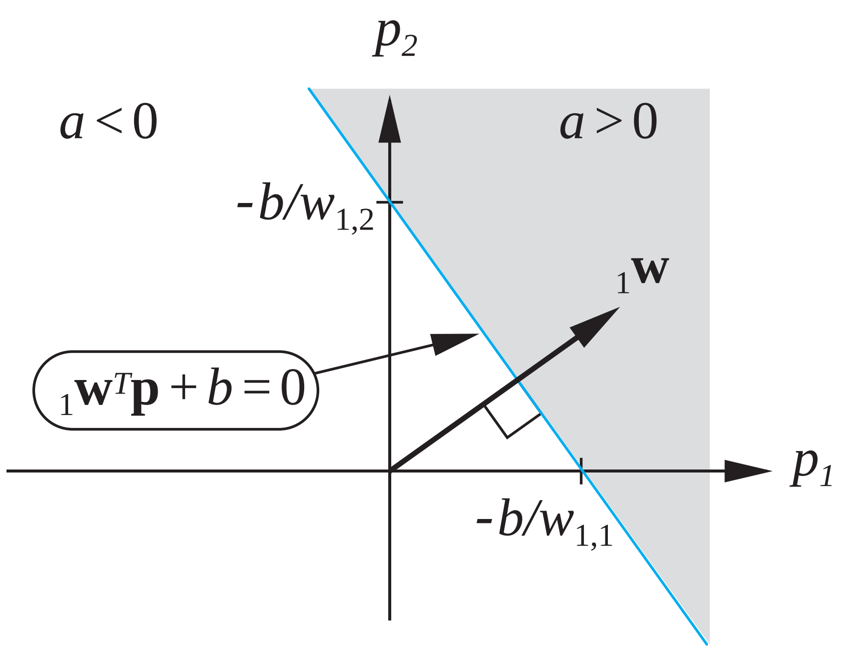
$$\mathbf{a} = \mathbf{purelin}(\mathbf{W}\mathbf{p} + \mathbf{b}) = \mathbf{W}\mathbf{p} + \mathbf{b}$$

$$a_i = purelin(n_i) = purelin({}_i\mathbf{w}^T\mathbf{p} + b_i) = {}_i\mathbf{w}^T\mathbf{p} + b_i$$

$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}$$

Two-Input ADALINE





$$a = purelin(n) = purelin({}_1\mathbf{w}^T\mathbf{p} + b) = {}_1\mathbf{w}^T\mathbf{p} + b$$

$$a = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b$$

Mean Square Error

Training Set:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Input: $\mathbf{p}_q$ Target: $\mathbf{t}_q$

Notation:

$$\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix} \qquad \mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \qquad a = {}_1\mathbf{w}^T\mathbf{p} + b \Longrightarrow a = \mathbf{x}^T\mathbf{z}$$

Mean Square Error:

$$F(\mathbf{x}) = E[e^2] = E[(t - a)^2] = E[(t - \mathbf{x}^T\mathbf{z})^2]$$

Error Analysis

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2]$$

$$F(\mathbf{x}) = E[t^2 - 2t\mathbf{x}^T\mathbf{z} + \mathbf{x}^T\mathbf{z}\mathbf{z}^T\mathbf{x}]$$

$$F(\mathbf{x}) = E[t^2] - 2\mathbf{x}^T\mathrm{E}[t\mathbf{z}] + \mathbf{x}^T E[\mathbf{z}\mathbf{z}^T]\mathbf{x}$$

$$\boxed{F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}}$$

$$c = E[t^2] \qquad \mathbf{h} = E[t\mathbf{z}] \qquad \mathbf{R} = E[\mathbf{z}\mathbf{z}^T]$$

The mean square error for the ADALINE Network is a quadratic function:

$$F(\mathbf{x}) = c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$$

$$\mathbf{d} = -2\mathbf{h} \qquad \mathbf{A} = 2\mathbf{R}$$

Stationary Point

Hessian Matrix:

$$\mathbf{A} = 2\mathbf{R}$$

The correlation matrix **R** must be at least positive semidefinite. If there are any zero eigenvalues, the performance index will either have a weak minumum or else no stationary point, otherwise there will be a unique global minimum **x***.

$$\nabla F(\mathbf{x}) = \nabla\left(c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \mathbf{d} + \mathbf{A}\mathbf{x} = -2\mathbf{h} + 2\mathbf{R}\mathbf{x}$$

$$-2\mathbf{h} + 2\mathbf{R}\mathbf{x} = \mathbf{0}$$

If R is positive definite:

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$$

Approximate Steepest Descent

Approximate mean square error (one sample):

$$\hat{F}(\mathbf{x}) = (t(k) - a(k))^2 = e^2(k)$$

Approximate (stochastic) gradient:

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k)$$

$$[\nabla e^2(k)]_j = \frac{\partial e^2(k)}{\partial w_{1,j}} = 2e(k)\frac{\partial e(k)}{\partial w_{1,j}} \qquad j = 1, 2, \ldots, R$$

$$[\nabla e^2(k)]_{R+1} = \frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e(k)}{\partial b}$$

Approximate Gradient Calculation

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial[t(k) - a(k)]}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}}[t(k) - ({}_1\mathbf{w}^T\mathbf{p}(k) + b)]$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}}\left[t(k) - \left(\sum_{i=1}^{R} w_{1,i}p_i(k) + b\right)\right]$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k) \qquad \frac{\partial e(k)}{\partial b} = -1$$

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k) = -2e(k)\mathbf{z}(k)$$

LMS Algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k) \mathbf{z}(k)$$

$${}_1\mathbf{w}(k+1) = {}_1\mathbf{w}(k) + 2\alpha e(k) \mathbf{p}(k)$$

$$b(k+1) = b(k) + 2\alpha e(k)$$

Multiple-Neuron Case

$${}_i\mathbf{w}(k+1) = {}_i\mathbf{w}(k) + 2\alpha e_i(k)\mathbf{p}(k)$$

$$b_i(k+1) = b_i(k) + 2\alpha e_i(k)$$

Matrix Form:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\mathbf{e}(k)\mathbf{p}^T(k)$$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\mathbf{e}(k)$$

Analysis of Convergence

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k)$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha E[e(k)\mathbf{z}(k)]$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{E[t(k)\mathbf{z}(k)] - E[(\mathbf{x}_k^T\mathbf{z}(k))\mathbf{z}(k)]\}$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{E[t_k\mathbf{z}(k)] - E[(\mathbf{z}(k)\mathbf{z}^T(k))\mathbf{x}_k]\}$$

$$E[\mathbf{x}_{k+1}] = E[\mathbf{x}_k] + 2\alpha\{\mathbf{h} - \mathbf{R}E[\mathbf{x}_k]\}$$

$$E[\mathbf{x}_{k+1}] = \underbrace{[\mathbf{I} - 2\alpha\mathbf{R}]}_{\text{For stability, the eigenvalues of this matrix must fall inside the unit circle.}}E[\mathbf{x}_k] + 2\alpha\mathbf{h}$$

Conditions for Stability

$$\left|eig([\mathbf{I} - 2\alpha\mathbf{R}])\right| = \left|1 - 2\alpha\lambda_i\right| < 1$$

(where λ_i is an eigenvalue of $\mathbf{R}$)

Since $\lambda_i > 0$, $1 - 2\alpha\lambda_i < 1$.

Therefore the stability condition simplifies to

$$1 - 2\alpha\lambda_i > -1$$

$$\alpha < 1/\lambda_i \quad \text{for all } i$$

$$\boxed{0 < \alpha < 1/\lambda_{max}}$$

Steady State Response

$$E[\mathbf{x}_{k+1}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_k] + 2\alpha\mathbf{h}$$

If the system is stable, then a steady state condition will be reached.

$$E[\mathbf{x}_{ss}] = [\mathbf{I} - 2\alpha\mathbf{R}]E[\mathbf{x}_{ss}] + 2\alpha\mathbf{h}$$

The solution to this equation is

$$E[\mathbf{x}_{ss}] = \mathbf{R}^{-1}\mathbf{h} = \mathbf{x}^*$$

This is also the strong minimum of the performance index.

Example

Banana $\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\}$ Apple $\left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$

$$\mathbf{R} = E[\mathbf{p}\mathbf{p}^T] = \frac{1}{2}\mathbf{p}_1\mathbf{p}_1^T + \frac{1}{2}\mathbf{p}_2\mathbf{p}_2^T$$

$$\mathbf{R} = \frac{1}{2}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

$$\lambda_1 = 1.0, \qquad \lambda_2 = 0.0, \qquad \lambda_3 = 2.0$$

$$\alpha < \frac{1}{\lambda_{max}} = \frac{1}{2.0} = 0.5$$

Iteration One

Banana $$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{p}_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0$$

$$e(0) = t(0) - a(0) = t_1 - a(0) = -1 - 0 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\alpha e(0)\mathbf{p}^T(0)$$

$$\mathbf{W}(1) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} + 2(0.2)(-1)\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix}$$

Iteration Two

Apple $a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$

$$e(1) = t(1) - a(1) = t_2 - a(1) = 1 - (-0.4) = 1.4$$

$$\mathbf{W}(2) = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} + 2(0.2)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix}$$

Iteration Three

$$a(2) = \mathbf{W}(2)\mathbf{p}(2) = \mathbf{W}(2)\mathbf{p}_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

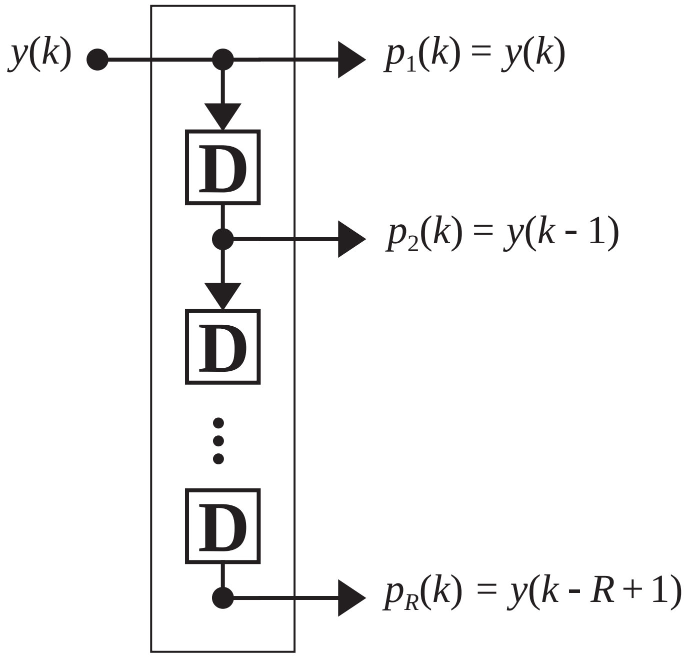
$$e(2) = t(2) - a(2) = t_1 - a(2) = -1 - (-0.64) = -0.36$$

$$\mathbf{W}(3) = \mathbf{W}(2) + 2\alpha e(2)\mathbf{p}^T(2) = \begin{bmatrix} 1.1040 & 0.0160 & -0.0160 \end{bmatrix}$$

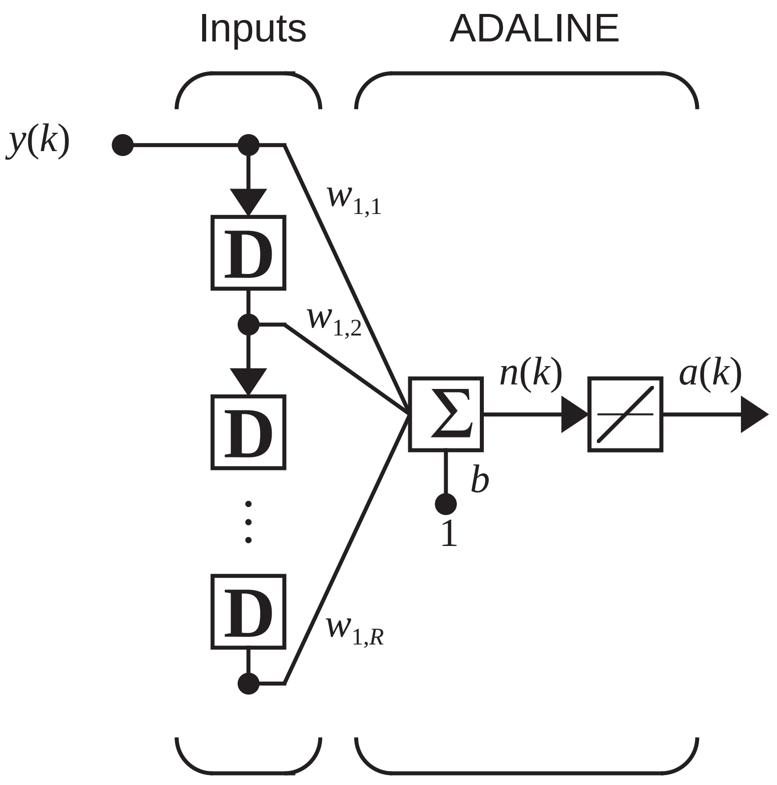
$$\mathbf{W}(\infty) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Adaptive Filtering

Tapped Delay Line



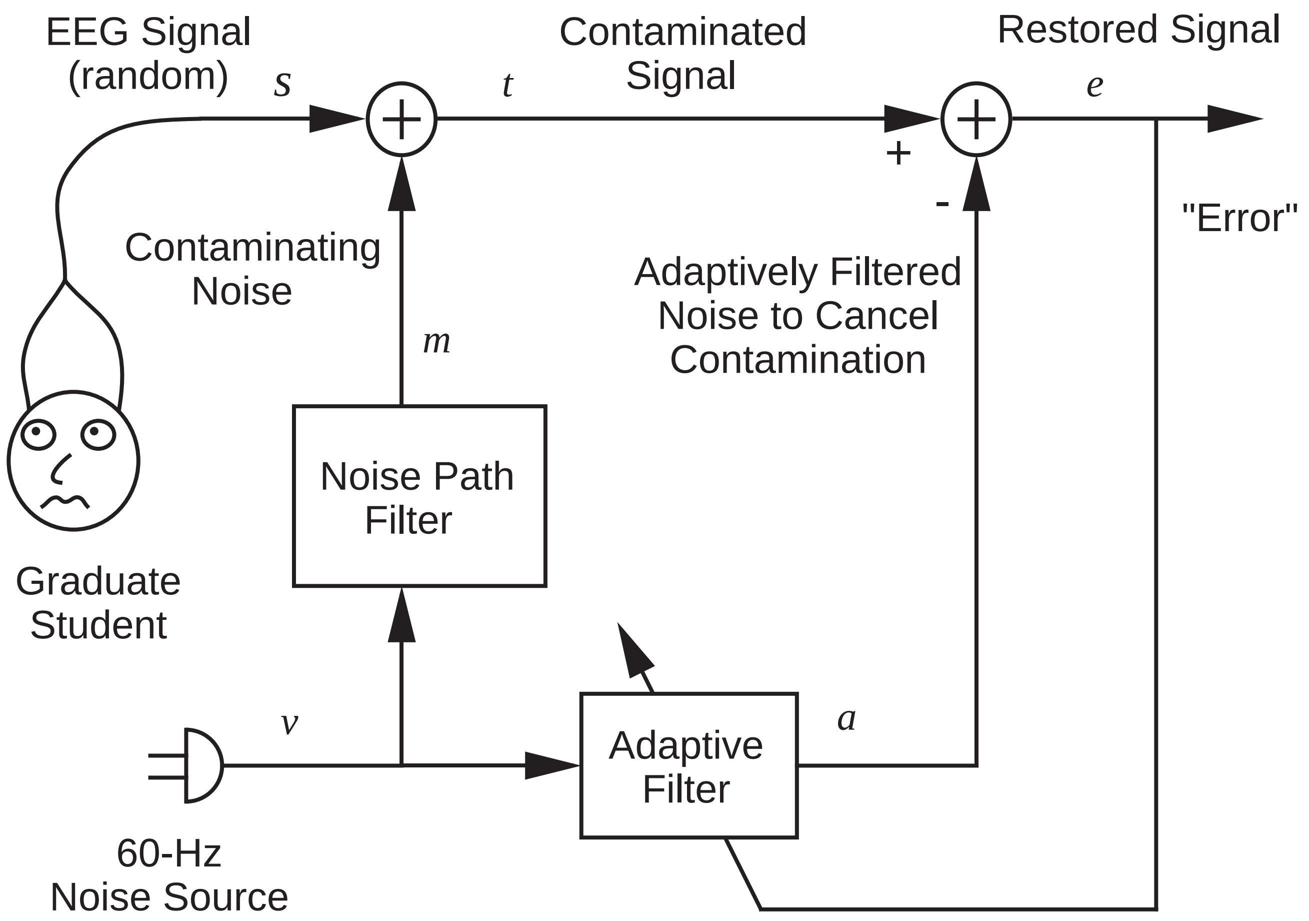
Adaptive Filter



$$a(k) = purelin(\mathbf{W}\mathbf{p}(k) + b)$$

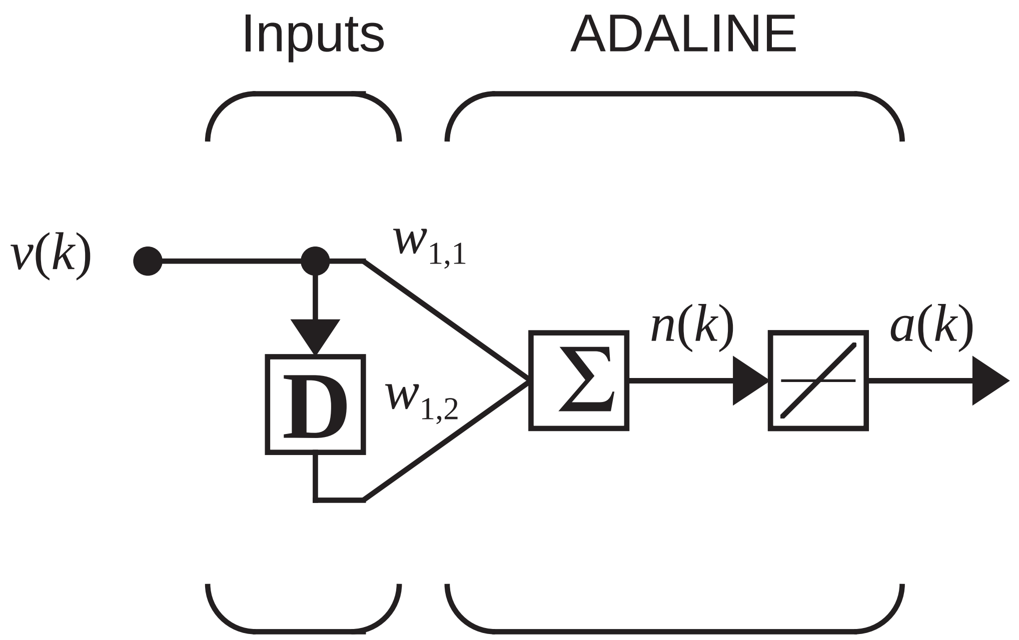
$$a(k) = purelin(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1,i}\, y(k - i + 1) + b$$

Example: Noise Cancellation



Adaptive Filter Adjusts to Minimize Error (and in doing this removes 60-Hz noise from contaminated signal)

Noise Cancellation Adaptive Filter



$$a(k) = w_{1,1}\, v(k) + w_{1,2}\, v(k-1)$$

Correlation Matrix

$$\mathbf{R} = [\mathbf{z}\mathbf{z}^T] \qquad \mathbf{h} = E[t\mathbf{z}]$$

$$\mathbf{z}(k) = \begin{bmatrix} v(k) \\ v(k-1) \end{bmatrix}$$

$$t(k) = s(k) + m(k)$$

$$\mathbf{R} = \begin{bmatrix} E[v^2(k)] & E[v(k)v(k-1)] \\ E[v(k-1)v(k)] & E[v^2(k-1)] \end{bmatrix}$$

$$\mathbf{h} = \begin{bmatrix} E[(s(k)+m(k))v(k)] \\ E[(s(k)+m(k))v(k-1)] \end{bmatrix}$$

Signals

$$v(k) = 1.2\ \sin\left(\frac{2\pi k}{3}\right) \qquad m(k) = 1.2\ \sin\left(\frac{2\pi k}{3} - \frac{3\pi}{4}\right)$$

$$E[v^2(k)] = (1.2)^2 \frac{1}{3}\sum_{k=1}^{3}\left(\sin\left(\frac{2\pi k}{3}\right)\right)^2 = (1.2)^2 0.5 = 0.72$$

$$E[v^2(k-1)] = E[v^2(k)] = 0.72$$

$$E[v(k)v(k-1)] = \frac{1}{3}\sum_{k=1}^{3}\left(1.2\ \sin\frac{2\pi k}{3}\right)\left(1.2\ \sin\frac{2\pi(k-1)}{3}\right)$$

$$= (1.2)^2 0.5\cos\left(\frac{2\pi}{3}\right) = -0.36$$

$$\mathbf{R} = \begin{bmatrix} 0.72 & -0.36 \\ -0.36 & 0.72 \end{bmatrix}$$

Stationary Point

$$E[(s(k)+m(k))v(k)] = E[s(k)v(k)] + E[m(k)v(k)]$$

$$\Downarrow$$

$$0$$

$$E[m(k)v(k)] = \frac{1}{3}\sum_{k=1}^{3}\left(1.2\sin\left(\frac{2\pi k}{3}-\frac{3\pi}{4}\right)\right)\left(1.2\sin\frac{2\pi k}{3}\right) = -0.51$$

$$E[(s(k)+m(k))v(k-1)] = E[s(k)v(k-1)] + E[m(k)v(k-1)]$$

$$\Downarrow$$

$$0$$

$$E[m(k)v(k-1)] = \frac{1}{3}\sum_{k=1}^{3}\left(1.2\sin\left(\frac{2\pi k}{3}-\frac{3\pi}{4}\right)\right)\left(1.2\sin\frac{2\pi(k-1)}{3}\right) = 0.70$$

$$\mathbf{h} = \begin{bmatrix} E[(s(k)+m(k))v(k)] \\ E[(s(k)+m(k))v(k-1)] \end{bmatrix} \quad \Rightarrow \quad \mathbf{h} = \begin{bmatrix} -0.51 \\ 0.70 \end{bmatrix}$$

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h} = \begin{bmatrix} 0.72 & -0.36 \\ -0.36 & 0.72 \end{bmatrix}^{-1} \begin{bmatrix} -0.51 \\ 0.70 \end{bmatrix} = \begin{bmatrix} -0.30 \\ 0.82 \end{bmatrix}$$

Performance Index

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$$

$$c = E[t^2(k)] = E[(s(k) + m(k))^2]$$

$$c = E[s^2(k)] + 2E[s(k)m(k)] + E[m^2(k)]$$

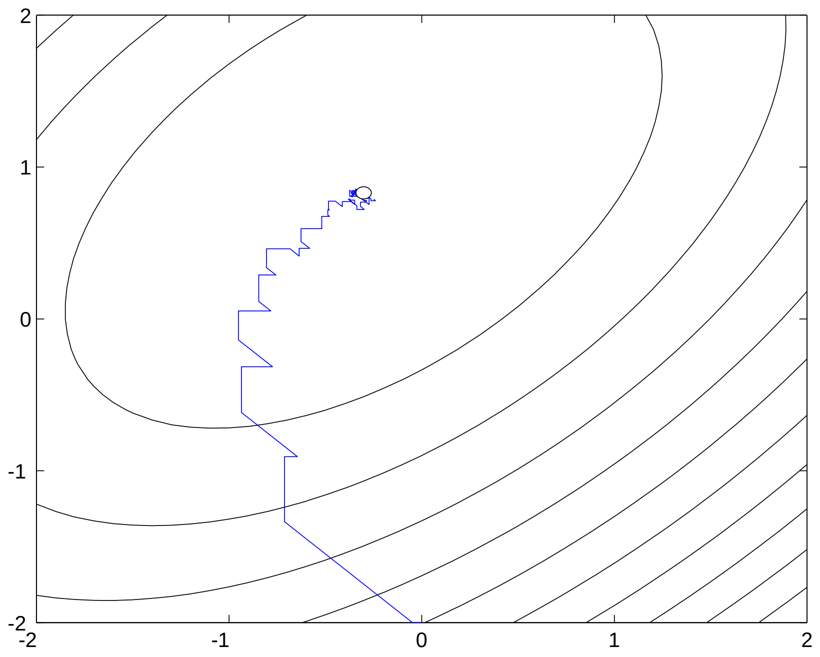
$$E[s^2(k)] = \frac{1}{0.4}\int_{-0.2}^{0.2} s^2 ds = \frac{1}{3(0.4)} s^3 \Big|_{-0.2}^{0.2} = 0.0133$$

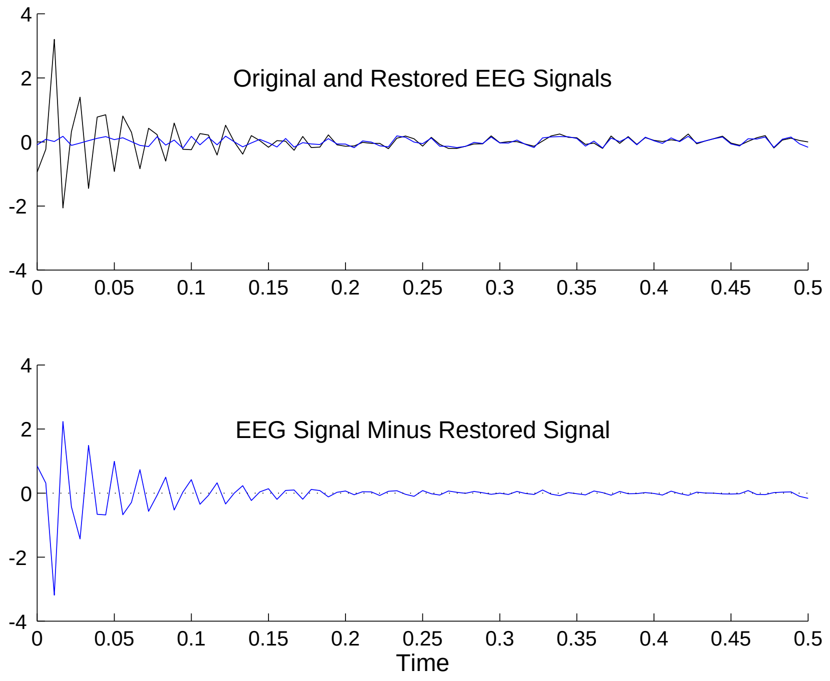
$$E[m^2(k)] = \frac{1}{3}\sum_{k=1}^{3}\left\{1.2 \sin\left(\frac{2\pi}{3} - \frac{3\pi}{4}\right)\right\}^2 = 0.72$$

$$c = 0.0133 + 0.72 = 0.7333$$

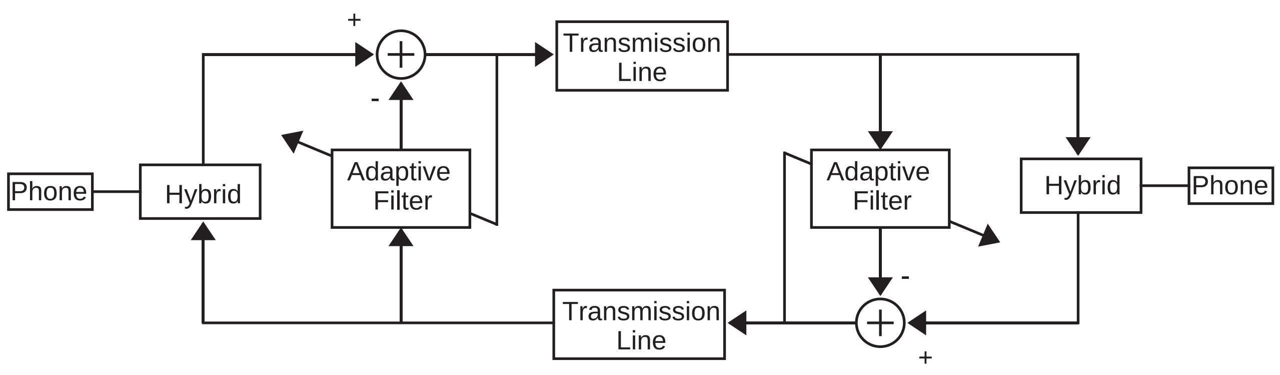
$$F(\mathbf{x}^*) = 0.7333 - 2(0.72) + 0.72 = 0.0133$$

LMS Response





Echo Cancellation

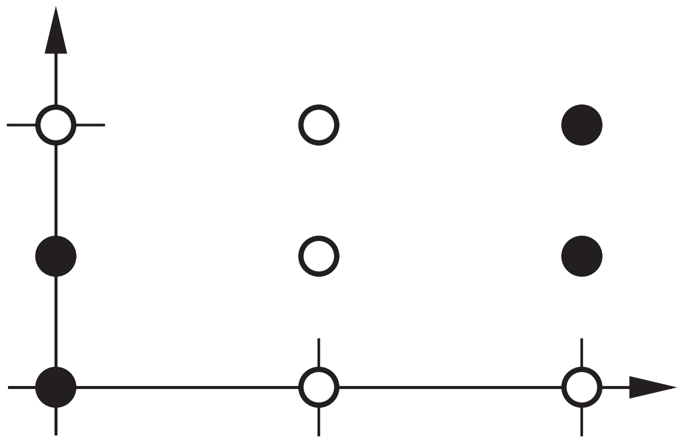


Backpropagation

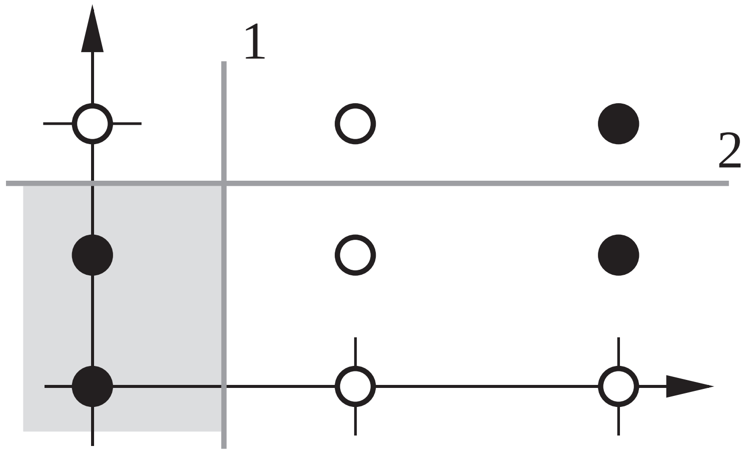
Multilayer Perceptron

R – S^1 – S^2 – S^3 Network

Example



Elementary Decision Boundaries



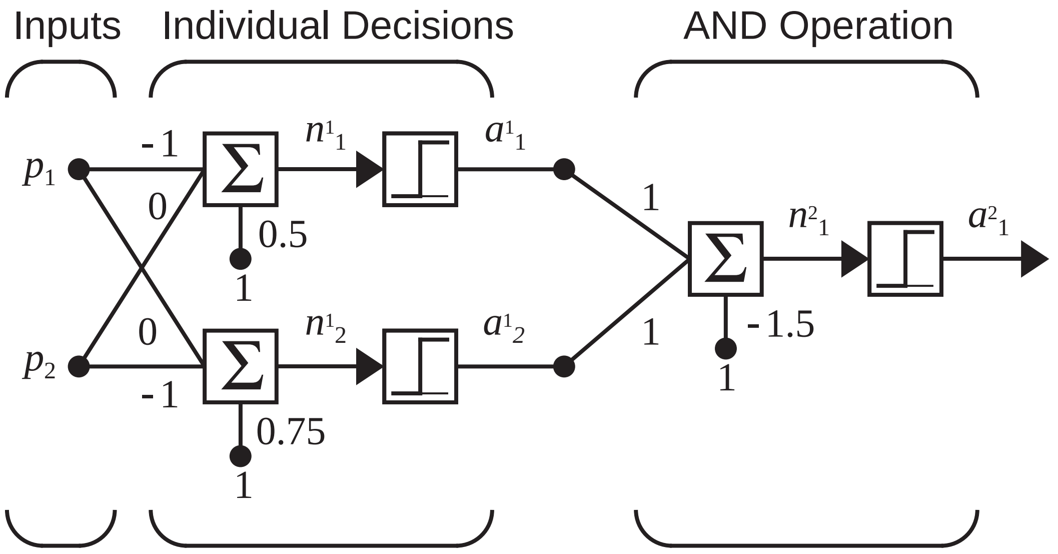
First Boundary:

$$a^1_1 = hardlim(\begin{bmatrix}-1 & 0\end{bmatrix}\mathbf{p} + 0.5)$$

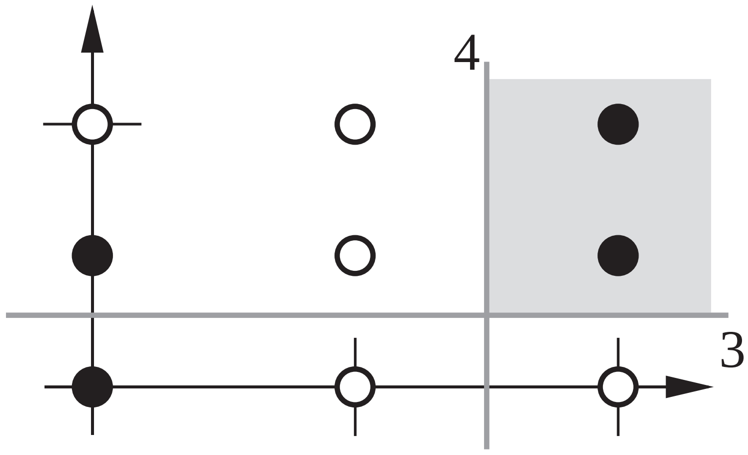
Second Boundary:

$$a^1_2 = hardlim(\begin{bmatrix}0 & -1\end{bmatrix}\mathbf{p} + 0.75)$$

First Subnetwork



Elementary Decision Boundaries



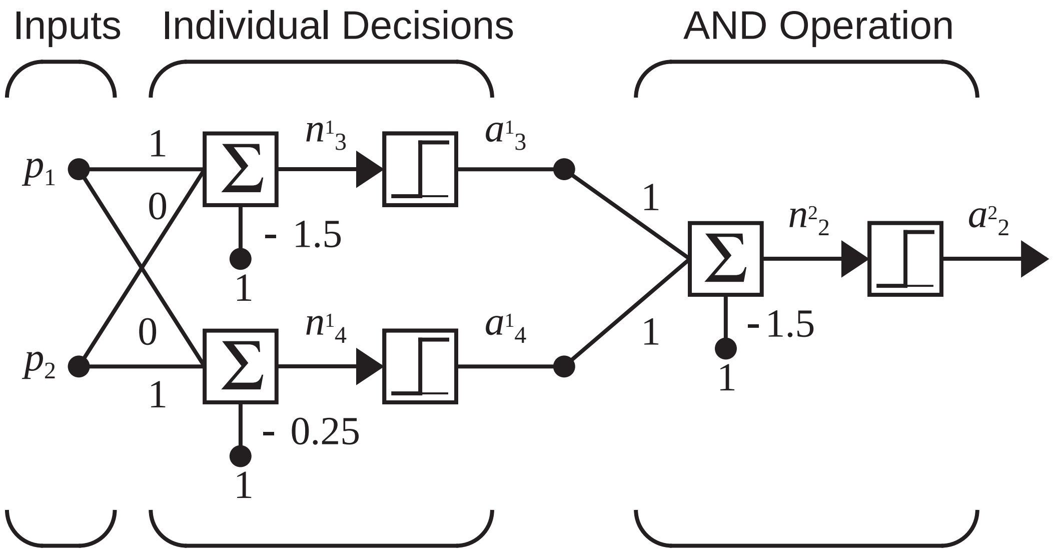
Third Boundary:

$$a_3^1 = hardlim(\begin{bmatrix}1 & 0\end{bmatrix}\mathbf{p} - 1.5)$$

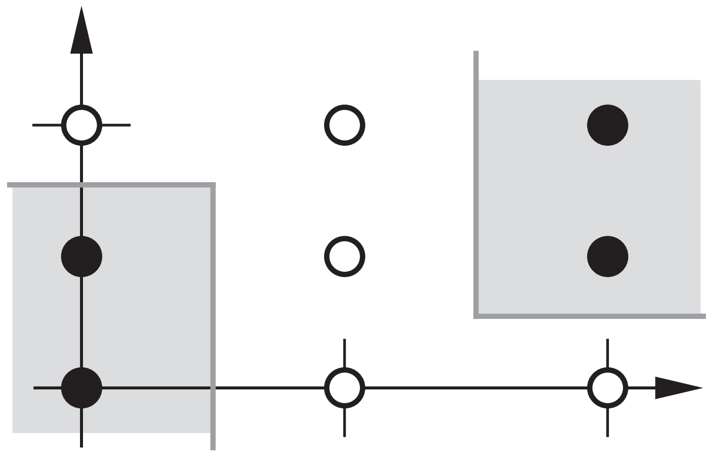
Fourth Boundary:

$$a_4^1 = hardlim(\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{p} - 0.25)$$

Second Subnetwork



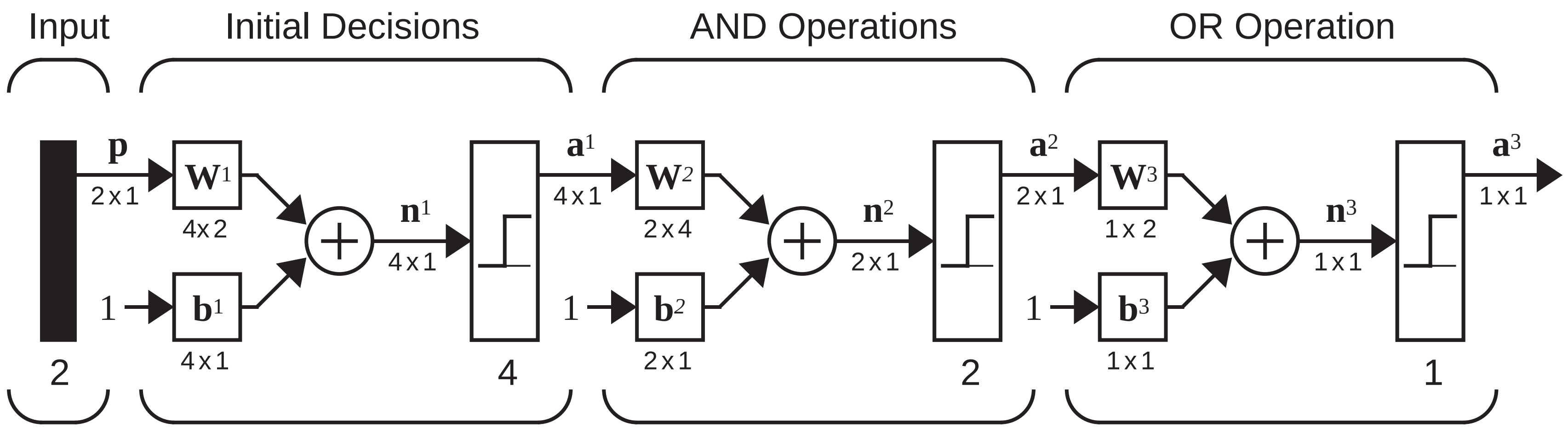
Total Network



$$\mathbf{W}^1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.75 \\ -1.5 \\ -0.25 \end{bmatrix}$$

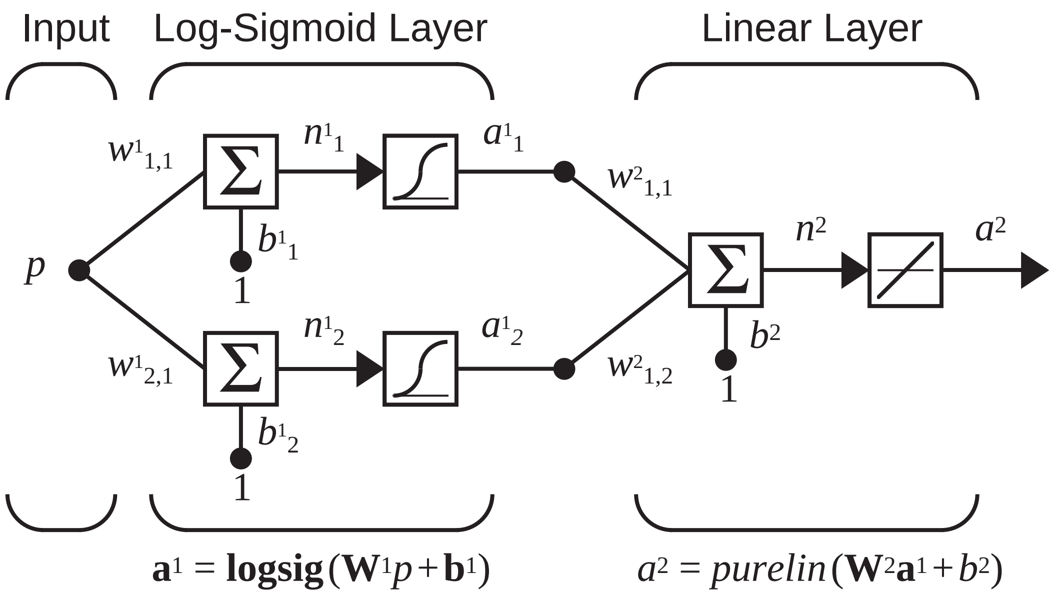
$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad \mathbf{b}^2 = \begin{bmatrix} -1.5 \\ -1.5 \end{bmatrix}$$

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathbf{b}^3 = \begin{bmatrix} -0.5 \end{bmatrix}$$



$\mathbf{a}^1 = \mathbf{hardlim}(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{hardlim}(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{hardlim}(\mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3)$

Function Approximation Example



$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$ $\quad a^2 = purelin(\mathbf{W}^2 \mathbf{a}^1 + b^2)$

$$f^1(n) = \frac{1}{1 + e^{-n}}$$

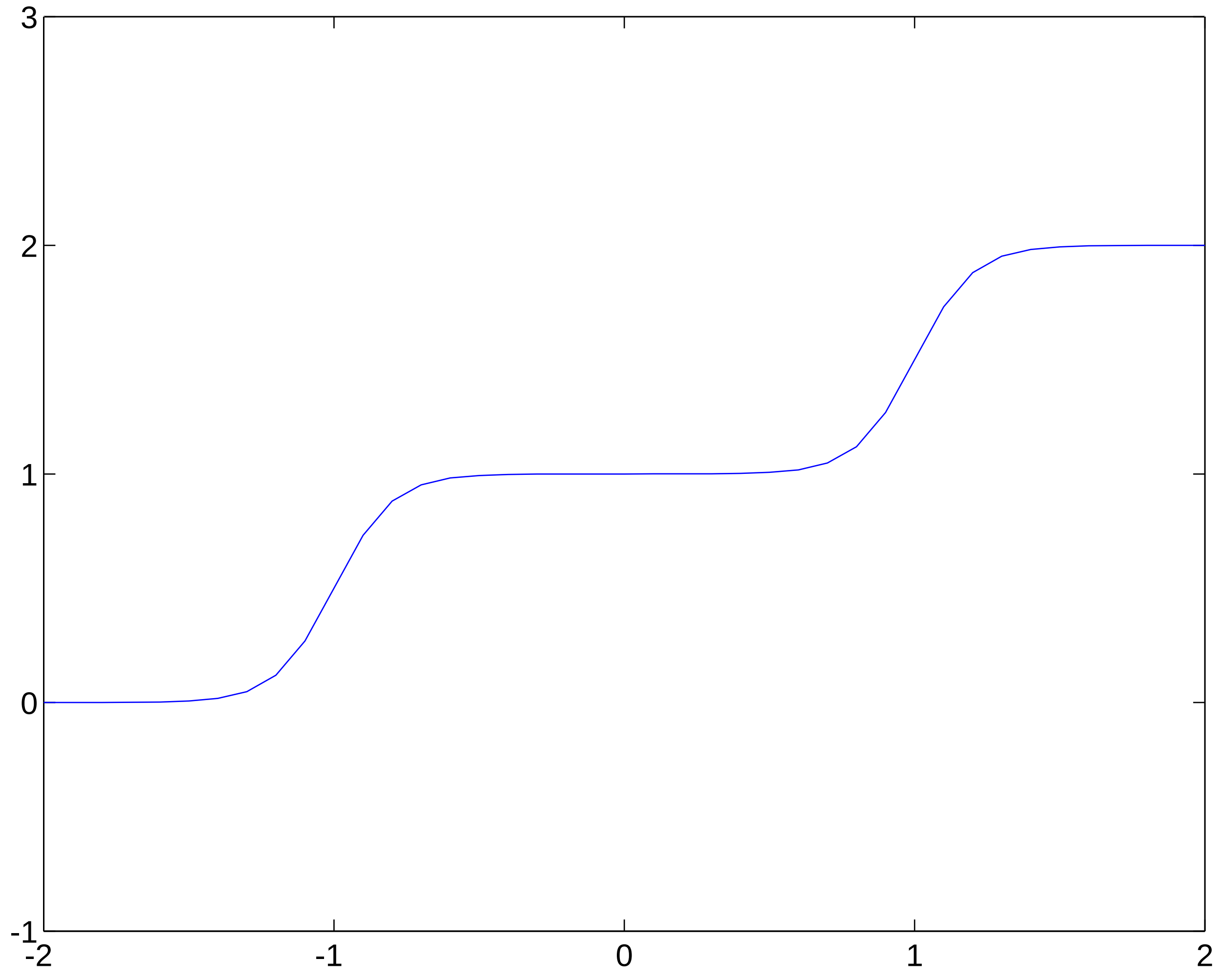
$$f^2(n) = n$$

Nominal Parameter Values

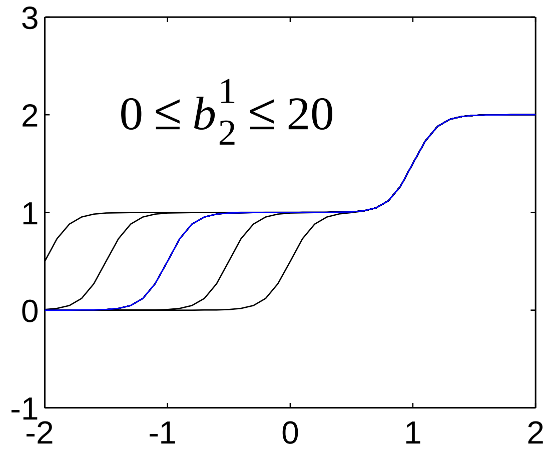
$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -10 \qquad b^1_2 = 10$

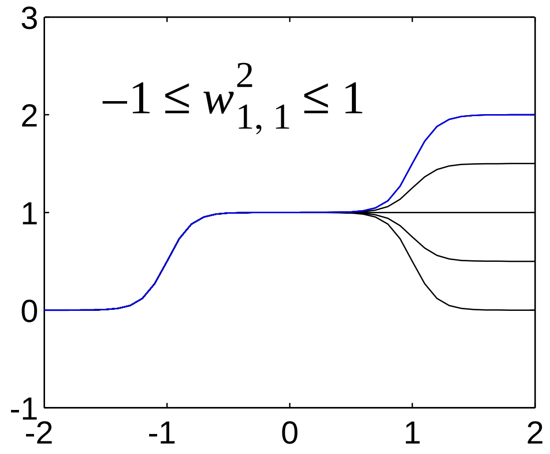
$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = 0$

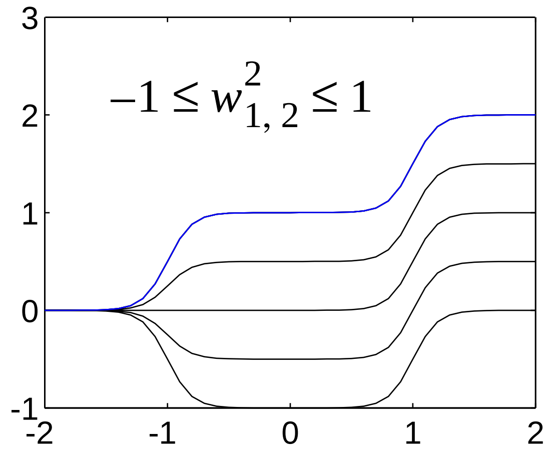
Nominal Response

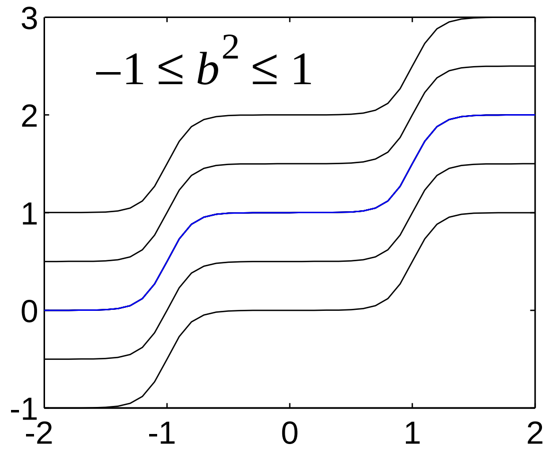


Parameter Variations









Multilayer Network

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a}^{0} = \mathbf{p}$$

$$\mathbf{a} = \mathbf{a}^{M}$$

Performance Index

Training Set

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Mean Square Error

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2]$$

Vector Case

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

Approximate Mean Square Error (Single Sample)

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Approximate Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}} \qquad b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i}$$

Chain Rule

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}$$

Example

$$f(n) = \cos(n) \qquad n = \mathrm{e}^{2w} \qquad f(n(w)) = \cos(e^{2w})$$

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (-\sin(n))(2e^{2w}) = (-\sin(e^{2w}))(2e^{2w})$$

Application to Gradient Calculation

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^{m}} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m}$$

Gradient Calculation

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

Sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

Gradient

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

Steepest Descent

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(\mathrm{k}) - \alpha s_i^m a_j^{m-1} \qquad b_i^m(k+1) = b_i^m(k) - \alpha s_i^m$$

$$\boxed{\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m}$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}$$

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \dots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \dots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \dots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m)$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m(\mathbf{n}^m)$$

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \dot{f}^m(n_{S^m}^m) \end{bmatrix}$$

11

Backpropagation (Sensitivities)

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}}$$

$$\boxed{\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}}$$

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

Initialization (Last Layer)

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M)$$

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M)$$

$$\boxed{\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})}$$

Summary

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Backpropagation

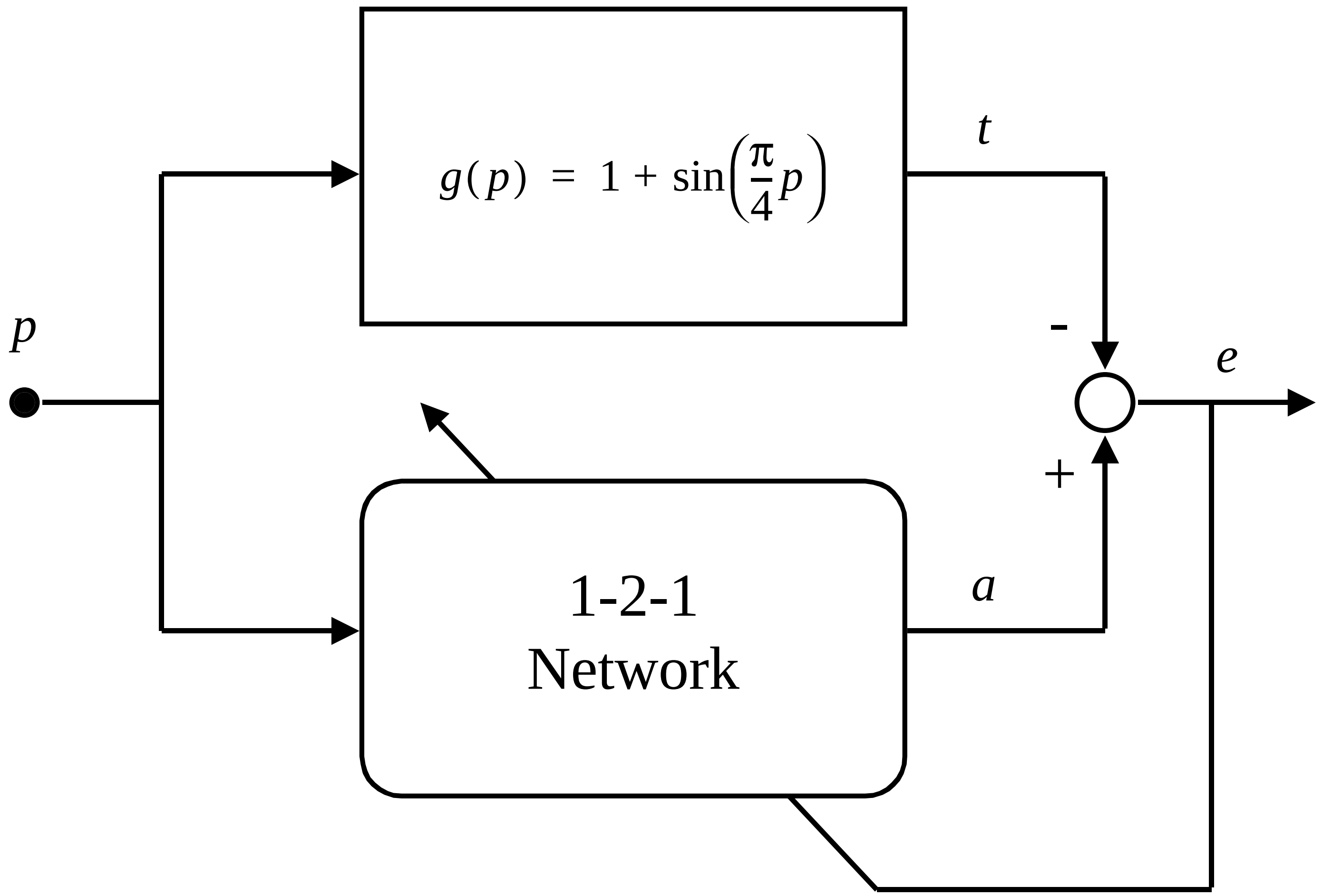
$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1, \ldots, 2, 1$$

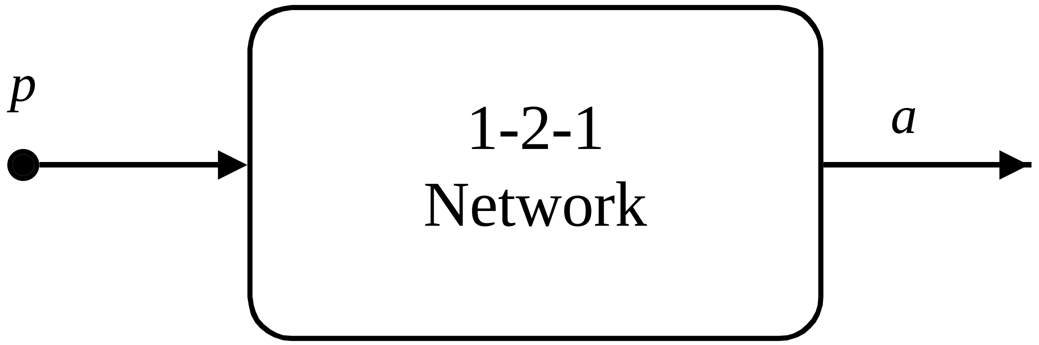
Weight Update

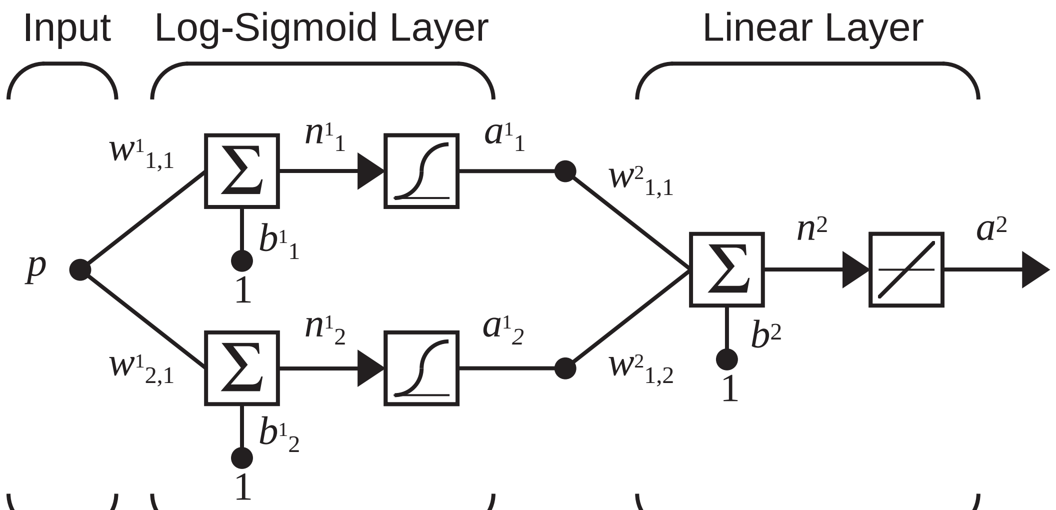
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

Example: Function Approximation



Network

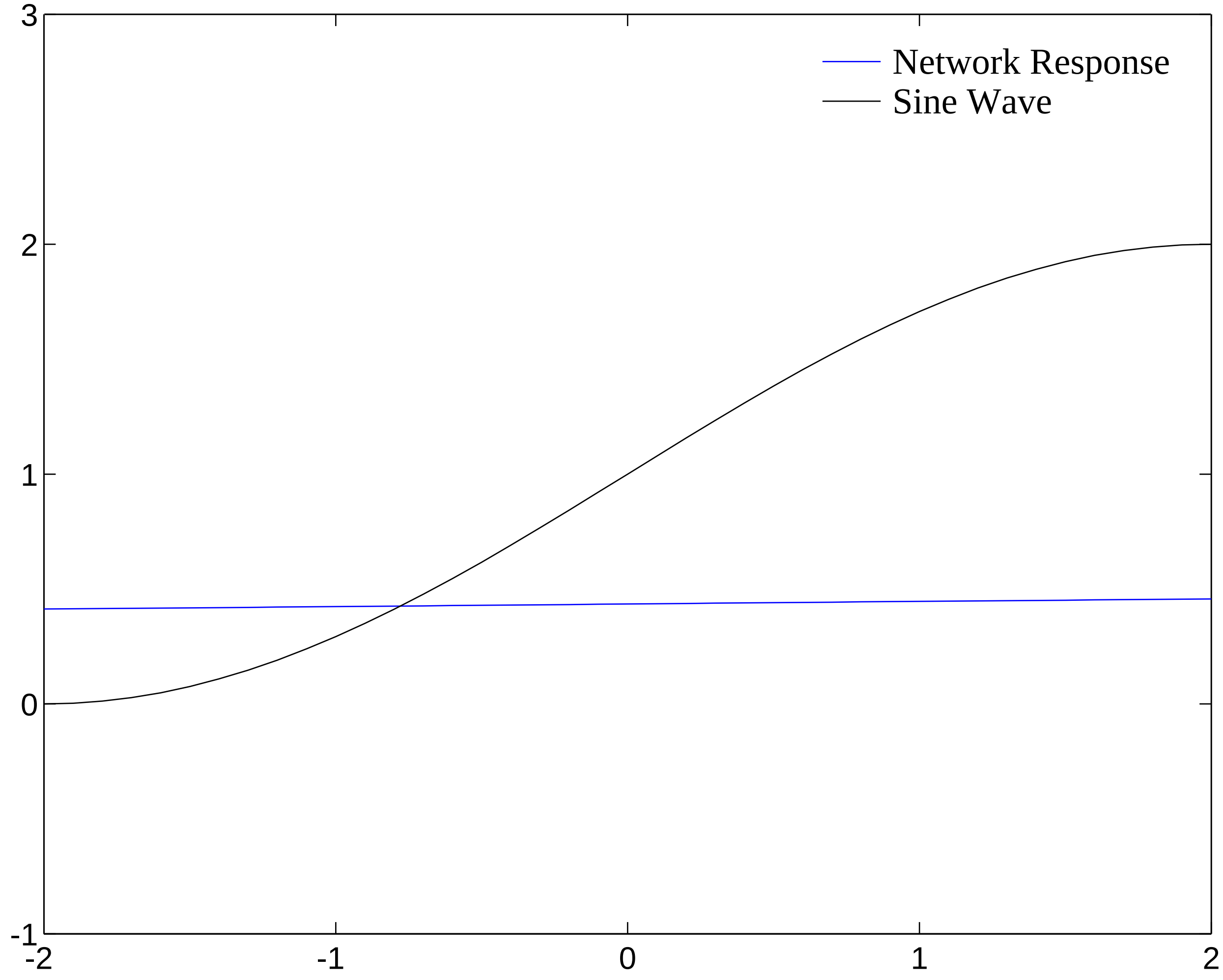




$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$ $a^2 = purelin(\mathbf{W}^2 \mathbf{a}^1 + b^2)$

Initial Conditions

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} \quad \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$



Forward Propagation

$$a^0 = p = 1$$

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix}\begin{bmatrix} 1 \end{bmatrix} + \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix} -0.75 \\ -0.54 \end{bmatrix}\right)$$

$$\mathbf{a}^1 = \begin{bmatrix} \dfrac{1}{1+e^{0.75}} \\ \dfrac{1}{1+e^{0.54}} \end{bmatrix} = \begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix}$$

$$a^2 = \mathrm{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix} 0.09 & -0.17 \end{bmatrix}\begin{bmatrix} 0.321 \\ 0.368 \end{bmatrix} + \begin{bmatrix} 0.48 \end{bmatrix}\right) = \begin{bmatrix} 0.446 \end{bmatrix}$$

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

Transfer Function Derivatives

$$\dot{f}^1(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^2} = \left(1 - \frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^1)(a^1)$$

$$\dot{f}^2(n) = \frac{d}{dn}(n) = 1$$

Backpropagation

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix} (1-a_1^1)(a_1^1) & 0 \\ 0 & (1-a_2^1)(a_2^1) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix} (1-0.321)(0.321) & 0 \\ 0 & (1-0.368)(0.368) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix} 0.218 & 0 \\ 0 & 0.233 \end{bmatrix} \begin{bmatrix} -0.227 \\ 0.429 \end{bmatrix} = \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix}$$

Weight Update

$$\alpha = 0.1$$

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha\mathbf{s}^2(\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1\begin{bmatrix} -2.522 \end{bmatrix}\begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

$$\mathbf{W}^2(1) = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha\mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1\begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix}$$

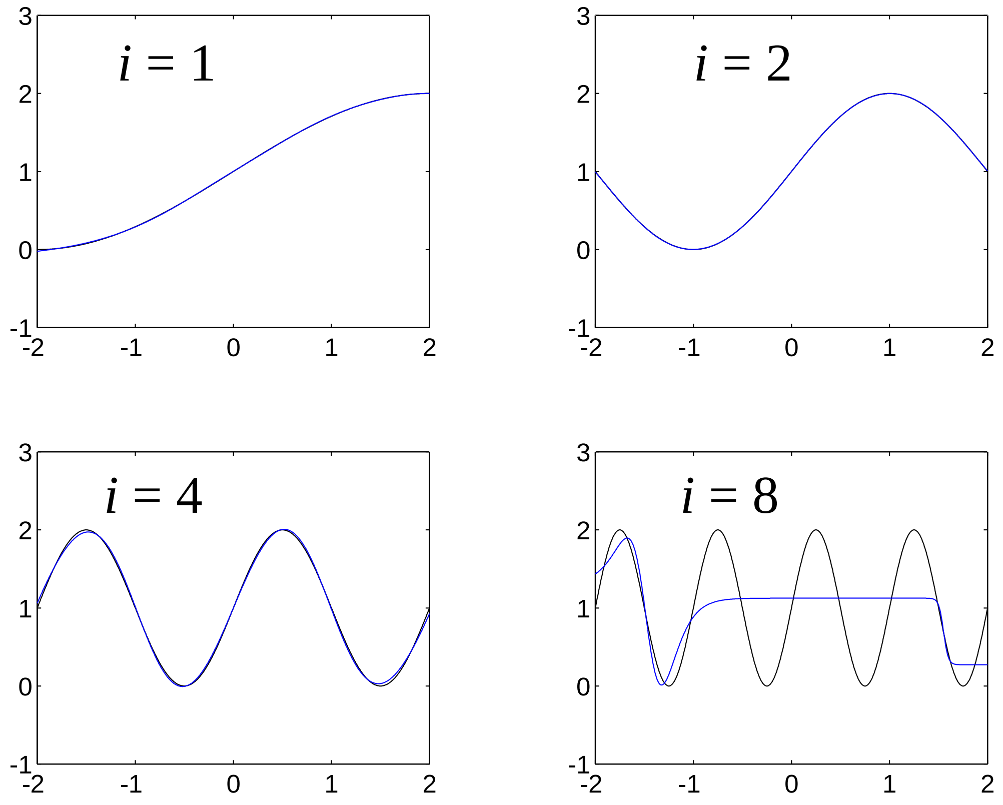
$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha\mathbf{s}^1(\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1\begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix}\begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha\mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1\begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}$$

Choice of Architecture

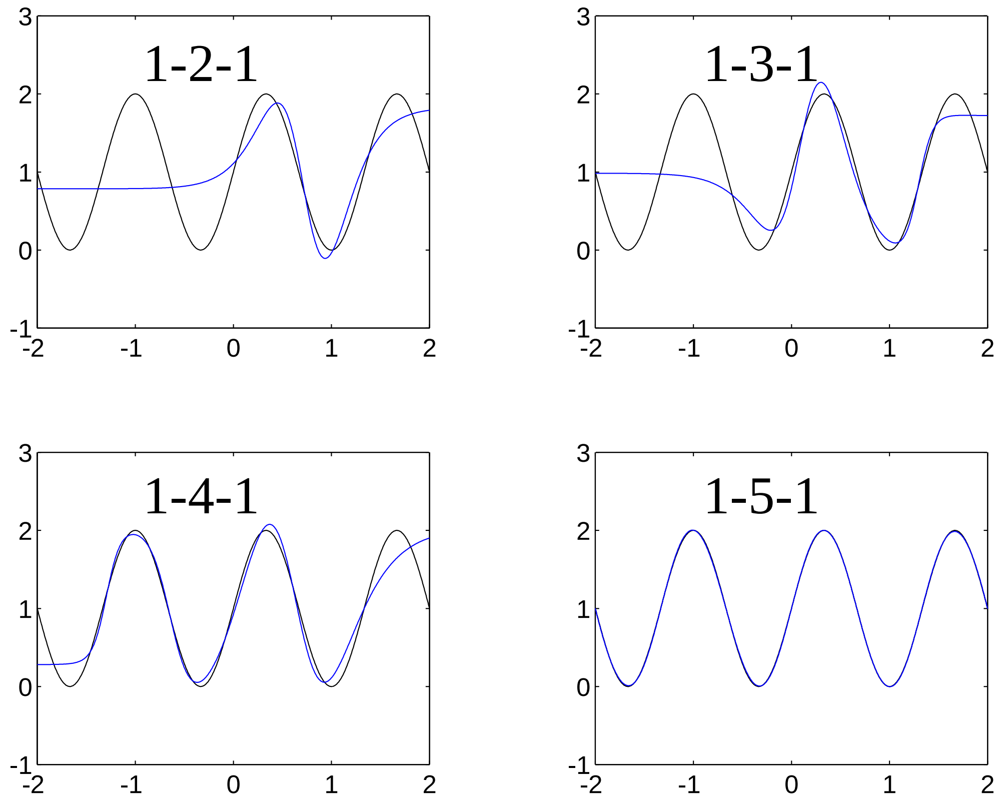
$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right)$$

1-3-1 Network



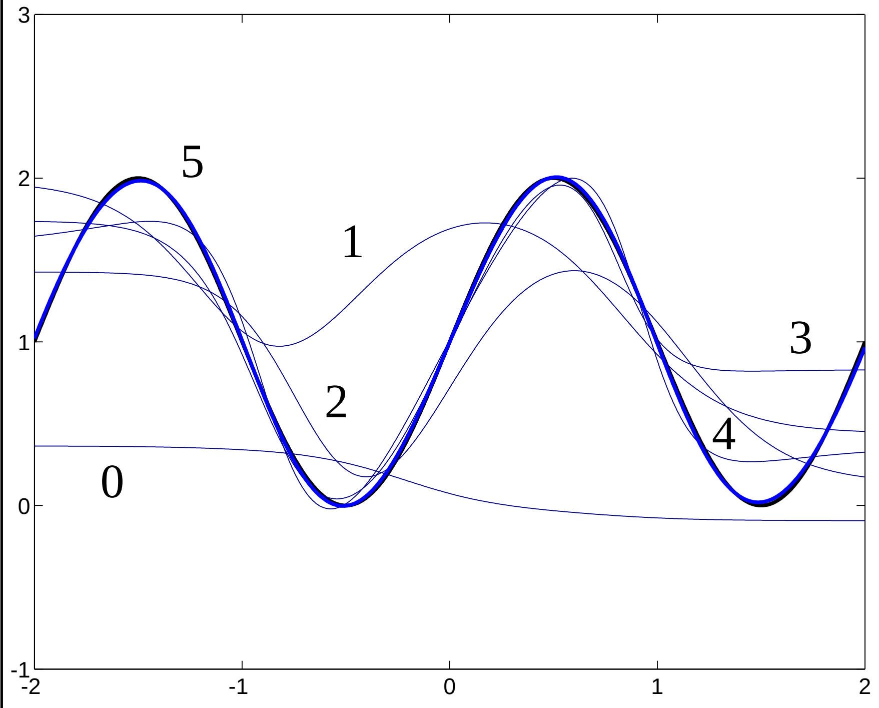
Choice of Network Architecture

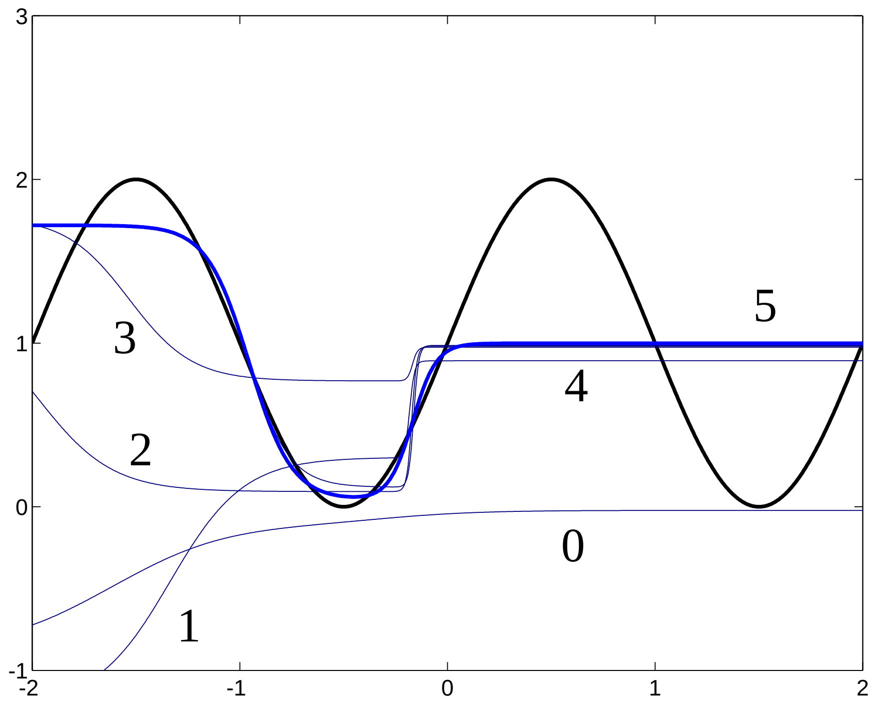
$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right)$$



Convergence

$$g(p) = 1 + \sin(\pi p)$$

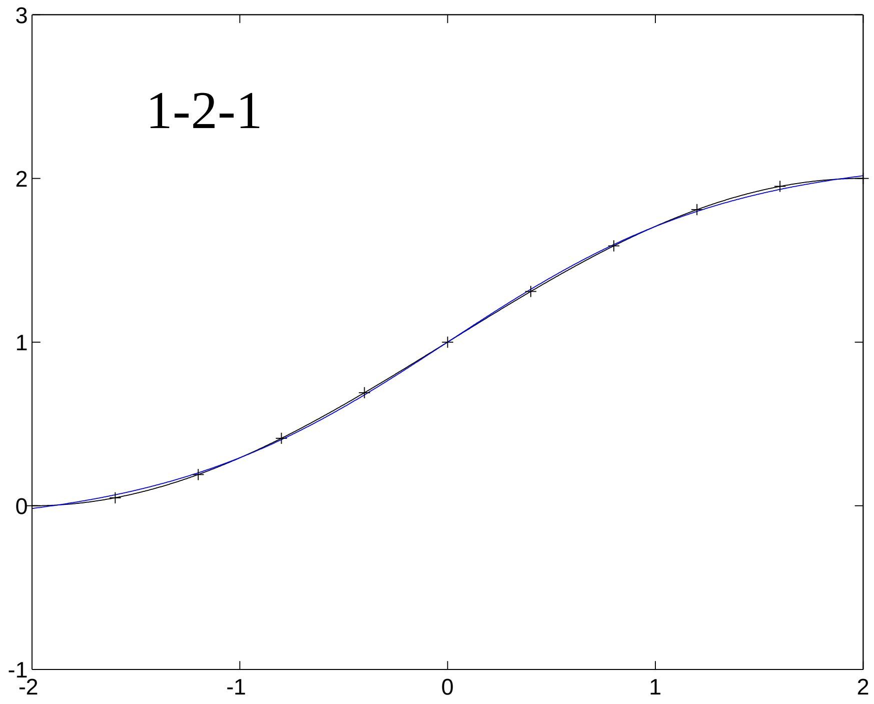


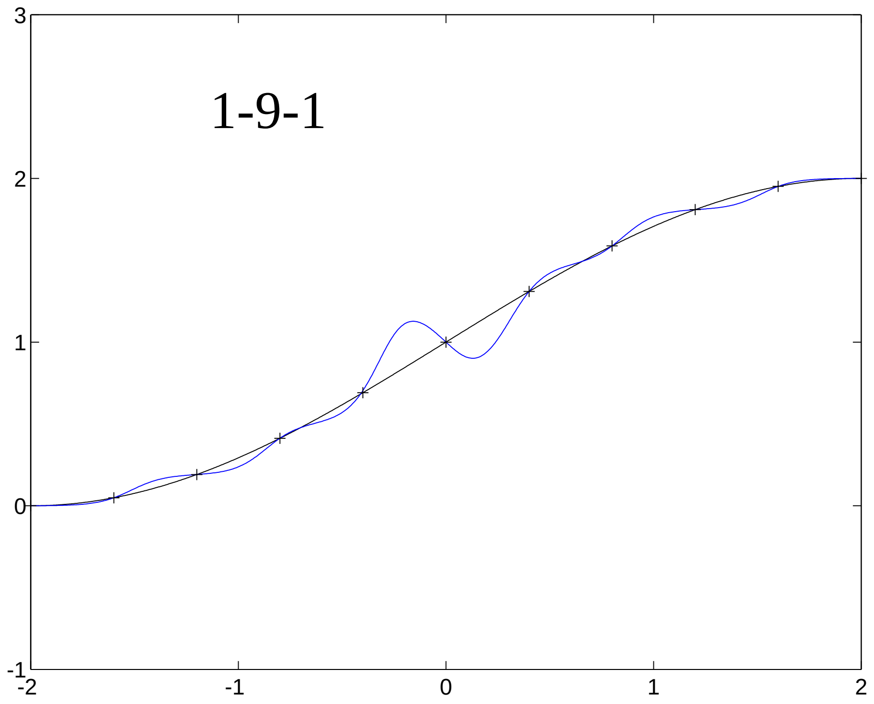


Generalization

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \qquad p = -2, -1.6, -1.2, \ldots, 1.6, 2$$





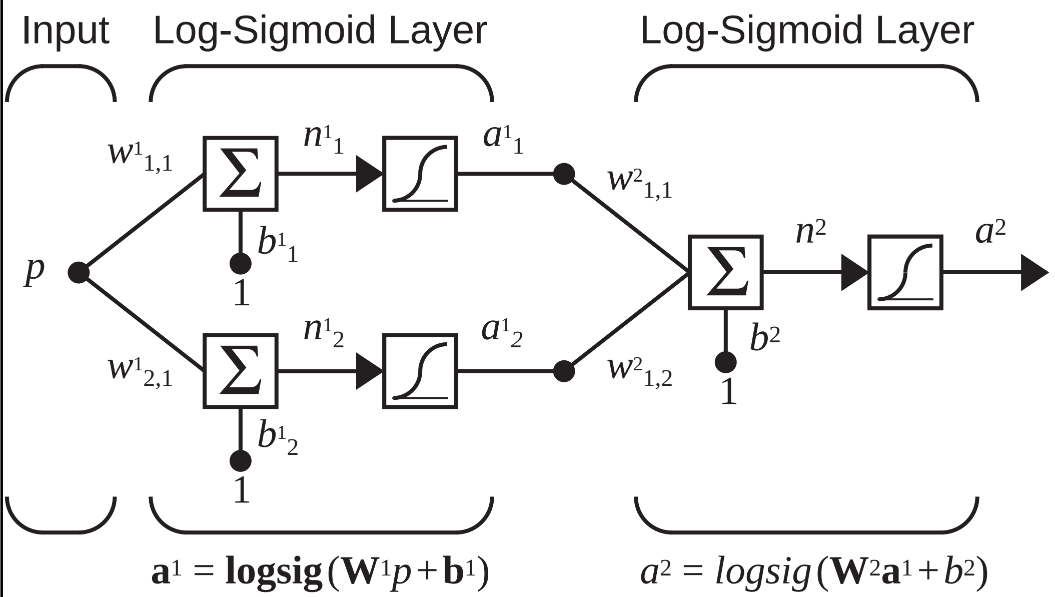
Variations on Backpropagation

Variations

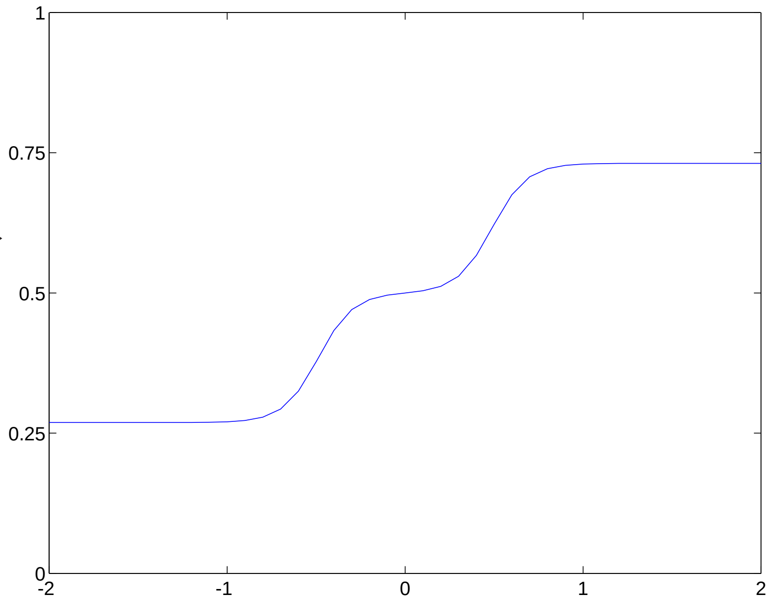
- Heuristic Modifications
  - Momentum
  - Variable Learning Rate
- Standard Numerical Optimization
  - Conjugate Gradient
  - Newton’s Method (Levenberg-Marquardt)

Performance Surface Example

Network Architecture



Nominal Function

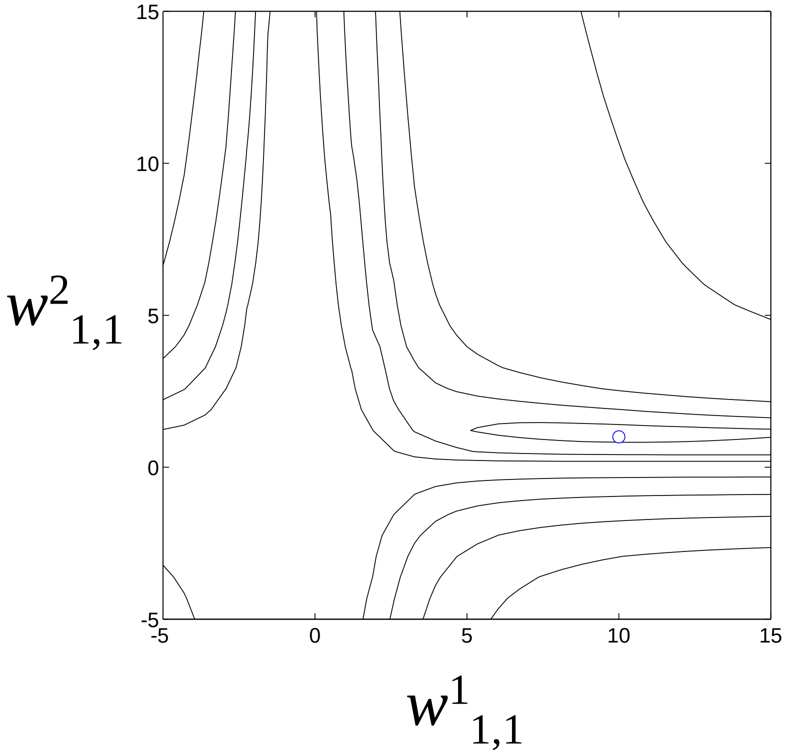


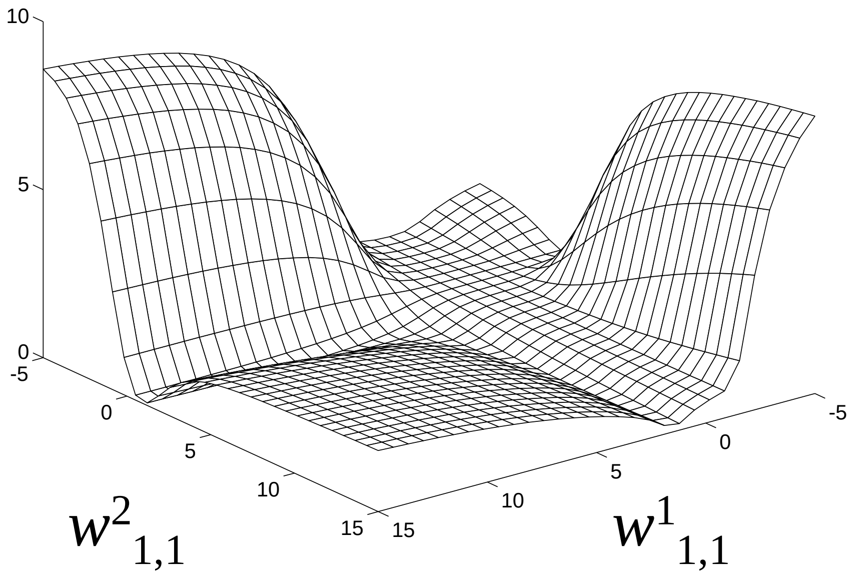
Parameter Values

$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -5 \qquad b^1_2 = 5$$

$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = -1$$

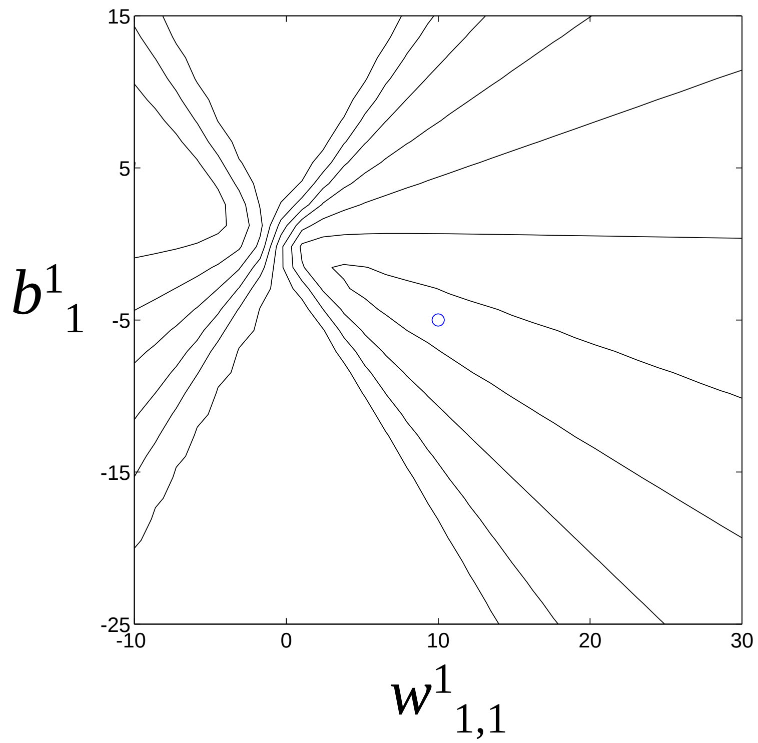
Squared Error vs. $w^1_{1,1}$ and $w^2_{1,1}$

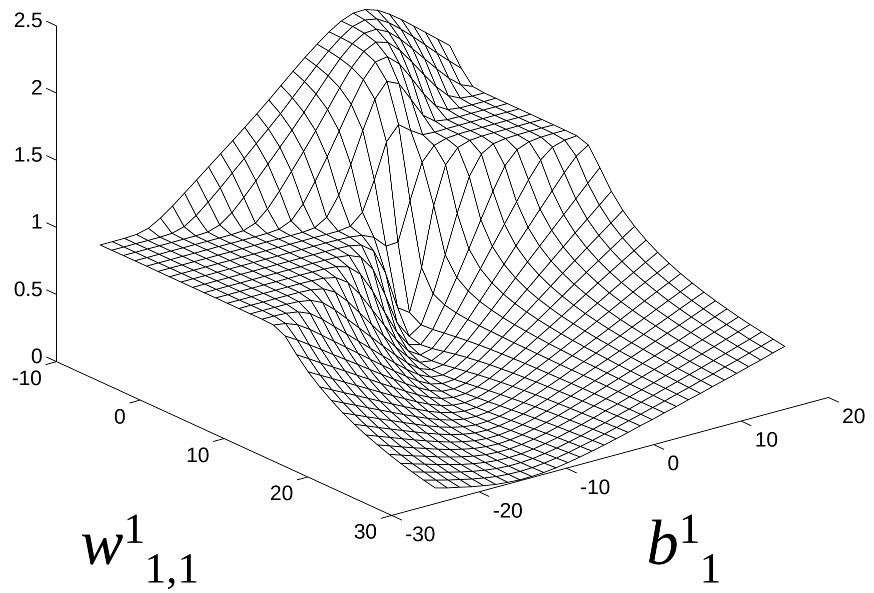




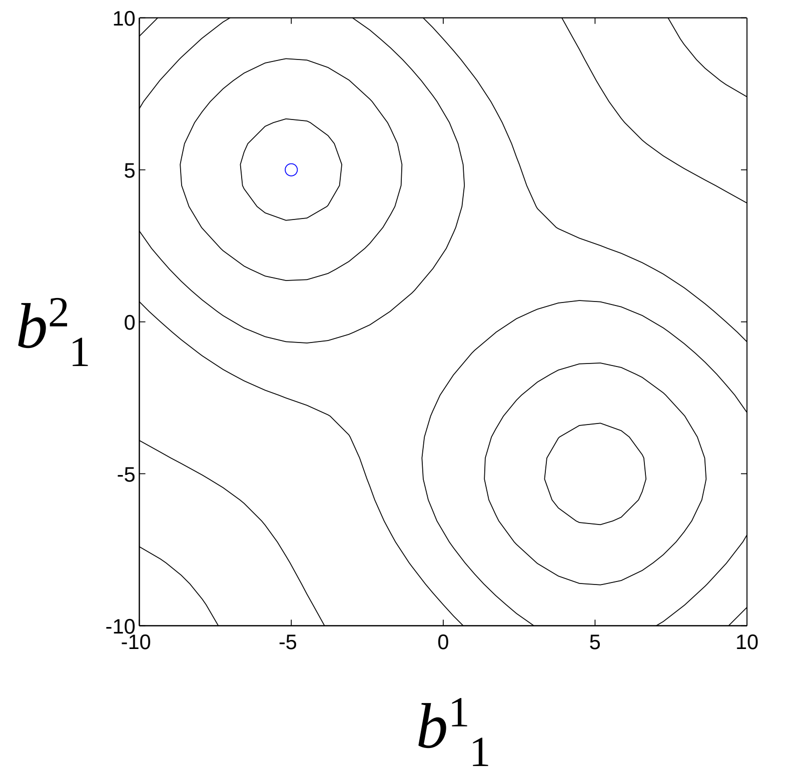
12

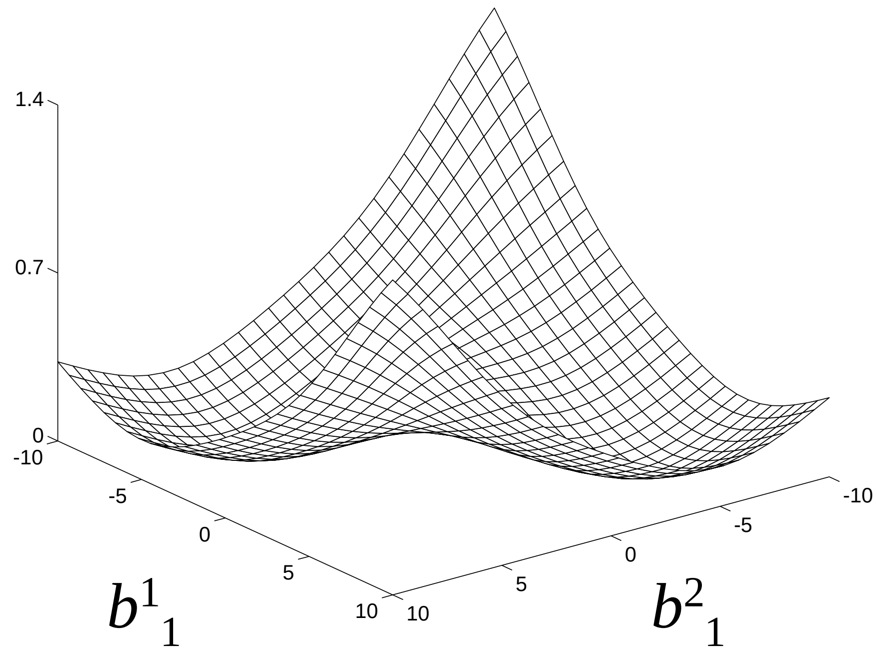
Squared Error vs. $w^1_{1,1}$ and b^1_1





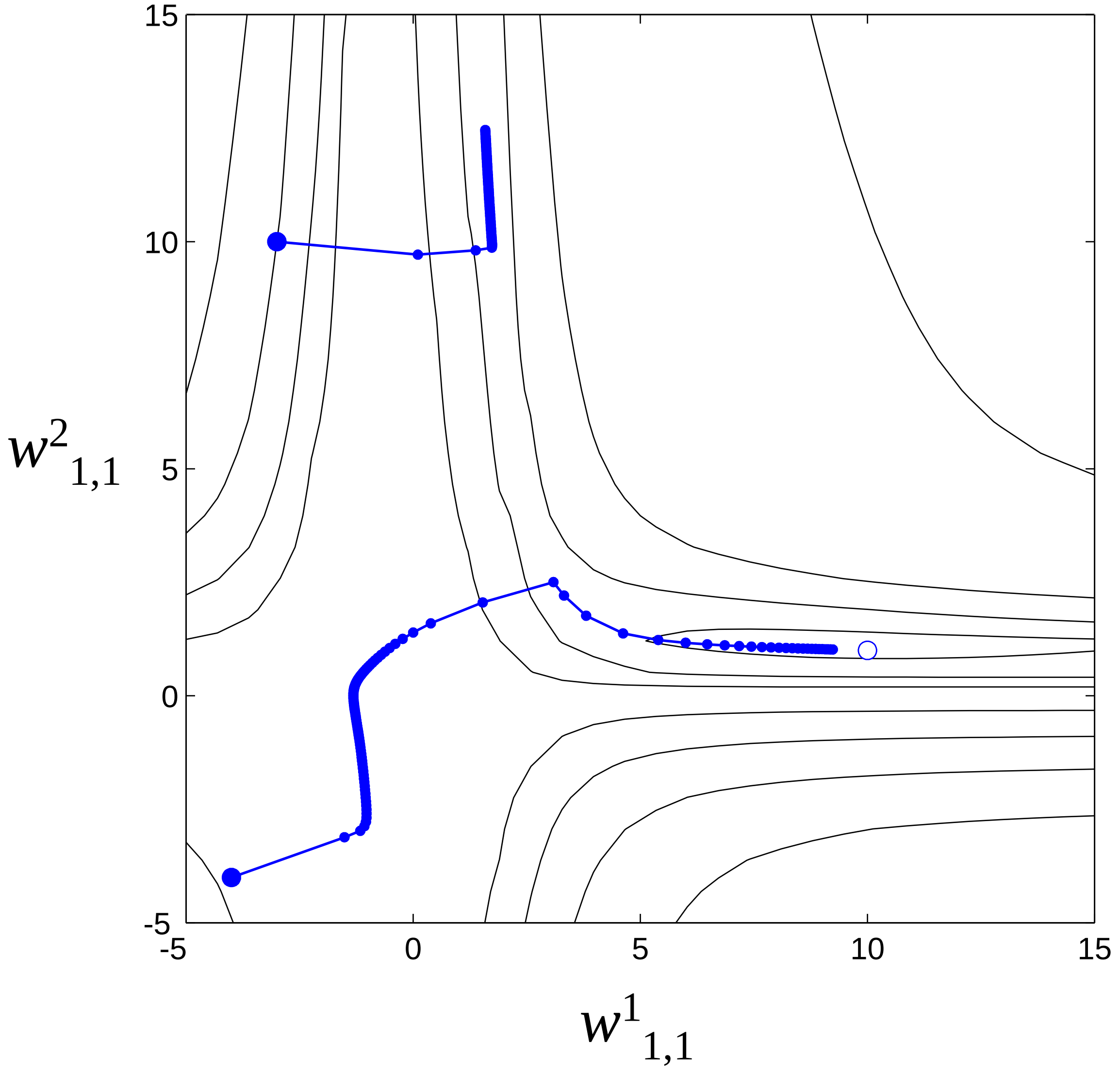
Squared Error vs. b^1_1 and b^1_2



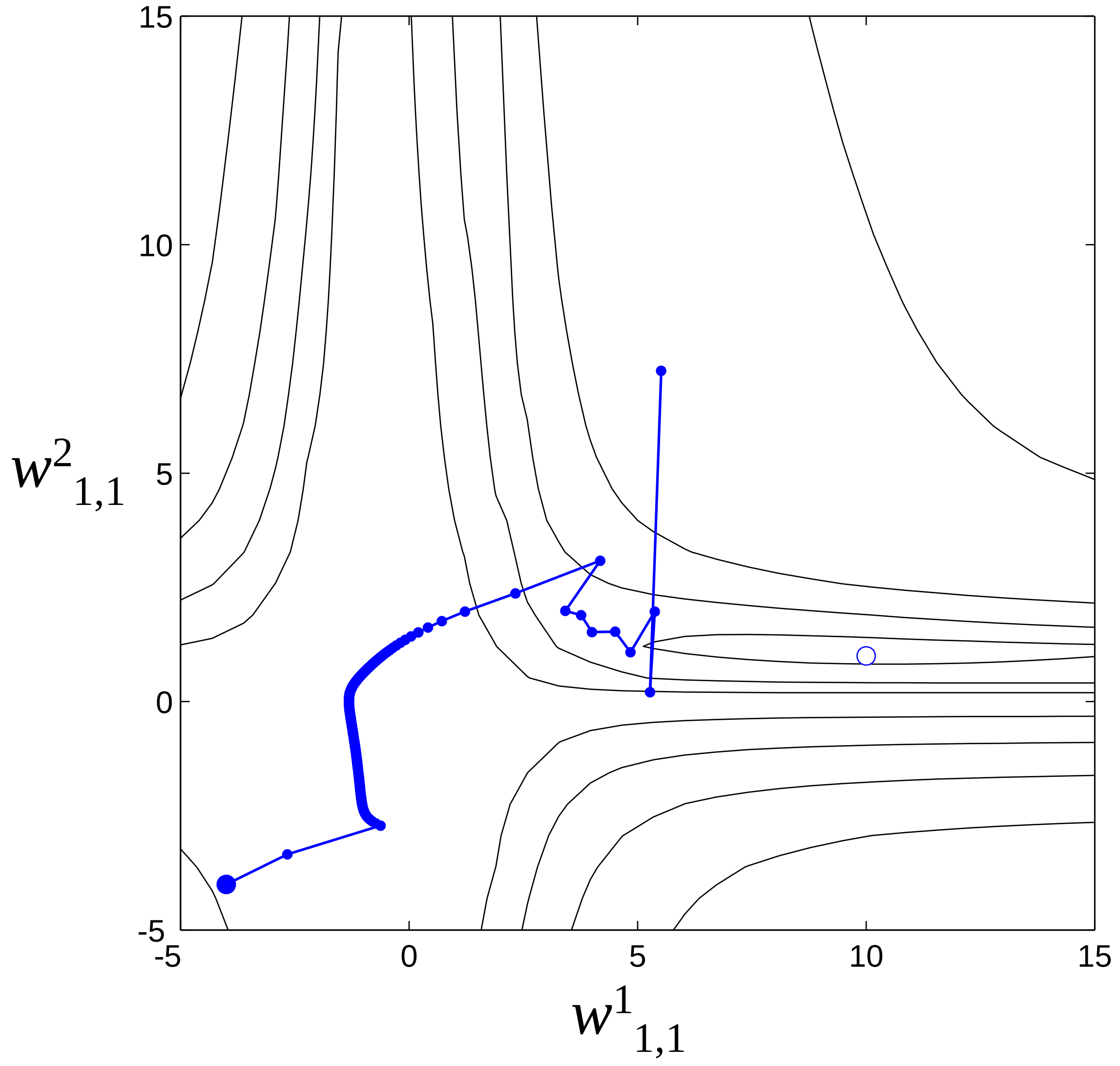


12

Convergence Example



Learning Rate Too Large



12

Momentum

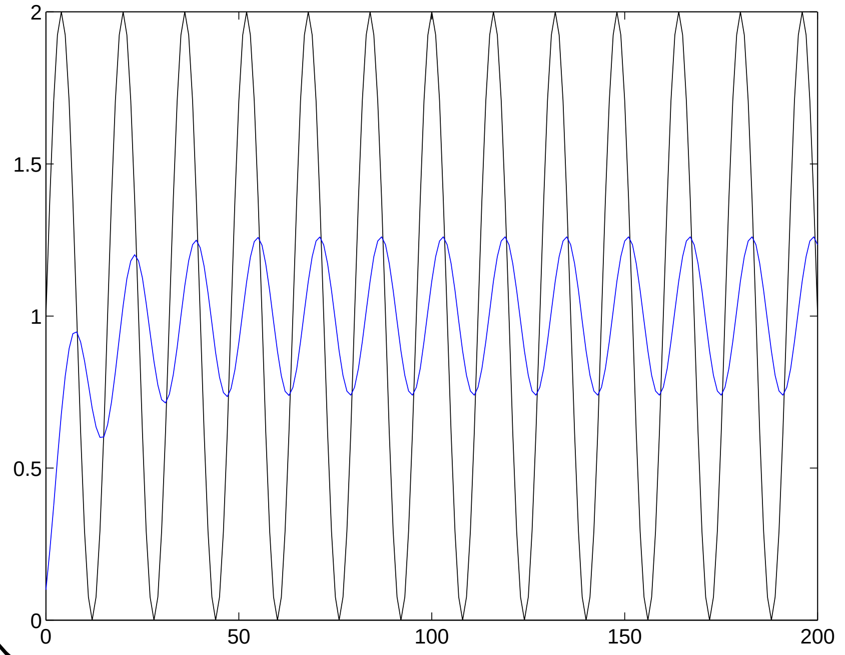
Filter

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k) \qquad 0 \le \gamma < 1$$

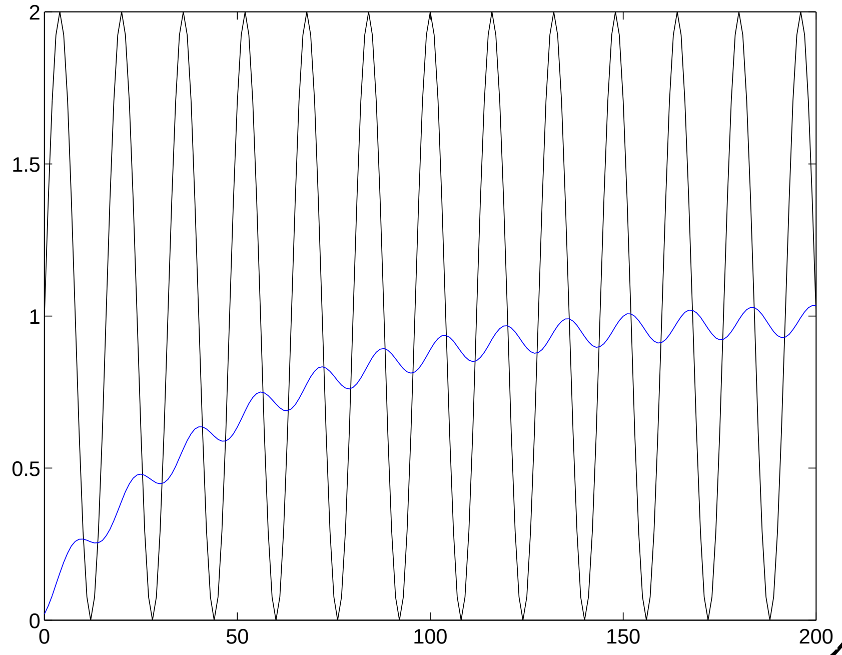
Example

$$w(k) = 1 + \sin\left(\frac{2\pi k}{16}\right)$$

$\gamma = 0.9$



$\gamma = 0.98$



Momentum Backpropagation

Steepest Descent Backpropagation
(SDBP)

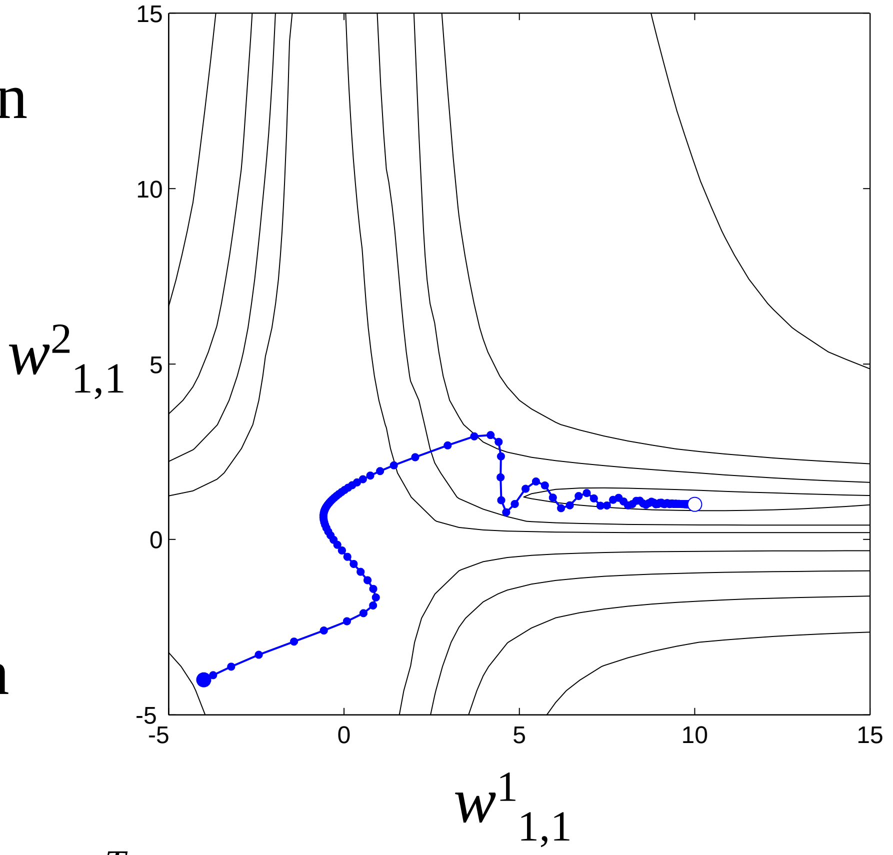
$$\Delta \mathbf{W}^m(k) = -\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m$$

Momentum Backpropagation
(MOBP)

$$\Delta \mathbf{W}^m(k) = \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m$$

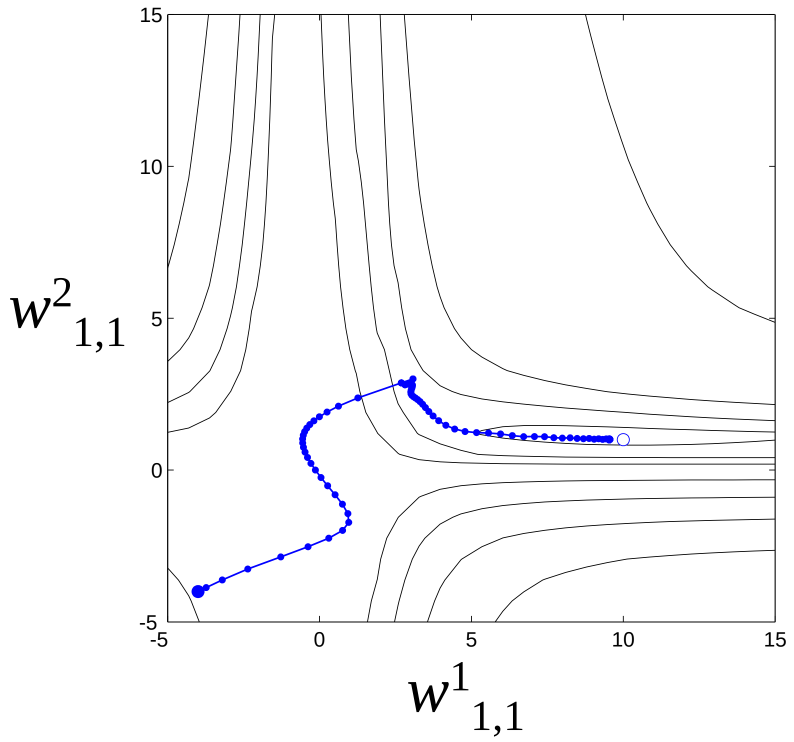


$\gamma = 0.8$

Variable Learning Rate (VLBP)

- If the squared error (over the entire training set) increases by more than some set percentage ζ after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $(1 > \rho > 0)$, and the momentum coefficient γ is set to zero.
- If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If γ has been previously set to zero, it is reset to its original value.
- If the squared error increases by less than ζ, then the weight update is accepted, but the learning rate and the momentum coefficient are unchanged.

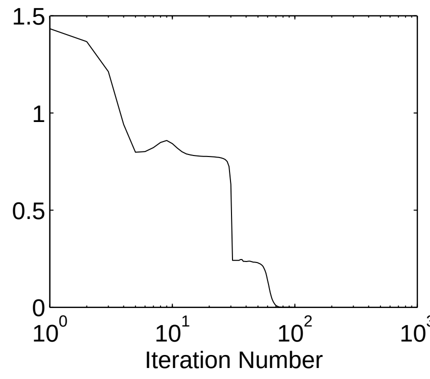
Example

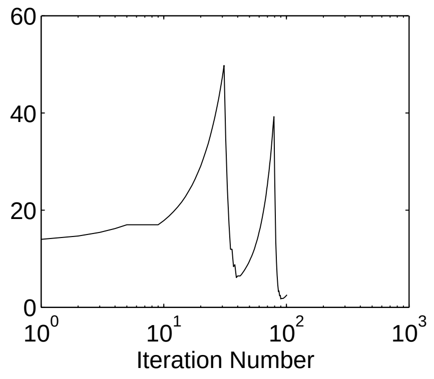


$\eta = 1.05$

$\rho = 0.7$

$\zeta = 4\%$





Conjugate Gradient

1. The first search direction is steepest descent.

$$\mathbf{p}_0 = -\mathbf{g}_0 \qquad \mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_k}$$

2. Take a step and choose the learning rate to minimize the function along the search direction.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

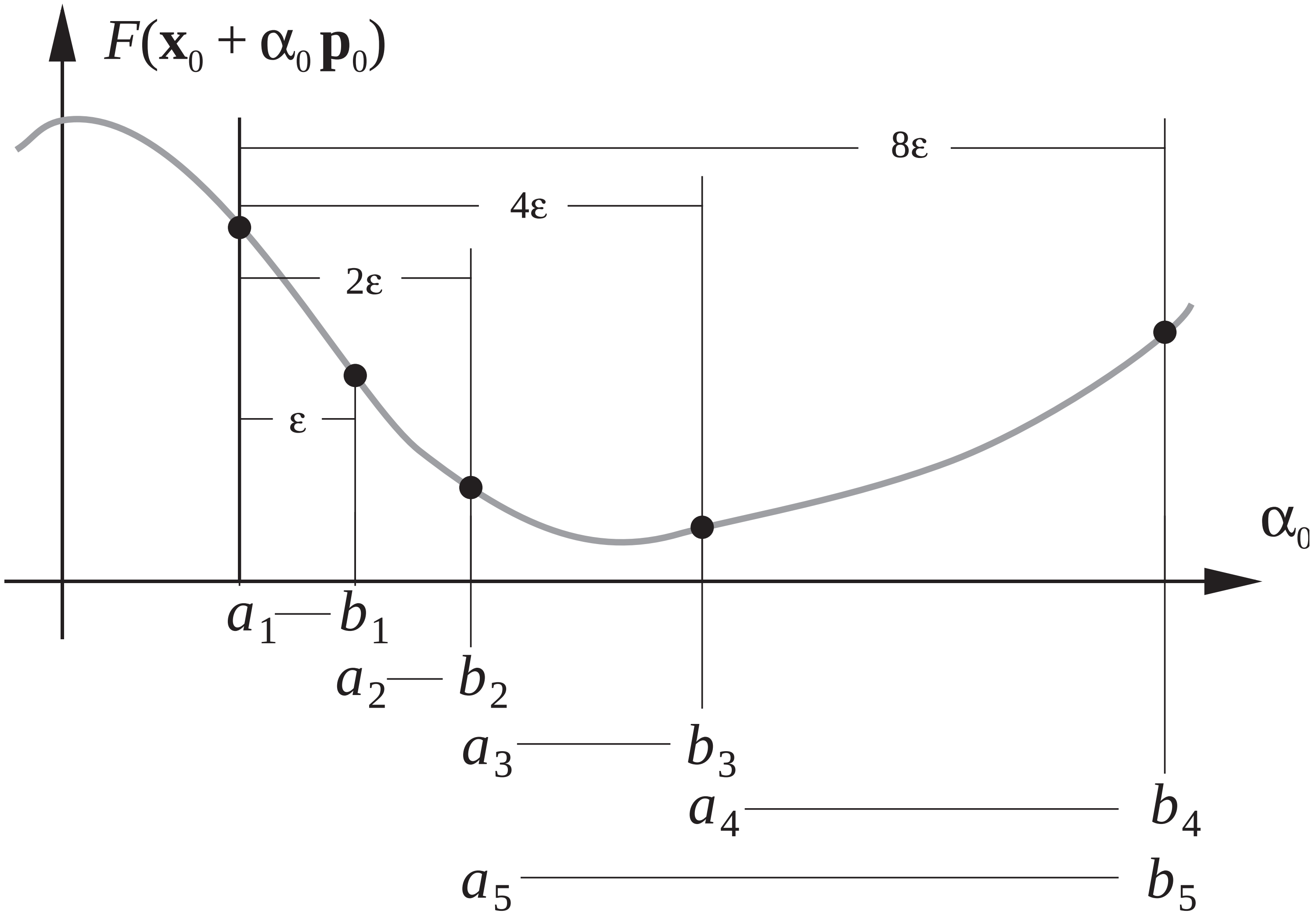
3. Select the next search direction according to:

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

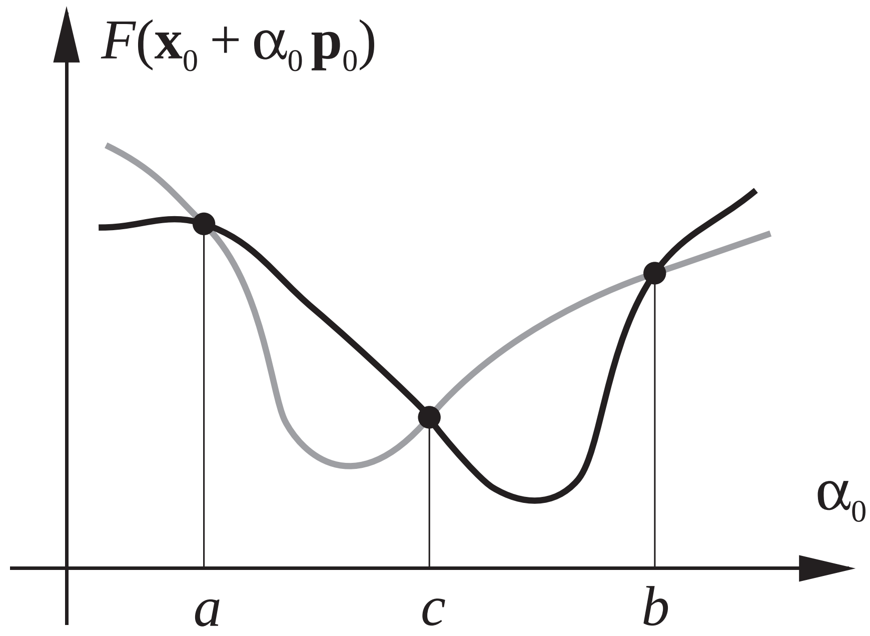
where

$$\beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta\mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

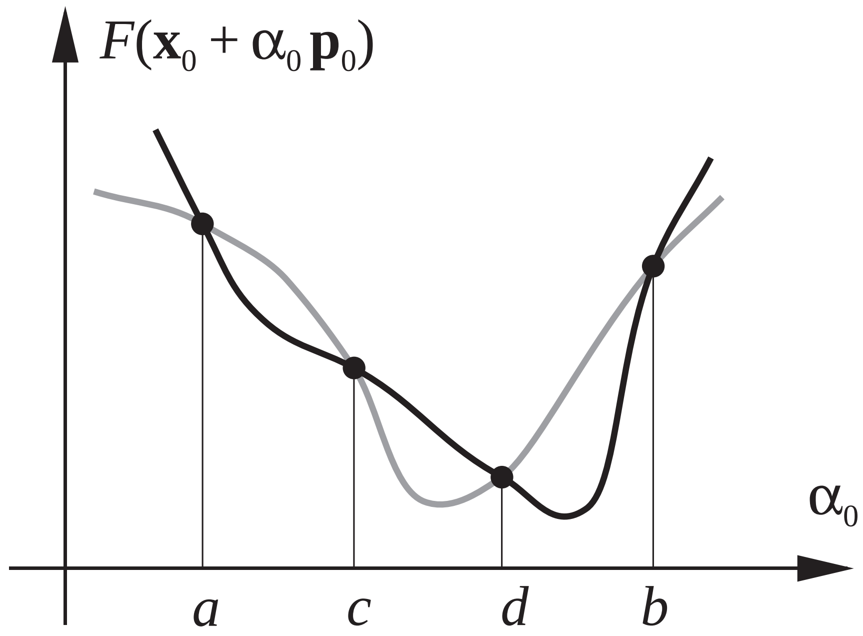
Interval Location



Interval Reduction



(a) Interval is not reduced.



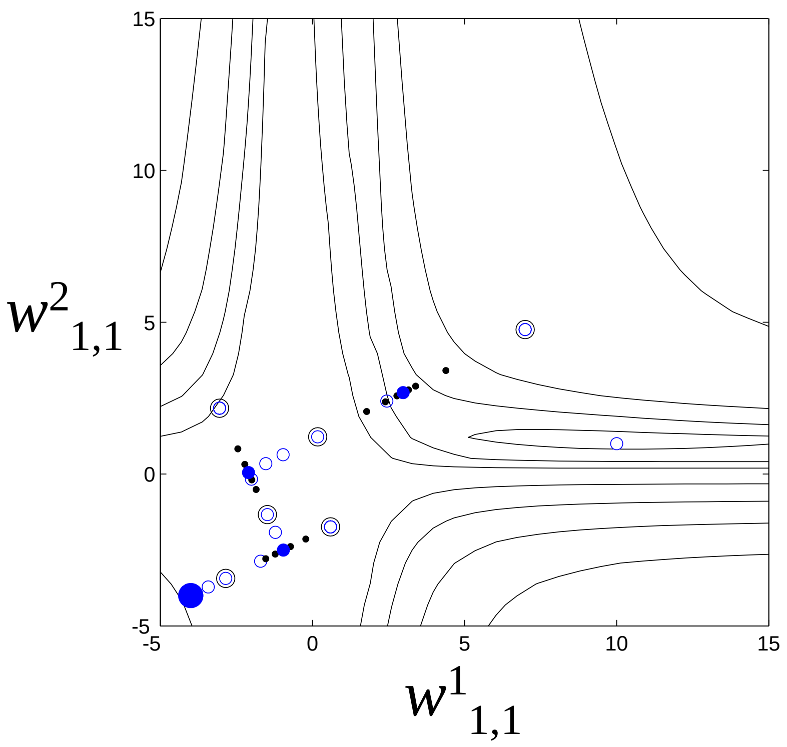
(b) Minimum must occur between *c* and *b*.

Golden Section Search

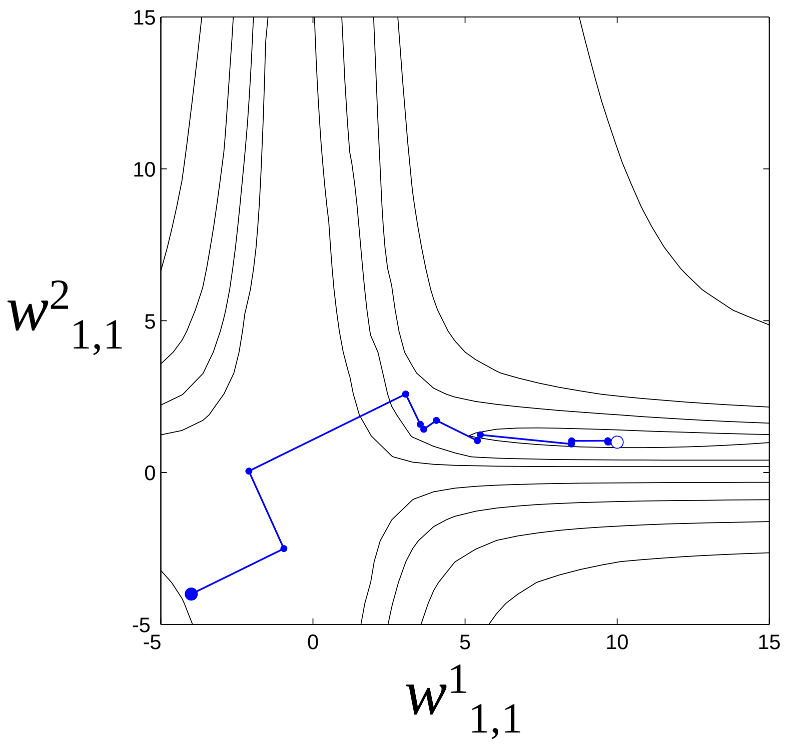
```
$\tau$=0.618
Set    $c_1 = a_1 + (1-\tau)(b_1-a_1)$, $F_c=F(c_1)$
       $d_1 = b_1 - (1-\tau)(b_1-a_1)$, $F_d=F(d_1)$
For $k$=1,2, ... repeat
       If $F_c < F_d$ then
              Set    $a_{k+1} = a_k$ ; $b_{k+1} = d_k$ ; $d_{k+1} = c_k$
                     $c_{k+1} = a_{k+1} + (1-\tau)(b_{k+1} - a_{k+1})$
                     $F_d = F_c$; $F_c=F(c_{k+1})$
       else
              Set    $a_{k+1} = c_k$ ; $b_{k+1} = b_k$ ; $c_{k+1} = d_k$
                     $d_{k+1} = b_{k+1} - (1-\tau)(b_{k+1} - a_{k+1})$
                     $F_c = F_d$; $F_d=F(d_{k+1})$
       end
end until $b_{k+1} - a_{k+1} < tol$
```

Conjugate Gradient BP (CGBP)

Intermediate Steps



Complete Trajectory



Newton's Method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k$$

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k} \qquad \mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

If the performance index is a sum of squares function:

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i^2(\mathbf{x}) = \mathbf{v}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

then the *j*th element of the gradient is

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2\sum_{i=1}^{N} v_i(\mathbf{x})\frac{\partial v_i(\mathbf{x})}{\partial x_j}$$

Matrix Form

The gradient can be written in matrix form:

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

where **J** is the Jacobian matrix:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Hessian

$$[\nabla^2 F(\mathbf{x})]_{k,j} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2\sum_{i=1}^{N}\left\{\frac{\partial v_i(\mathbf{x})}{\partial x_k}\frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x})\frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j}\right\}$$

$$\nabla^2 \mathrm{F}(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x})$$

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^{N} v_i(\mathbf{x})\nabla^2 v_i(\mathbf{x})$$

Gauss-Newton Method

Approximate the Hessian matrix as:

$$\nabla^2 F(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x})$$

Newton's method becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

$$= \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

Levenberg-Marquardt

Gauss-Newton approximates the Hessian by:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

This matrix may be singular, but can be made invertible as follows:

$$\mathbf{G} = \mathbf{H} + \mu\mathbf{I}$$

If the eigenvalues and eigenvectors of **H** are:

$$\{\lambda_1, \lambda_2, \dots, \lambda_n\} \qquad \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$$

then

$$\mathbf{G}\mathbf{z}_i = [\mathbf{H} + \mu\mathbf{I}]\mathbf{z}_i = \mathbf{H}\mathbf{z}_i + \mu\mathbf{z}_i = \lambda_i\mathbf{z}_i + \mu\mathbf{z}_i = \overbrace{(\lambda_i + \mu)}^{\text{Eigenvalues of } \mathbf{G}}\mathbf{z}_i$$

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)}$$

Adjustment of μ_k

As $\mu_k \rightarrow 0$, LM becomes Gauss-Newton.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

As $\mu_k \rightarrow \infty$, LM becomes Steepest Descent with small learning rate.

$$\mathbf{x}_{k+1} \cong \mathbf{x}_k - \frac{1}{\mu_k}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - \frac{1}{2\mu_k}\nabla F(\mathbf{x})$$

Therefore, begin with a small μ_k to use Gauss-Newton and speed convergence. If a step does not yield a smaller $F(\mathbf{x})$, then repeat the step with an increased μ_k until $F(\mathbf{x})$ is decreased. $F(\mathbf{x})$ must decrease eventually, since we will be taking a very small step in the steepest descent direction.

12

Application to Multilayer Network

The performance index for the multilayer network is:

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) = \sum_{q=1}^{Q} \mathbf{e}_q^T \mathbf{e}_q = \sum_{q=1}^{Q} \sum_{j=1}^{S^M} (e_{j,q})^2 = \sum_{i=1}^{N} (v_i)^2$$

The error vector is:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_N \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{2,1} & \dots & e_{S^M,1} & e_{1,2} & \dots & e_{S^M,Q} \end{bmatrix}$$

The parameter vector is:

$$\mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \dots & w^1_{S^1,R} & b^1_1 & \dots & b^1_{S^1} & w^2_{1,1} & \dots & b^M_{S^M} \end{bmatrix}$$

The dimensions of the two vectors are:

$$N = Q \times S^M \qquad n = S^1(R+1) + S^2(S^1+1) + \cdots + S^M(S^{M-1}+1)$$

Jacobian Matrix

$$
\mathbf{J}(\mathbf{x}) = \begin{bmatrix}
\frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} & \ldots & \frac{\partial e_{1,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} & \ldots \\
\frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} & \ldots & \frac{\partial e_{2,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} & \ldots \\
\vdots & \vdots & & \vdots & \vdots & \\
\frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{S^M,1}}{\partial w^1_{1,2}} & \ldots & \frac{\partial \mathrm{e}_{e_{S^M,1}}}{\partial w^1_{S^1,R}} & \frac{\partial \mathrm{e}_{e_{S^M,1}}}{\partial b^1_1} & \ldots \\
\frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \ldots & \frac{\partial e_{1,2}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} & \ldots \\
\vdots & \vdots & & \vdots & \vdots &
\end{bmatrix}
$$

Computing the Jacobian

SDBP computes terms like:

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial x_l} = \frac{\partial \mathbf{e}_q^T \mathbf{e}_q}{\partial x_l}$$

using the chain rule:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m}$$

where the sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

is computed using backpropagation.

For the Jacobian we need to compute terms like:

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial x_l}$$

Marquardt Sensitivity

If we define a Marquardt sensitivity:

$$\tilde{s}_{i,h}^{m} \equiv \frac{\partial v_h}{\partial n_{i,q}^{m}} = \frac{\partial e_{k,q}}{\partial n_{i,q}^{m}} \qquad h = (q-1)S^{M} + k$$

We can compute the Jacobian as follows:

weight

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w_{i,j}^{m}} = \frac{\partial e_{k,q}}{\partial n_{i,q}^{m}} \times \frac{\partial n_{i,q}^{m}}{\partial w_{i,j}^{m}} = \tilde{s}_{i,h}^{m} \times \frac{\partial n_{i,q}^{m}}{\partial w_{i,j}^{m}} = \tilde{s}_{i,h}^{m} \times a_{j,q}^{m-1}$$

bias

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b_{i}^{m}} = \frac{\partial e_{k,q}}{\partial n_{i,q}^{m}} \times \frac{\partial n_{i,q}^{m}}{\partial b_{i}^{m}} = \tilde{s}_{i,h}^{m} \times \frac{\partial n_{i,q}^{m}}{\partial b_{i}^{m}} = \tilde{s}_{i,h}^{m}$$

Computing the Sensitivities

Initialization

$$\tilde{s}^M_{i,h} = \frac{\partial v_h}{\partial n^M_{i,q}} = \frac{\partial e_{k,q}}{\partial n^M_{i,q}} = \frac{\partial(\mathrm{t}_{k,q} - a^M_{k,q})}{\partial n^M_{i,q}} = -\frac{\partial \mathrm{a}^M_{k,q}}{\partial n^M_{i,q}}$$

$$\tilde{s}^M_{i,h} = \begin{cases} -\dot{f}^M(n^M_{i,q}) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{cases}$$

$$\tilde{\mathbf{S}}^M_q = -\dot{\mathbf{F}}^M(\mathbf{n}^M_q)$$

Backpropagation

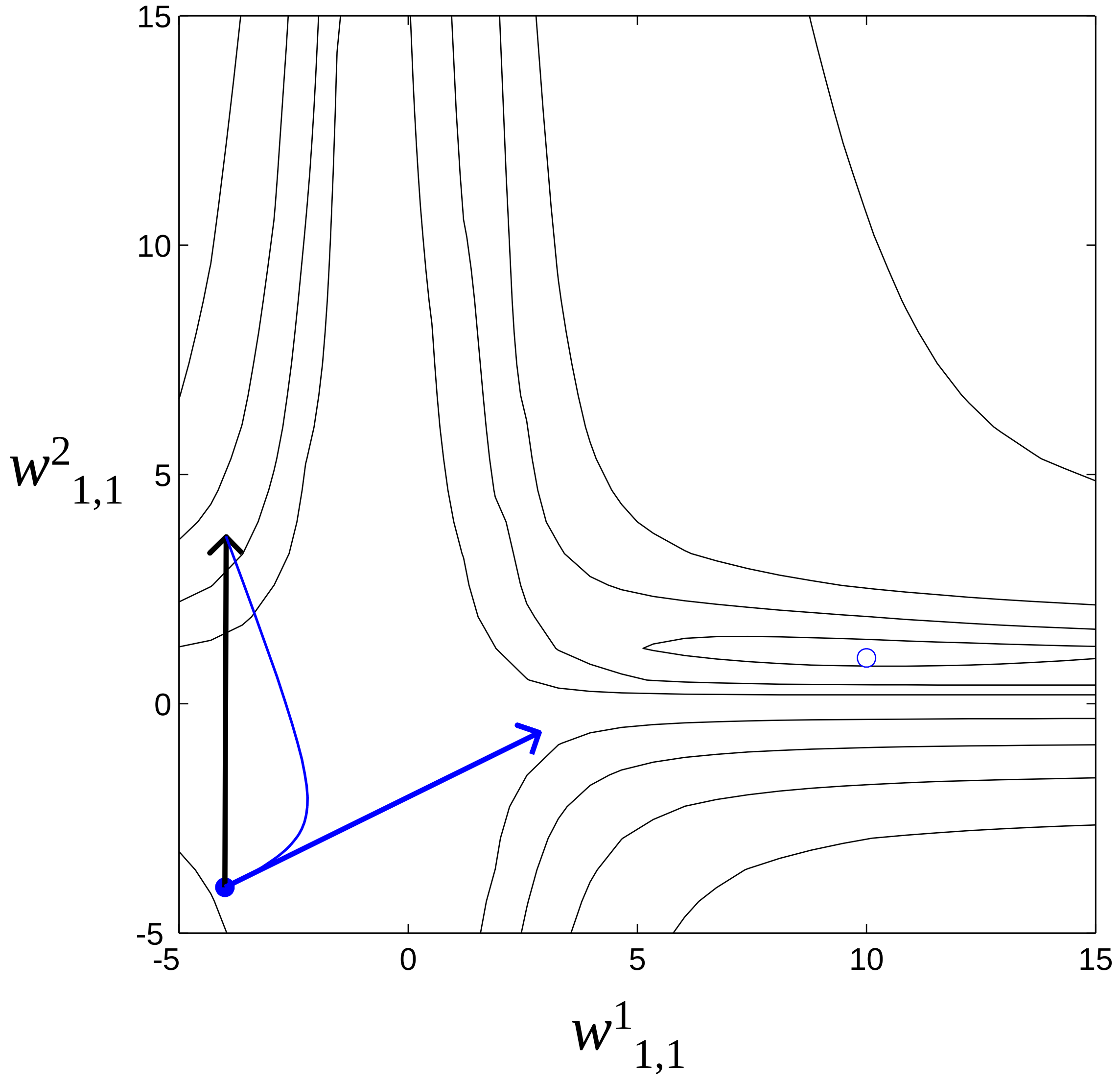
$$\tilde{\mathbf{S}}^m_q = \dot{\mathbf{F}}^m(\mathbf{n}^m_q)(\mathbf{W}^{m+1})^T \tilde{\mathbf{S}}^{m+1}_q$$

$$\tilde{\mathbf{S}}^m = \left[\tilde{\mathbf{S}}^m_1 \middle| \tilde{\mathbf{S}}^m_2 \middle| \dots \middle| \tilde{\mathbf{S}}^m_{\mathrm{Q}} \right]$$

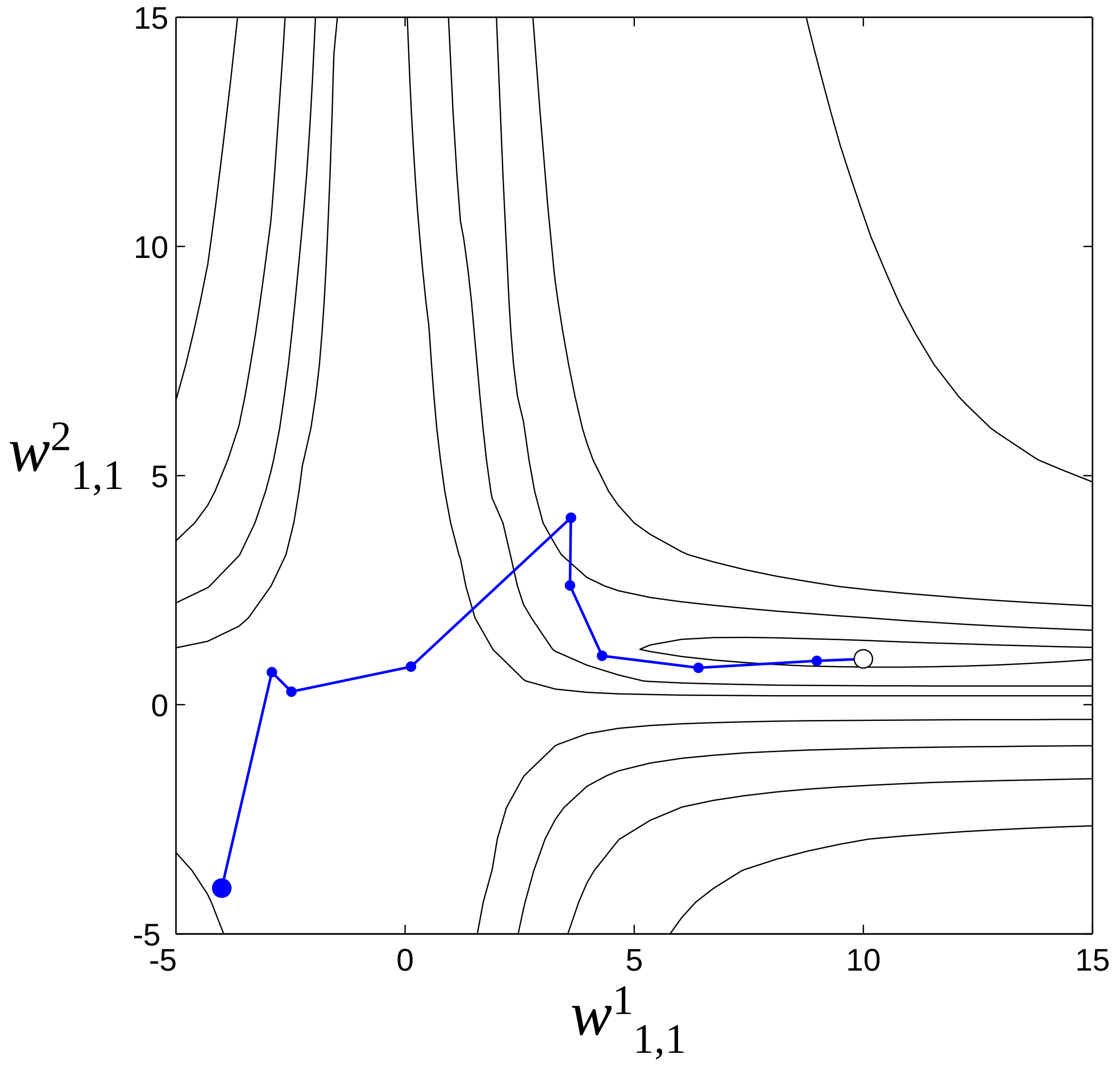
LMBP

- Present all inputs to the network and compute the corresponding network outputs and the errors. Compute the sum of squared errors over all inputs.
- Compute the Jacobian matrix. Calculate the sensitivities with the backpropagation algorithm, after initializing. Augment the individual matrices into the Marquardt sensitivities. Compute the elements of the Jacobian matrix.
- Solve to obtain the change in the weights.
- Recompute the sum of squared errors with the new weights. If this new sum of squares is smaller than that computed in step 1, then divide μ_k by υ, update the weights and go back to step 1. If the sum of squares is not reduced, then multiply μ_k by υ and go back to step 3.

Example LMBP Step



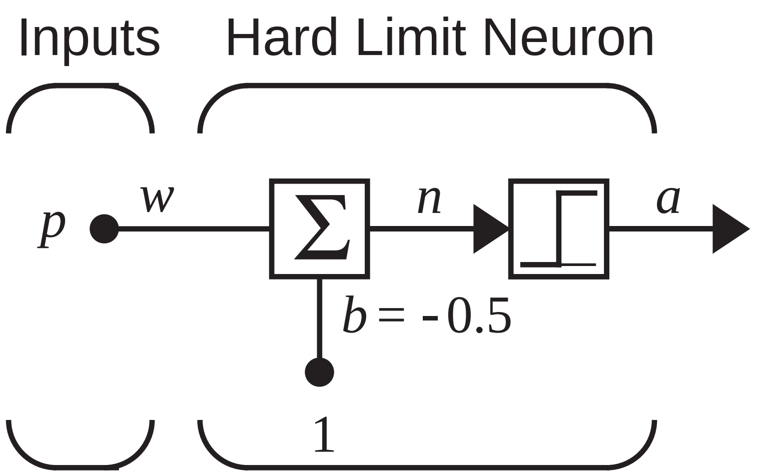
LMBP Trajectory



13

Associative Learning

Simple Associative Network

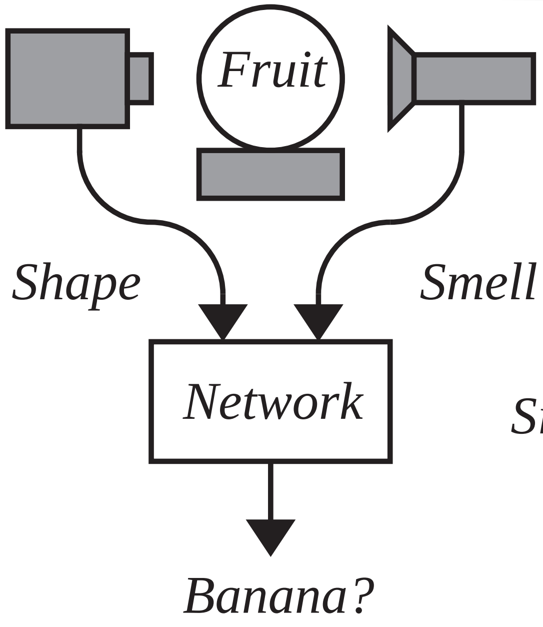


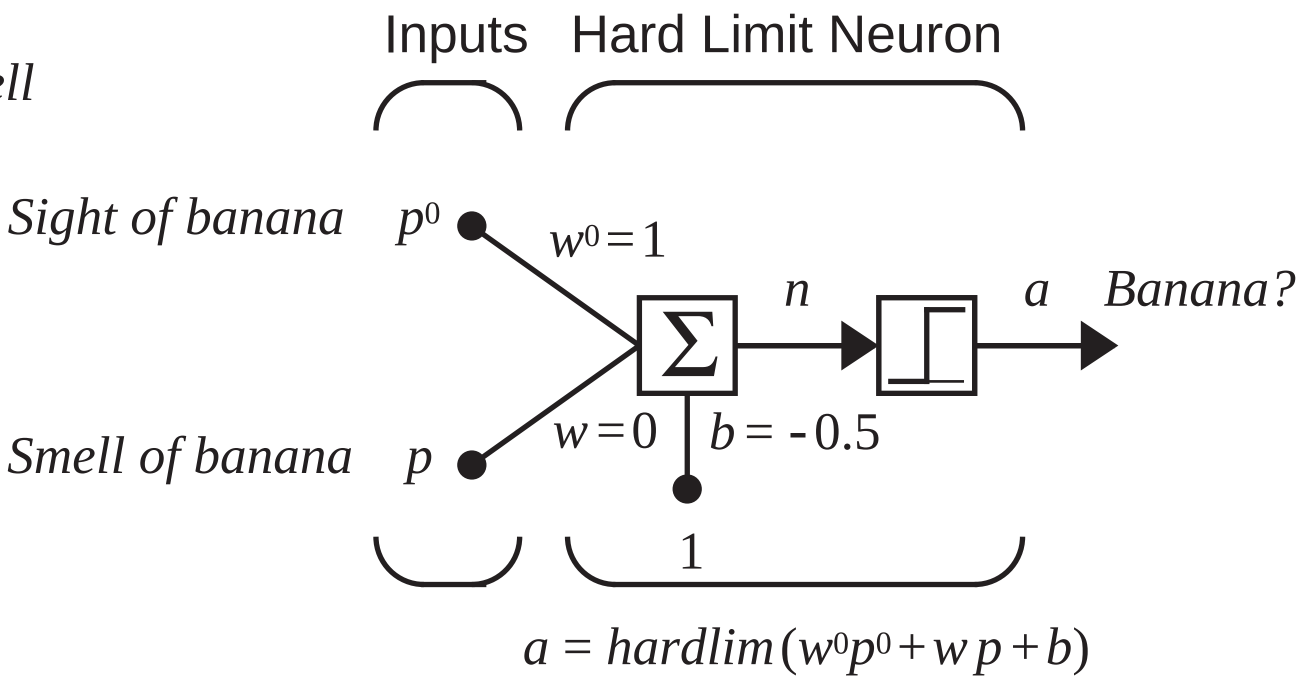
$$a = hardlim(wp + b)$$

$$a = hardlim(wp + b) = hardlim(wp - 0.5)$$

$$p = \begin{cases} 1, & \text{stimulus} \\ 0, & \text{no stimulus} \end{cases} \qquad a = \begin{cases} 1, & \text{response} \\ 0, & \text{no response} \end{cases}$$

Banana Associator





Unconditioned Stimulus

$$p^0 = \begin{cases} 1, & \text{shape detected} \\ 0, & \text{shape not detected} \end{cases}$$

Conditioned Stimulus

$$p = \begin{cases} 1, & \text{smell detected} \\ 0, & \text{smell not detected} \end{cases}$$

Unsupervised Hebb Rule

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q)$$

Vector Form:

$$\boxed{\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q) \mathbf{p}^T(q)}$$

Training Sequence:

$$\mathbf{p}(1), \mathbf{p}(2), \dots, \mathbf{p}(Q)$$

Banana Recognition Example

Initial Weights:

$$w^0 = 1, w(0) = 0$$

Training Sequence:

$$\{p^0(1) = 0, p(1) = 1\}, \{p^0(2) = 1, p(2) = 1\}, \ldots$$

$$\alpha = 1$$

$$w(q) = w(q-1) + a(q)p(q)$$

First Iteration (sight fails):

$$\begin{aligned} a(1) &= hardlim(w^0 p^0(1) + w(0)p(1) - 0.5) \\ &= hardlim(1 \cdot 0 + 0 \cdot 1 - 0.5) = 0 \quad \text{(no response)} \end{aligned}$$

$$w(1) = w(0) + a(1)p(1) = 0 + 0 \cdot 1 = 0$$

Example

Second Iteration (sight works):

$$a(2) = hardlim(w^0 p^0(2) + w(1)p(2) - 0.5)$$
$$= hardlim(1 \cdot 1 + 0 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(2) = w(1) + a(2)p(2) = 0 + 1 \cdot 1 = 1$$

Third Iteration (sight fails):

$$a(3) = hardlim(w^0 p^0(3) + w(2)p(3) - 0.5)$$
$$= hardlim(1 \cdot 0 + 1 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(3) = w(2) + a(3)p(3) = 1 + 1 \cdot 1 = 2$$

Banana will now be detected if either sensor works.

Problems with Hebb Rule

- Weights can become arbitrarily large
- There is no mechanism for weights to decrease

Hebb Rule with Decay

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha\mathbf{a}(q)\mathbf{p}^T(q) - \gamma\mathbf{W}(q-1)$$

$$\boxed{\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha\mathbf{a}(q)\mathbf{p}^T(q)}$$

This keeps the weight matrix from growing without bound, which can be demonstrated by setting both a_i and p_j to 1:

$$w_{ij}^{max} = (1-\gamma)w_{ij}^{max} + \alpha a_i p_j$$

$$w_{ij}^{max} = (1-\gamma)w_{ij}^{max} + \alpha$$

$$w_{ij}^{max} = \frac{\alpha}{\gamma}$$

Example: Banana Associator

$\alpha = 1$ $\gamma = 0.1$

First Iteration (sight fails):

$$a(1) = hardlim(w^0 p^0(1) + w(0)p(1) - 0.5)$$
$$= hardlim(1 \cdot 0 + 0 \cdot 1 - 0.5) = 0 \quad \text{(no response)}$$

$$w(1) = w(0) + a(1)p(1) - 0.1w(0) = 0 + 0 \cdot 1 - 0.1(0) = 0$$

Second Iteration (sight works):

$$a(2) = hardlim(w^0 p^0(2) + w(1)p(2) - 0.5)$$
$$= hardlim(1 \cdot 1 + 0 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

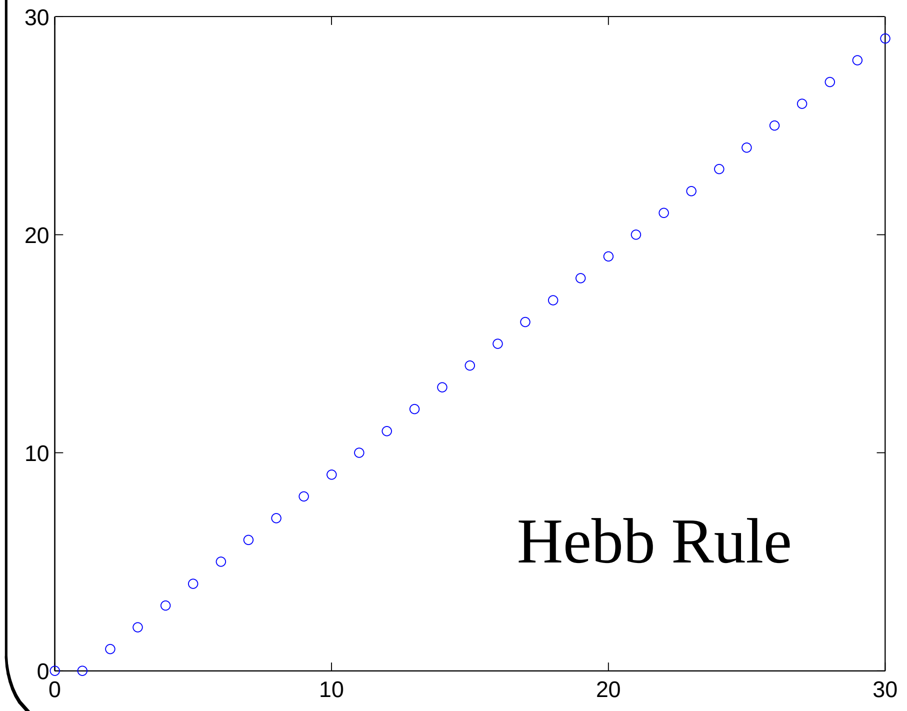
$$w(2) = w(1) + a(2)p(2) - 0.1w(1) = 0 + 1 \cdot 1 - 0.1(0) = 1$$

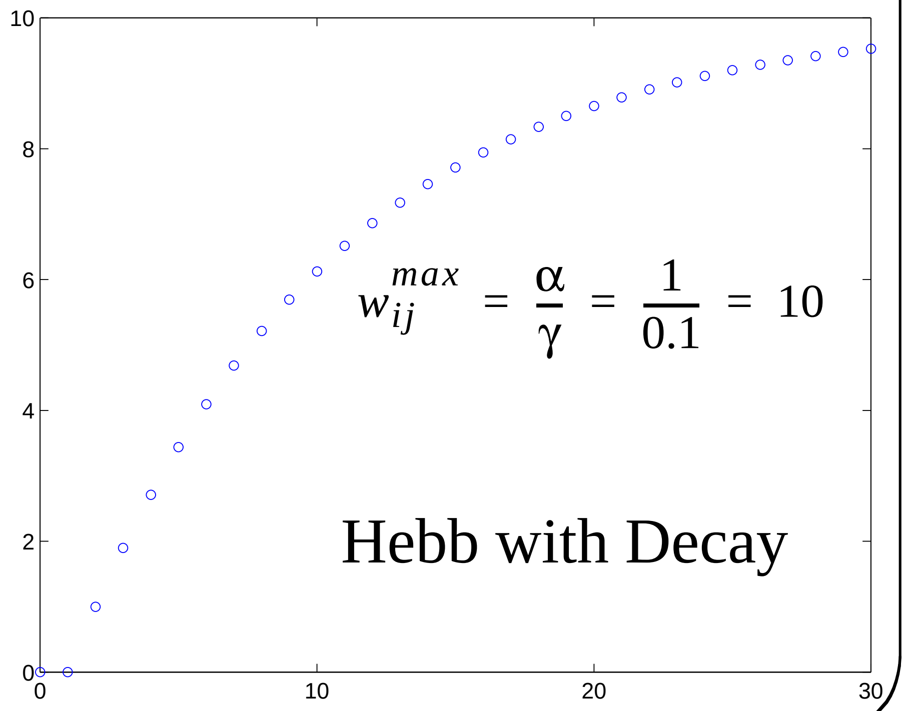
Example

Third Iteration (sight fails):

$$a(3) = hardlim(w^0 p^0(3) + w(2)p(3) - 0.5)$$
$$= hardlim(1 \cdot 0 + 1 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(3) = w(2) + a(3)p(3) - 0.1w(3) = 1 + 1 \cdot 1 - 0.1(1) = 1.9$$





$$w_{ij}^{max} = \frac{\alpha}{\gamma} = \frac{1}{0.1} = 10$$

Problem of Hebb with Decay

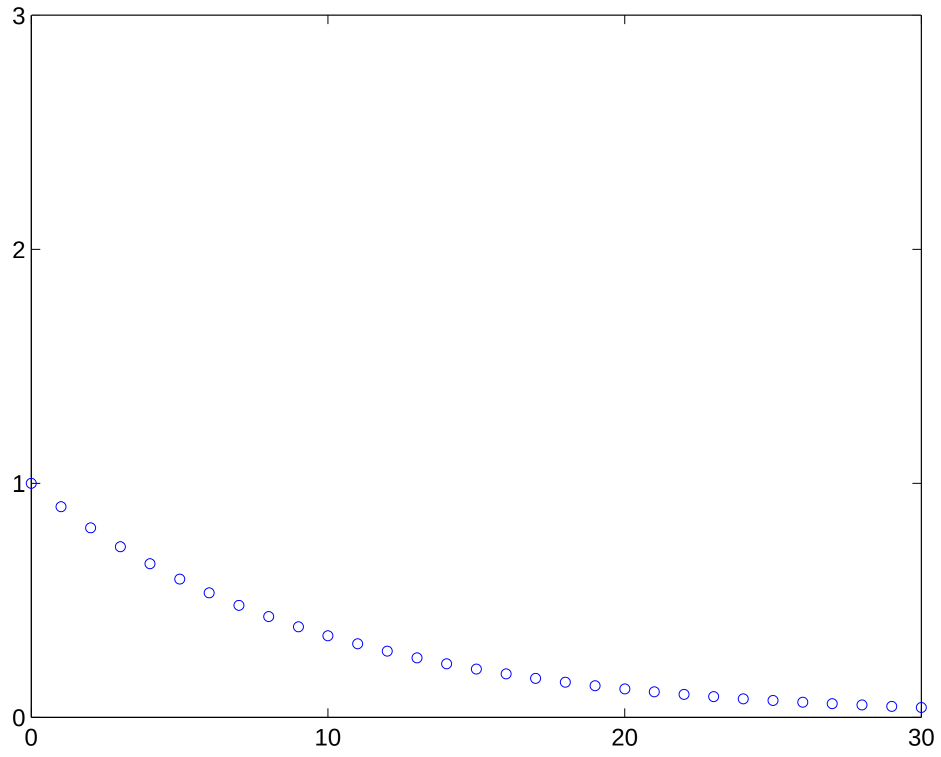
- Associations will decay away if stimuli are not occasionally presented.

If $a_i = 0$, then

$$w_{ij}(q) = (1-\gamma)w_{ij}(q-1)$$

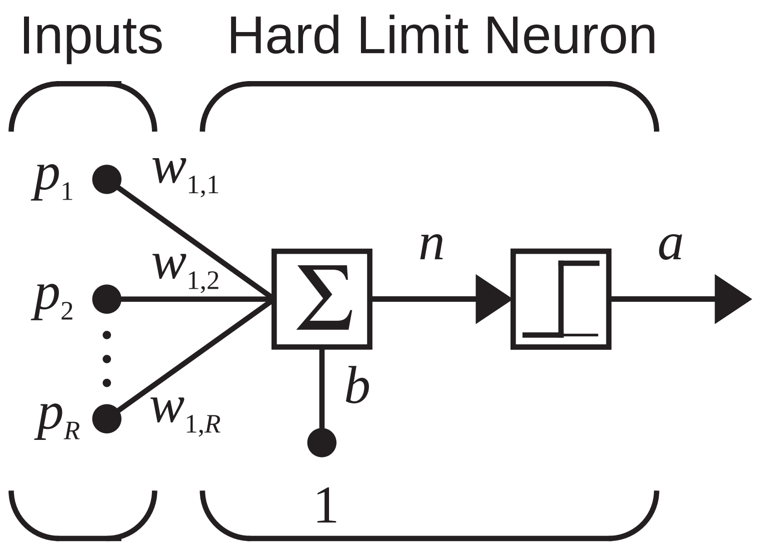
If $\gamma = 0$, this becomes

$$w_{ij}(q) = (0.9)w_{ij}(q-1)$$



Therefore the weight decays by 10% at each iteration where there is no stimulus.

Instar (Recognition Network)



$$a = hardlim(\mathbf{Wp} + b)$$

Instar Operation

$$a = hardlim(\mathbf{W}\mathbf{p} + b) = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$$

The instar will be active when

$${}_1\mathbf{w}^T\mathbf{p} \ge -b$$

or

$${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|\cos\theta \ge -b$$

For normalized vectors, the largest inner product occurs when the angle between the weight vector and the input vector is zero -- the input vector is equal to the weight vector.

The rows of a weight matrix represent patterns to be recognized.

Vector Recognition

If we set

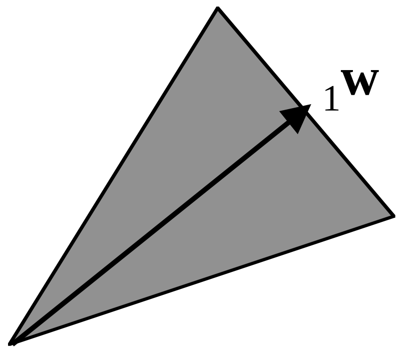
$$b = -\|{}_1\mathbf{w}\|\|\mathbf{p}\|$$

the instar will only be active when $\theta = 0$.

If we set

$$b > -\|{}_1\mathbf{w}\|\|\mathbf{p}\|$$

the instar will be active for a range of angles.



As *b* is increased, the more patterns there will be (over a wider range of θ) which will activate the instar.

Instar Rule

Hebb with Decay

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q)$$

Modify so that learning and forgetting will only occur when the neuron is active - Instar Rule:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma a_i(q) w_{ij}(q-1)$$

or

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)(p_j(q) - w_{ij}(q-1))$$

Vector Form:

$$\boxed{{}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))}$$

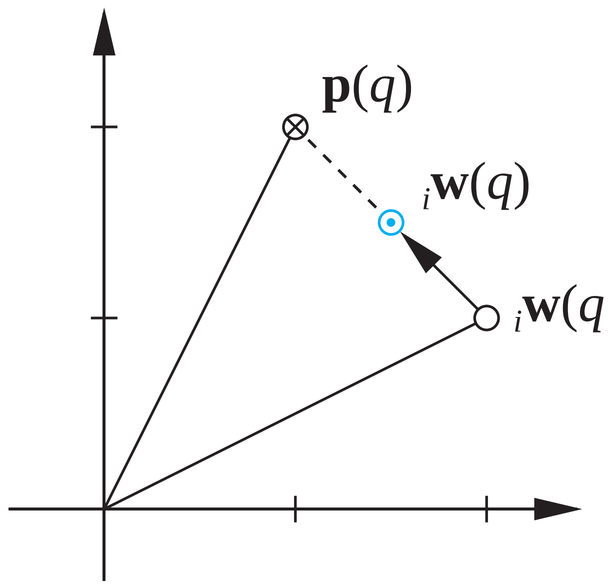
Graphical Representation

For the case where the instar is active ($a_i = 1$):

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) $$

or

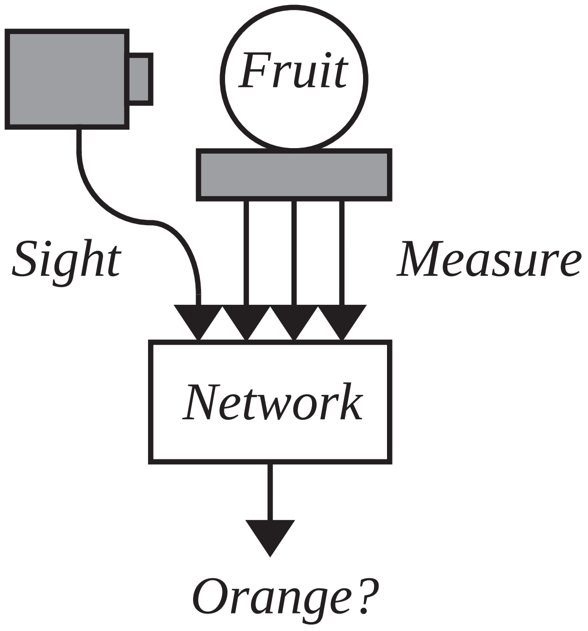
$$ {}_i\mathbf{w}(q) = (1-\alpha)\,{}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) $$



For the case where the instar is inactive ($a_i = 0$):

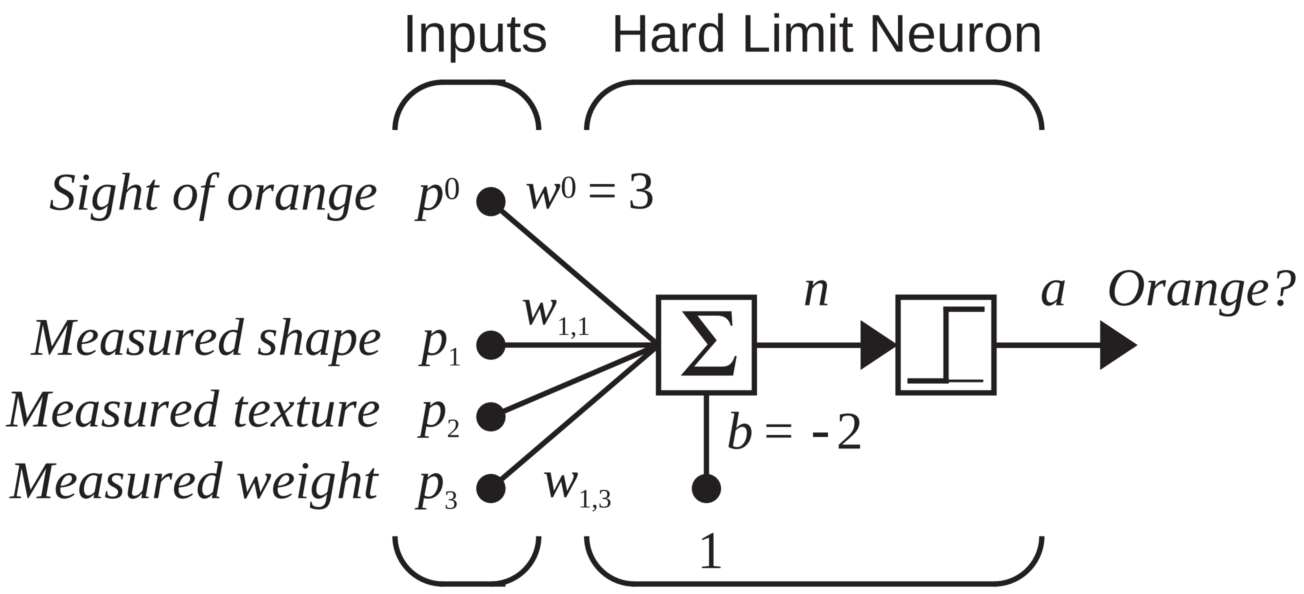
$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) $$

Example



$$p^0 = \begin{cases} 1, & \text{orange detected visually} \\ 0, & \text{orange not detected} \end{cases}$$

$$\mathbf{p} = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix}$$



$$a = hardlim(w^0 p^0 + \mathbf{W}\mathbf{p} + b)$$

13

Training

$$\mathbf{W}(0) = {}_1\mathbf{w}^T(0) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$\left\{ p^0(1) = 0, \mathbf{p}(1) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \left\{ p^0(2) = 1, \mathbf{p}(2) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \right\}, \dots$$

First Iteration (α=1):

$$a(1) = hardlim(w^0 p^0(1) + \mathbf{W}\mathbf{p}(1) - 2)$$

$$a(1) = hardlim\left(3 \cdot 0 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2 \right) = 0 \quad \text{(no response)}$$

$${}_1\mathbf{w}(1) = {}_1\mathbf{w}(0) + a(1)(\mathbf{p}(1) - {}_1\mathbf{w}(0)) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + 0\left(\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Further Training

$$a(2) = hardlim(w^0 p^0(2) + \mathbf{W}\mathbf{p}(2) - 2) = hardlim\left(3 \cdot 1 + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2\right) = 1$$

(orange)

$${}_1\mathbf{w}(2) = {}_1\mathbf{w}(1) + a(2)(\mathbf{p}(2) - {}_1\mathbf{w}(1)) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + 1\left(\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

$$a(3) = hardlim(w^0 p^0(3) + \mathbf{W}\mathbf{p}(3) - 2) = hardlim\left(3 \cdot 0 + \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - 2\right) = 1$$

(orange)

$${}_1\mathbf{w}(3) = {}_1\mathbf{w}(2) + a(3)(\mathbf{p}(3) - {}_1\mathbf{w}(2)) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 1\left(\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

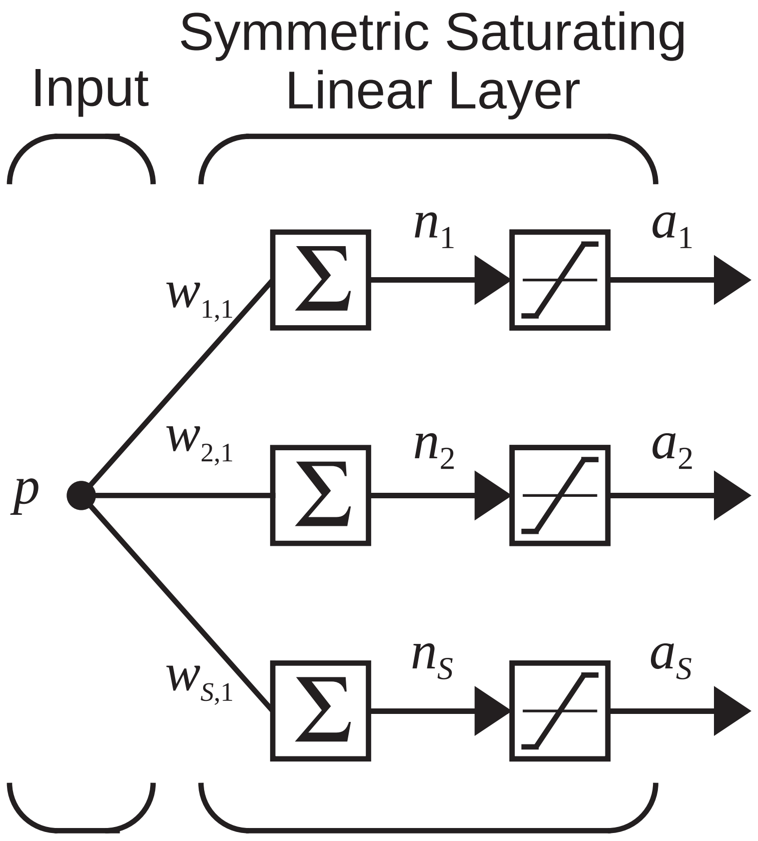
Orange will now be detected if either set of sensors works.

Kohonen Rule

$${}_{1}\mathbf{w}(q) = {}_{1}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{1}\mathbf{w}(q-1)), \quad \text{for } i \in X(q)$$

Learning occurs when the neuron's index *i* is a member of the set *X*(*q*). We will see in Chapter 14 that this can be used to train all neurons in a given neighborhood.

Outstar (Recall Network)



$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}p)$$

Outstar Operation

Suppose we want the outstar to recall a certain pattern **a***
whenever the input $p = 1$ is presented to the network. Let

$$\mathbf{W} = \mathbf{a}^*$$

Then, when $p = 1$

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}p) = \mathbf{satlins}(\mathbf{a}^* \cdot 1) = \mathbf{a}^*$$

and the pattern is correctly recalled.

The columns of a weight matrix represent patterns to be recalled.

Outstar Rule

For the instar rule we made the weight decay term of the Hebb rule proportional to the output of the network. For the outstar rule we make the weight decay term proportional to the input of the network.

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma p_j(q) w_{ij}(q-1)$$

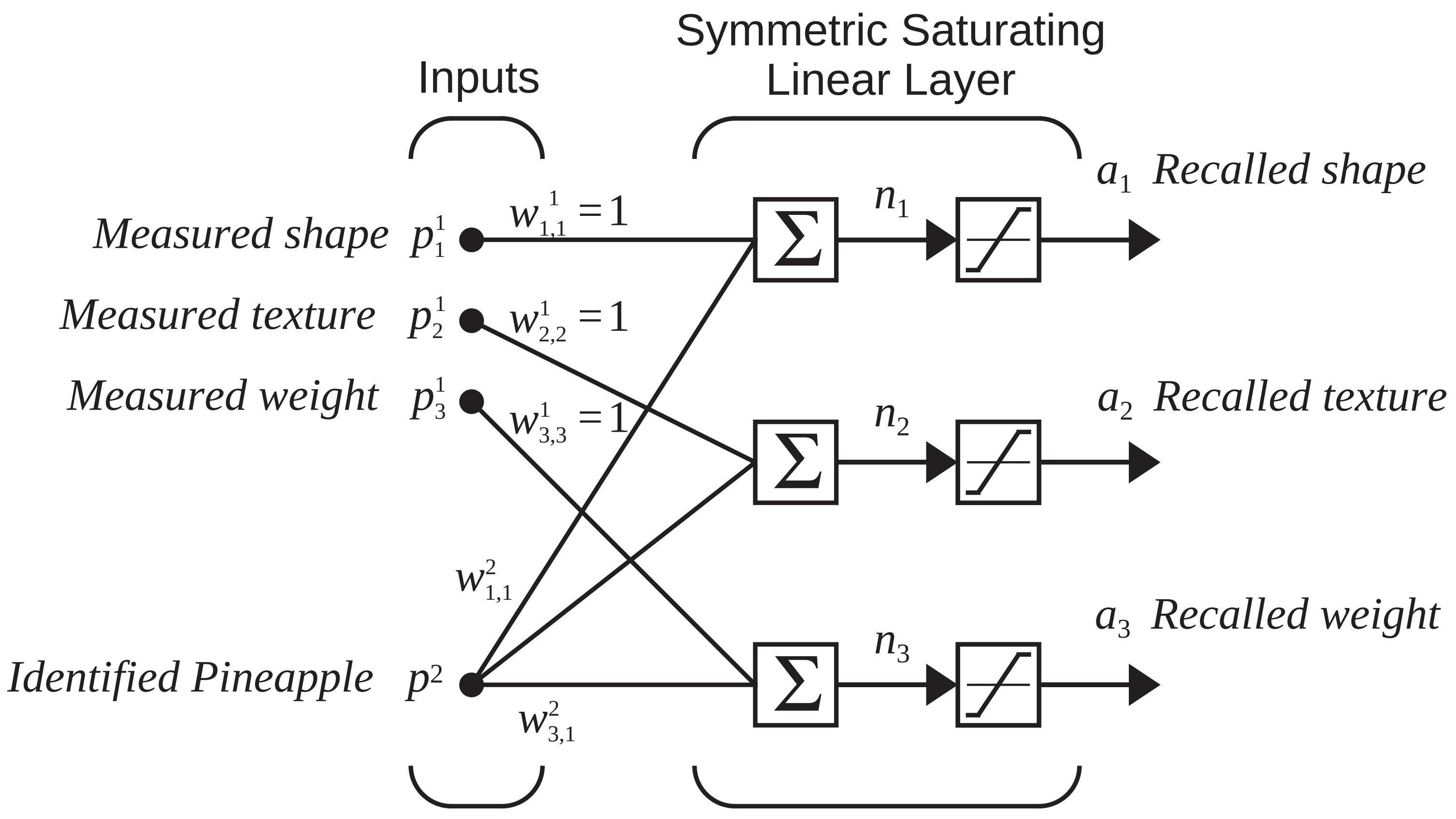
If we make the decay rate γ equal to the learning rate α,

$$w_{ij}(\mathrm{q}) = w_{ij}(q-1) + \alpha(a_i(q) - w_{ij}(q-1)) p_j(q)$$

Vector Form:

$$\boxed{\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha(\mathbf{a}(q) - \mathbf{w}_j(q-1)) p_j(q)}$$

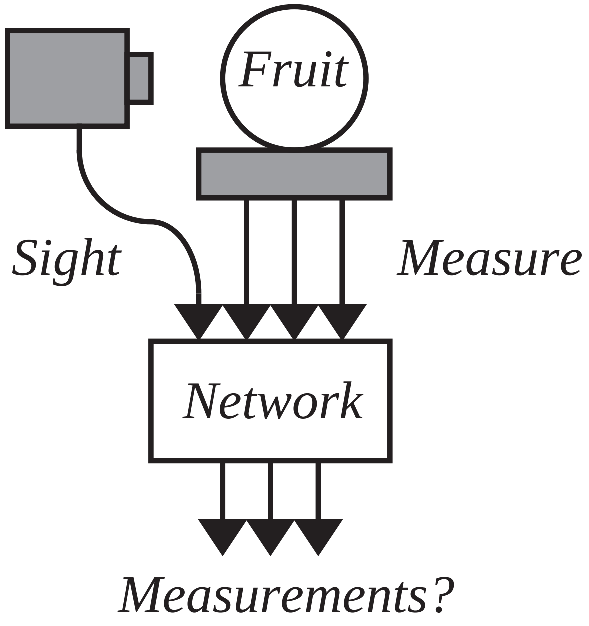
Example - Pineapple Recall



$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}^0\mathbf{p}^0 + \mathbf{W}p)$$

Definitions

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}^0\mathbf{p}^0 + \mathbf{W}p)$$



$$\mathbf{W}^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}^0 = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix} \qquad \mathbf{p}^{pineapple} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

$$p = \begin{cases} 1, & \text{if a pineapple can be seen} \\ 0, & \text{otherwise} \end{cases}$$

Iteration 1

$$\left\{\mathbf{p}^0(1) = \begin{bmatrix}0\\0\\0\end{bmatrix}, p(1) = 1\right\}, \left\{\mathbf{p}^0(2) = \begin{bmatrix}-1\\-1\\1\end{bmatrix}, p(2) = 1\right\}, \ldots$$

$$\alpha = 1$$

$$\mathbf{a}(1) = \mathbf{satlins}\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}1\right) = \begin{bmatrix}0\\0\\0\end{bmatrix} \quad \text{(no response)}$$

$$\mathbf{w}_1(1) = \mathbf{w}_1(0) + (\mathbf{a}(1) - \mathbf{w}_1(0))p(1) = \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

Convergence

$$\mathbf{a}(2) = \mathbf{satlins}\left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad \text{(measurements given)}$$

$$\mathbf{w}_1(2) = \mathbf{w}_1(1) + (\mathbf{a}(2) - \mathbf{w}_1(1))p(2) = \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

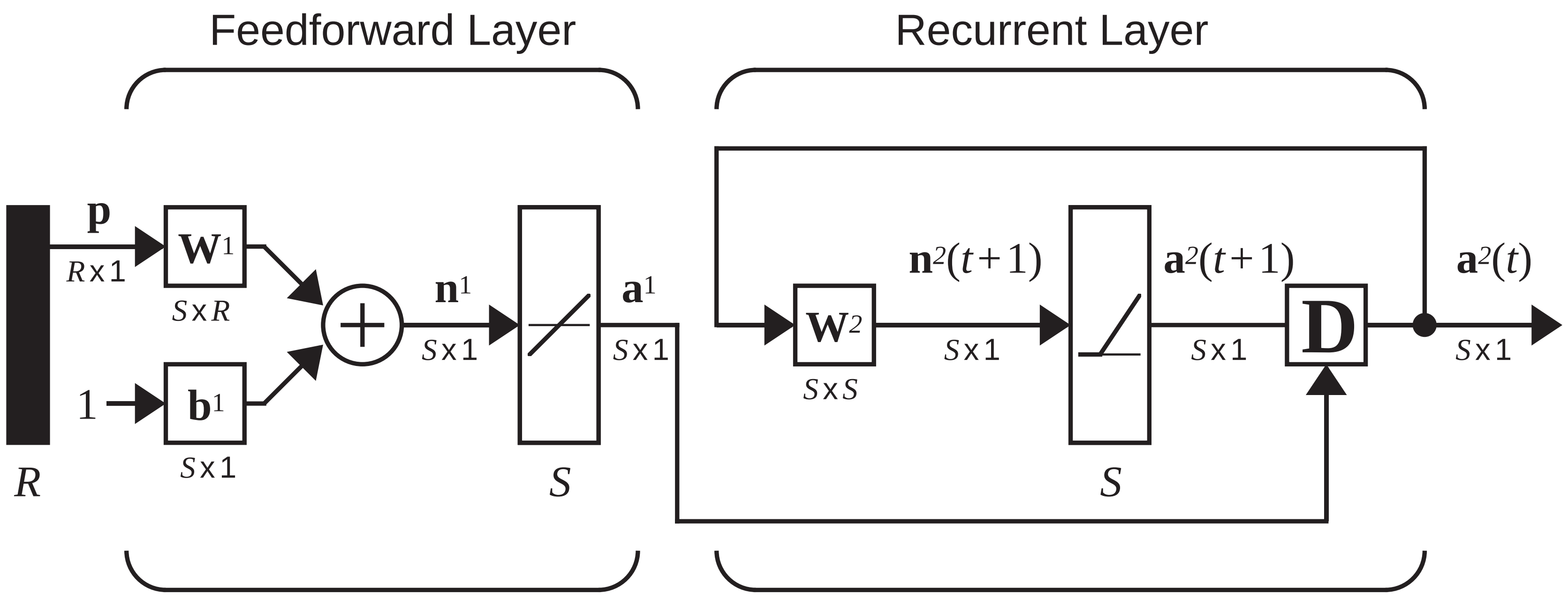
$$\mathbf{a}(3) = \mathbf{satlins}\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}-1\\-1\\1\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad \text{(measurements recalled)}$$

$$\mathbf{w}_1(3) = \mathbf{w}_1(2) + (\mathbf{a}(2) - \mathbf{w}_1(2))p(2) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}-1\\-1\\1\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

14

Competitive Networks

Hamming Network



$$\mathbf{a}^1 = \mathbf{purelin}(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2(0) = \mathbf{a}^1 \qquad \mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

Layer 1 (Correlation)

We want the network to recognize the following prototype vectors:

$$\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_Q\}$$

The first layer weight matrix and bias vector are given by:

$$\mathbf{W}^1 = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} \qquad \mathbf{b}^1 = \begin{bmatrix} R \\ R \\ \vdots \\ R \end{bmatrix}$$

The response of the first layer is:

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + R \\ \mathbf{p}_2^T\mathbf{p} + R \\ \vdots \\ \mathbf{p}_Q^T\mathbf{p} + R \end{bmatrix}$$

The prototype closest to the input vector produces the largest response.

Layer 2 (Competition)

$$\mathbf{a}^2(0) = \mathbf{a}^1$$

The second layer is initialized with the output of the first layer.

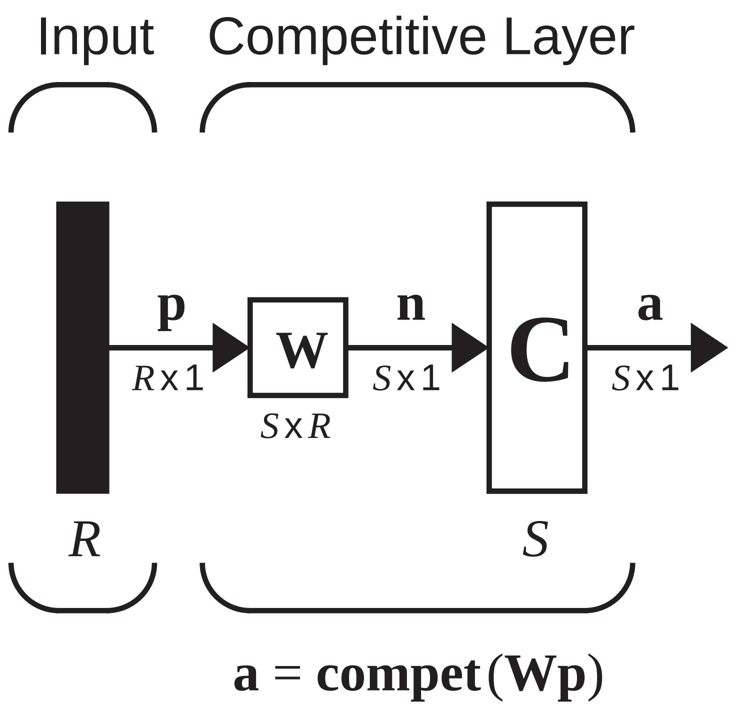
$$\mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

$$w_{ij}^2 = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{otherwise} \end{cases} \qquad 0 < \varepsilon < \frac{1}{S-1}$$

$$a_i^2(t+1) = poslin\left(a_i^2(t) - \varepsilon \sum_{j \neq i} a_j^2(t)\right)$$

The neuron with the largest initial condition will win the competiton.

Competitive Layer



$$\mathbf{n} = \mathbf{W}\mathbf{p} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}\mathbf{p} = \begin{bmatrix} {}_1\mathbf{w}^T\mathbf{p} \\ {}_2\mathbf{w}^T\mathbf{p} \\ \vdots \\ {}_S\mathbf{w}^T\mathbf{p} \end{bmatrix} = \begin{bmatrix} L^2\cos\theta_1 \\ L^2\cos\theta_2 \\ \vdots \\ L^2\cos\theta_S \end{bmatrix}$$

$$\mathbf{a} = \mathbf{compet}(\mathbf{n})$$

$$a_i = \begin{cases} 1, & i = i^* \\ 0, & i \neq i^* \end{cases} \qquad n_{i^*} \geq n_i, \forall i \qquad i^* \leq i, \forall n_i = n_{i^*}$$

Competitive Learning

Instar Rule

$${}_{i}\mathbf{w}(q) = {}_{i}\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_{i}\mathbf{w}(q-1))$$

For the competitive network, the winning neuron has an ouput of 1, and the other neurons have an output of 0.

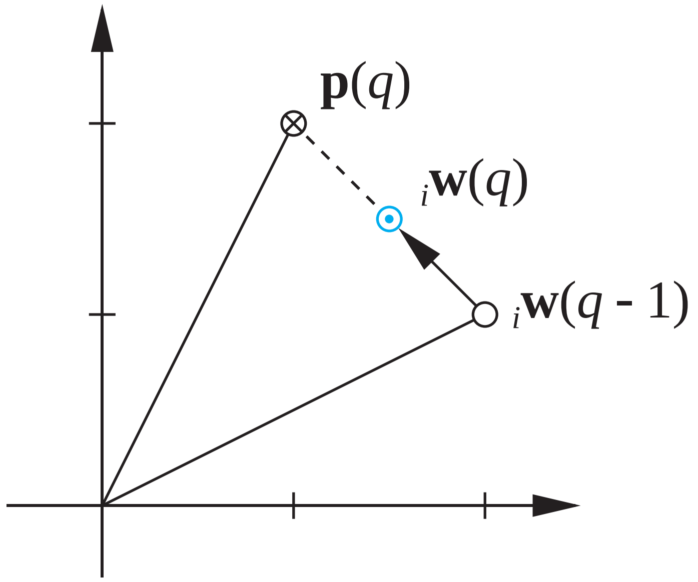
Kohonen Rule

$${}_{i*}\mathbf{w}(q) = {}_{i*}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{i*}\mathbf{w}(q-1))$$

$${}_{i*}\mathbf{w}(q) = (1-\alpha)\,{}_{i*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q)$$

$${}_{i}\mathbf{w}(q) = {}_{i}\mathbf{w}(q-1) \qquad i \neq i*$$

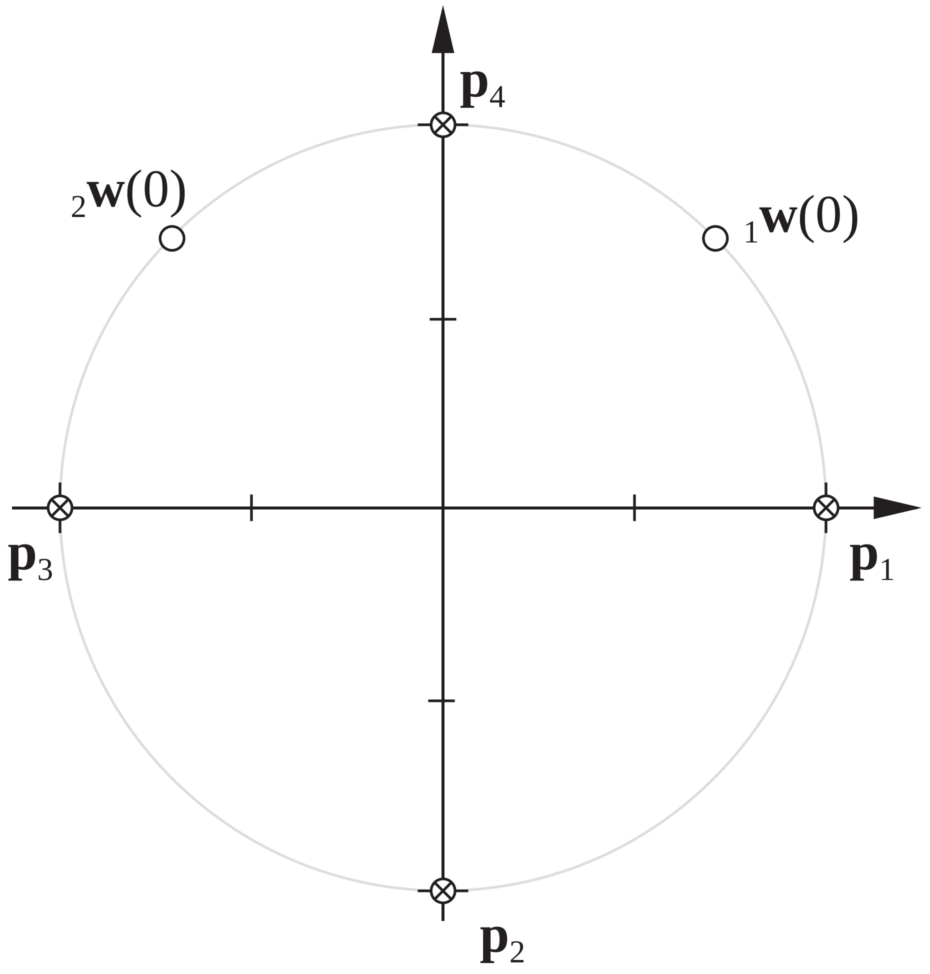
Graphical Representation



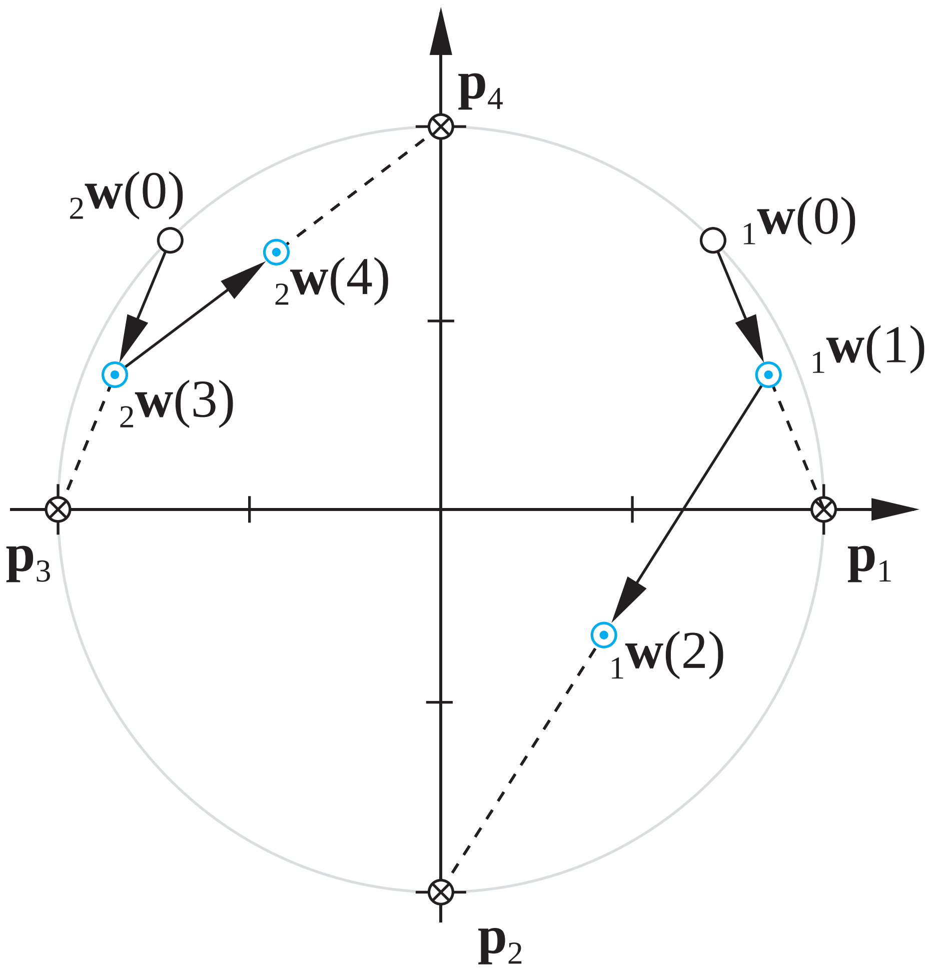
$$ {}_{i*}\mathbf{w}(q) = {}_{i*}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{i*}\mathbf{w}(q-1)) $$

$$ {}_{i*}\mathbf{w}(q) = (1-\alpha)\,{}_{i*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q) $$

Example


$\mathbf{p}_4$
${}_2\mathbf{w}(0)$
${}_1\mathbf{w}(0)$
$\mathbf{p}_3$
$\mathbf{p}_1$
$\mathbf{p}_2$

Four Iterations

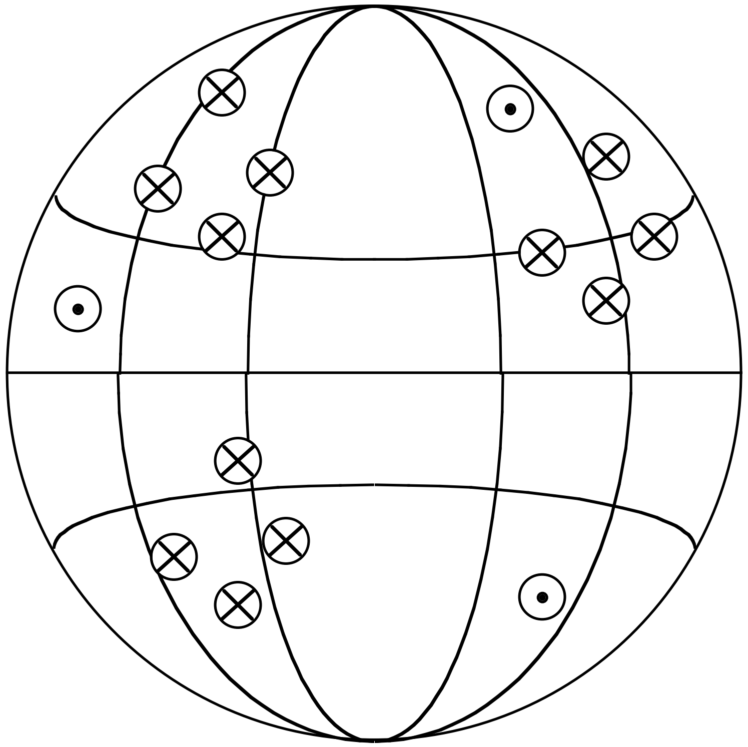


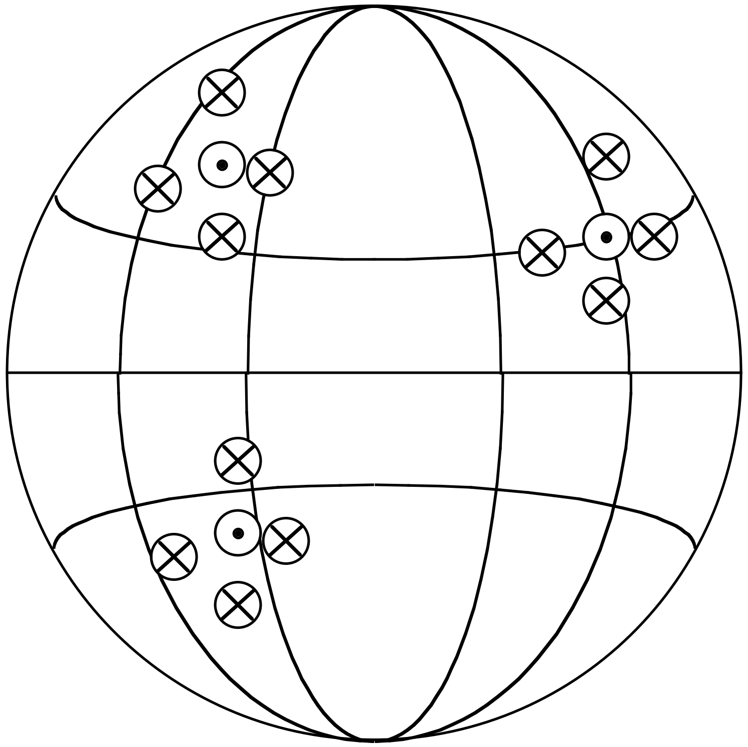
14

Typical Convergence (Clustering)

⊙ Weights

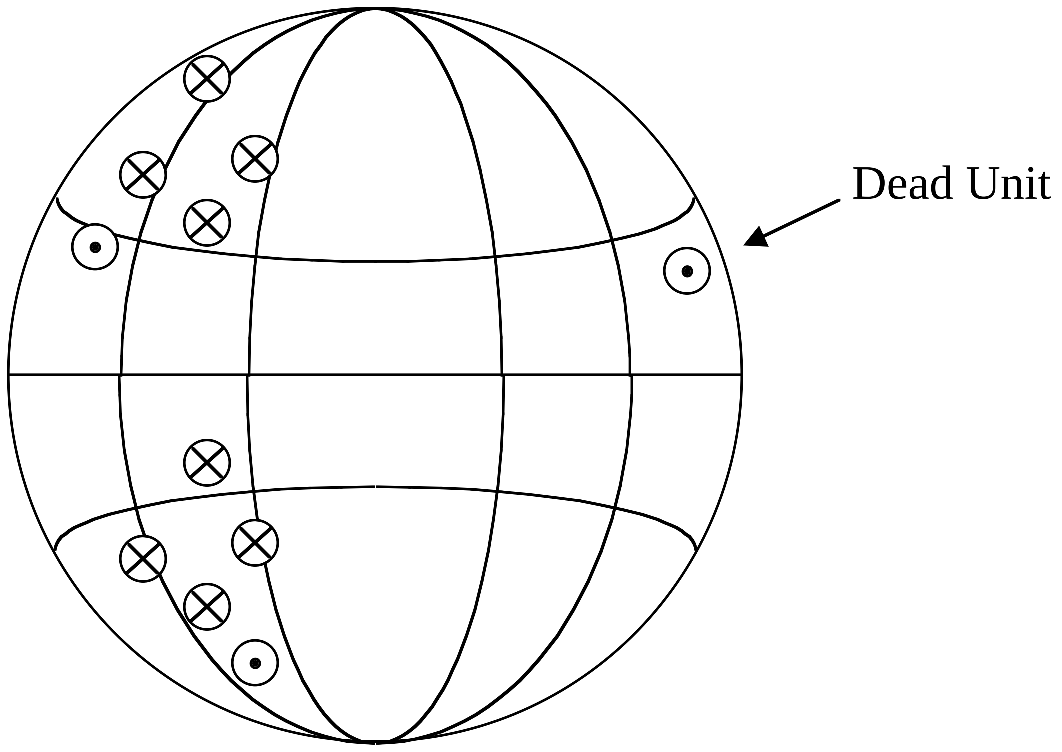
⊗ Input Vectors


Before Training


After Training

Dead Units

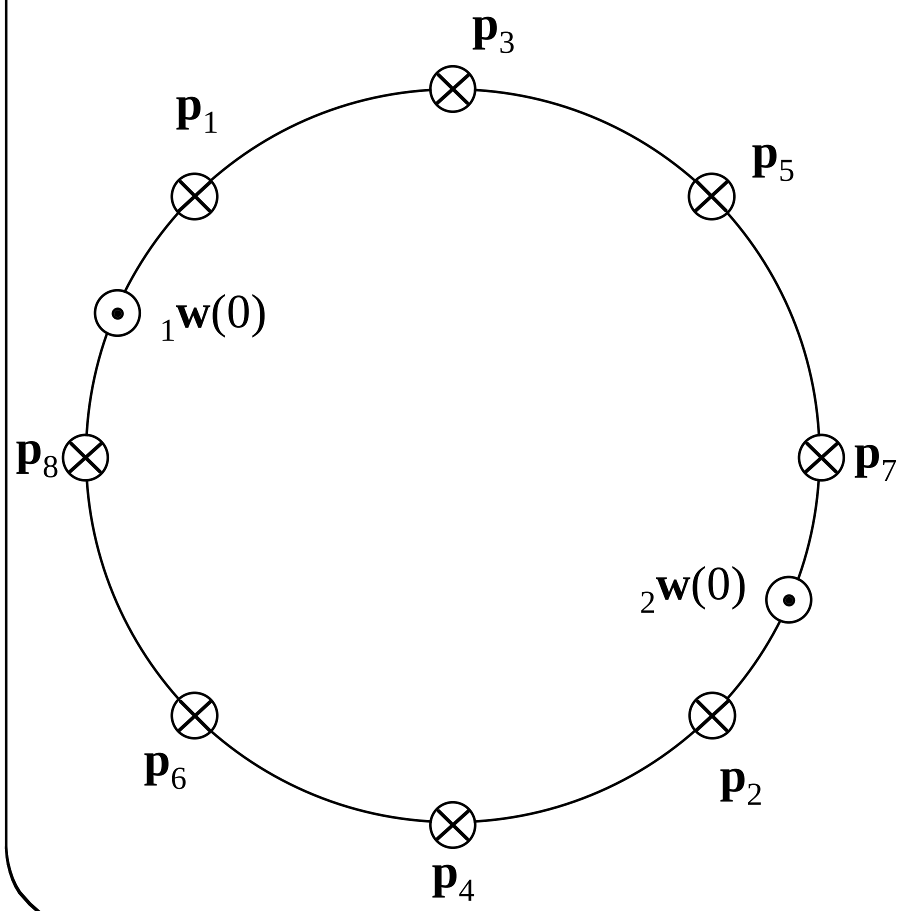
One problem with competitive learning is that neurons with initial weights far from any input vector may never win.

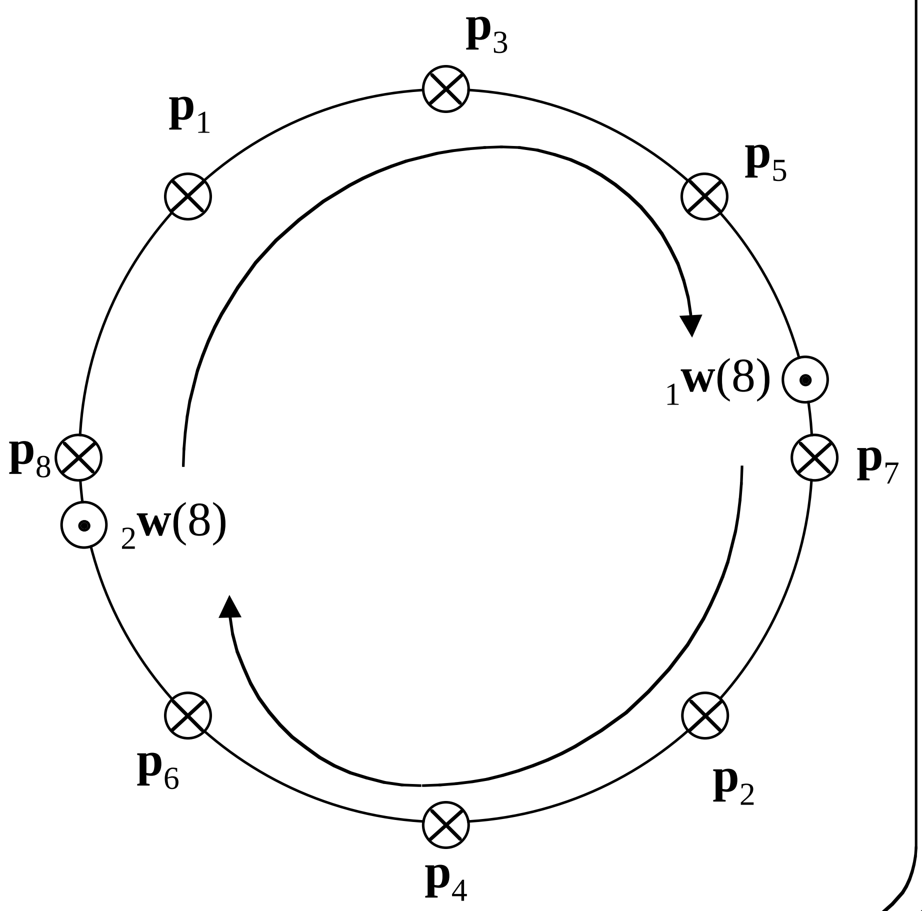


Solution: Add a negative bias to each neuron, and increase the magnitude of the bias as the neuron wins. This will make it harder to win if a neuron has won often. This is called a “conscience.”

Stability

If the input vectors don’t fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution.





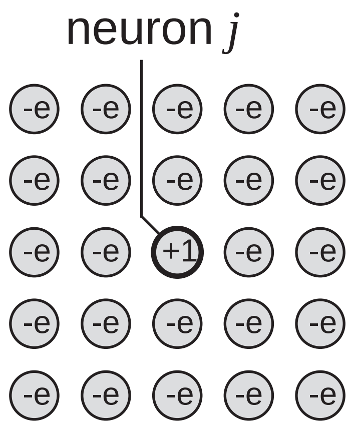
Competitive Layers in Biology

On-Center/Off-Surround Connections for Competition

Weights in the competitive layer of the Hamming network:

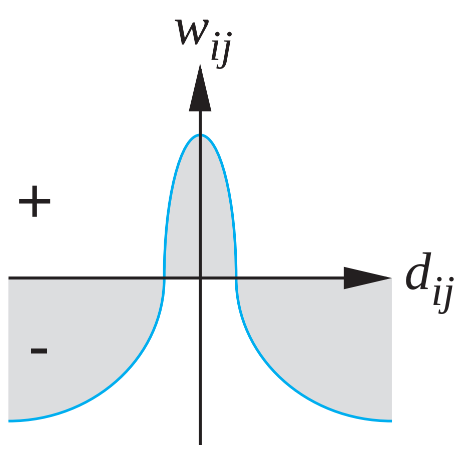
$$w_{i,j} = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{if } i \neq j \end{cases}$$

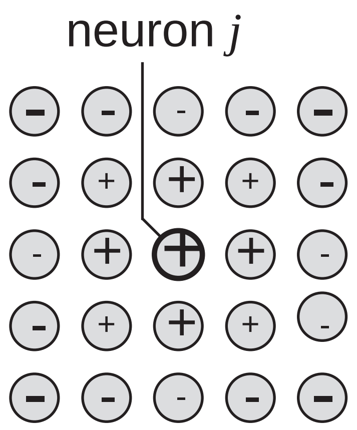
Weights assigned based on distance:



$$w_{i,j} = \begin{cases} 1, & \text{if } d_{i,j} = 0 \\ -\varepsilon, & \text{if } d_{i,j} > 0 \end{cases}$$

Mexican-Hat Function





Feature Maps

Update weight vectors in a neighborhood of the winning neuron.

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)) $$

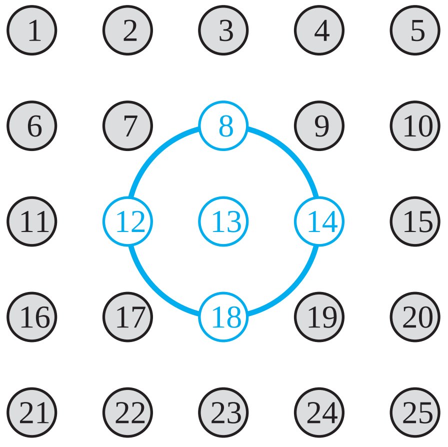
$$ i \in N_{i*}(d) $$

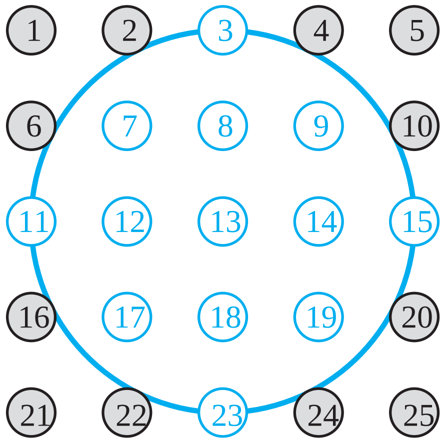
$$ {}_i\mathbf{w}(q) = (1-\alpha)\,{}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) $$

$$ N_i(d) = \{j, \mathrm{d}_{i,j} \le d\} $$

$$ N_{13}(1) = \{8, 12, 13, 14, 18\} $$

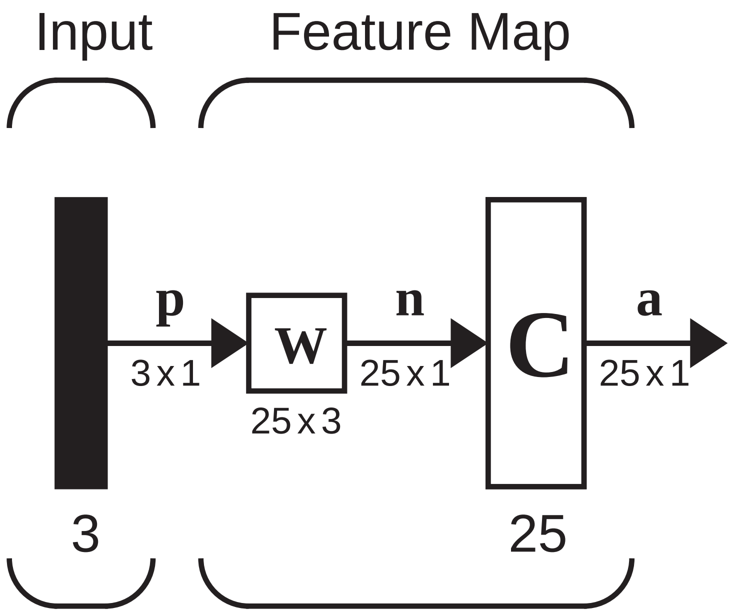
$$ N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\} $$

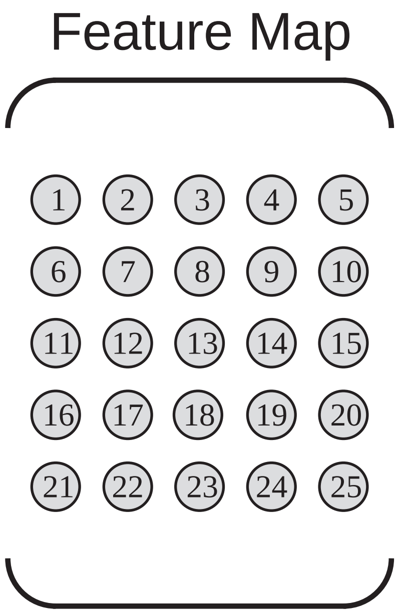




$N_{13}(1)$ $N_{13}(2)$

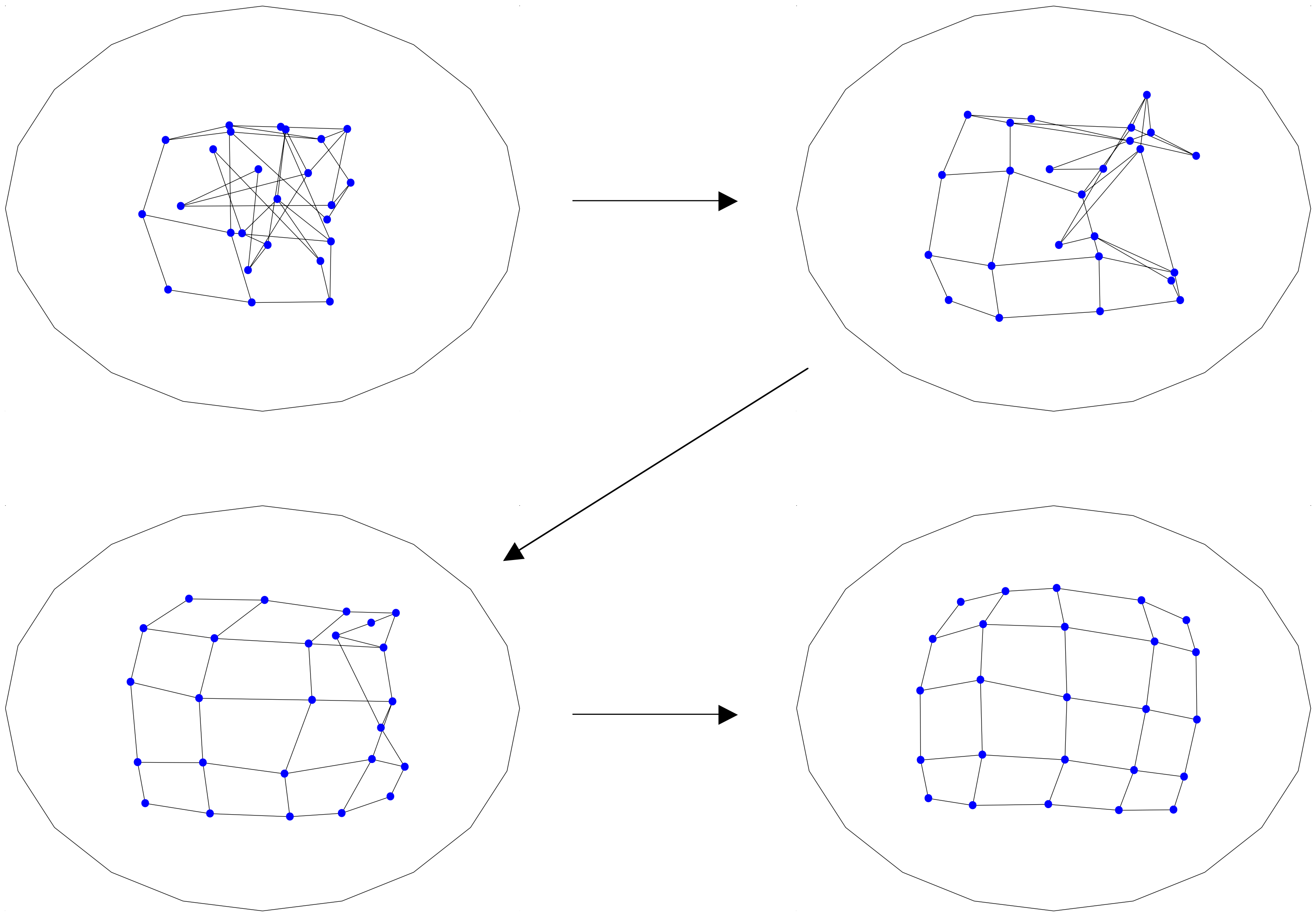
Example



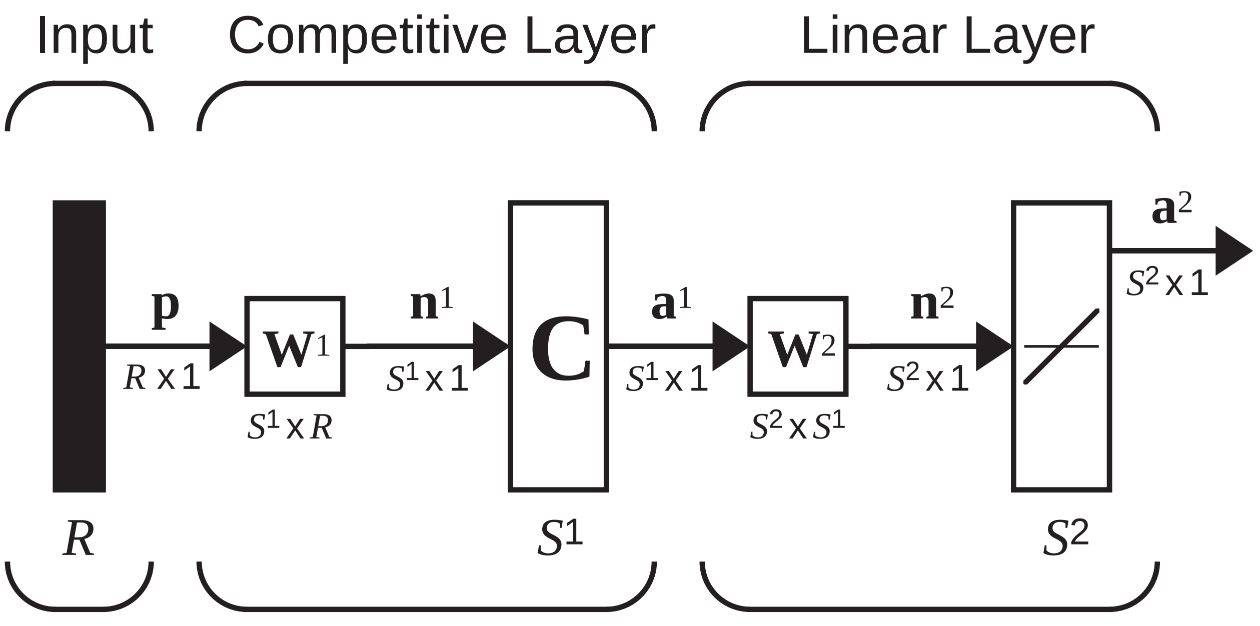


$$\mathbf{a} = \mathbf{compet}(\mathbf{W}\mathbf{p})$$

Convergence



Learning Vector Quantization



$$n_i^1 = -\| {}_i\mathbf{w}^1 - \mathbf{p} \|$$

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1)$$

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$$

The net input is not computed by taking an inner product of the prototype vectors with the input. Instead, the net input is the negative of the distance between the prototype vectors and the input.

Subclass

For the LVQ network, the winning neuron in the first layer indicates the **subclass** which the input vector belongs to. There may be several different neurons (subclasses) which make up each class.

The second layer of the LVQ network combines subclasses into a single class. The columns of $\mathbf{W}^2$ represent subclasses, and the rows represent classes. $\mathbf{W}^2$ has a single 1 in each column, with the other elements set to zero. The row in which the 1 occurs indicates which class the appropriate subclass belongs to.

$$(w^2_{k,i} = 1) \Rightarrow \text{subclass } i \text{ is a part of class } k$$

Example

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

- Subclasses 1, 3 and 4 belong to class 1.
- Subclass 2 belongs to class 2.
- Subclasses 5 and 6 belong to class 3.

A single-layer competitive network can create convex classification regions. The second layer of the LVQ network can combine the convex regions to create more complex categories.

LVQ Learning

LVQ learning combines competive learning with supervision.
It requires a training set of examples of proper network behavior.

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots , \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

If the input pattern is classified correctly, then move the winning weight toward the input vector according to the Kohonen rule.

$$_{i*}\mathbf{w}^1(q) = {}_{i*}\mathbf{w}^1(q-1) + \alpha(\mathbf{p}(q) - {}_{i*}\mathbf{w}^1(q-1)) \qquad a^2_{k*} = t_{k*} = 1$$

If the input pattern is classified incorrectly, then move the winning weight away from the input vector.

$$_{i*}\mathbf{w}^1(q) = {}_{i*}\mathbf{w}^1(q-1) - \alpha(\mathbf{p}(q) - {}_{i*}\mathbf{w}^1(q-1)) \qquad a^2_{k*} = 1 \neq t_{k*} = 0$$

Example

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \left\{\mathbf{p}_2 = \begin{bmatrix}1\\0\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \left\{\mathbf{p}_3 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}\right\} \left\{\mathbf{p}_4 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\}$$

$$\mathbf{W}^1(0) = \begin{bmatrix}({}_1\mathbf{w}^1)^T\\({}_2\mathbf{w}^1)^T\\({}_3\mathbf{w}^1)^T\\({}_4\mathbf{w}^1)^T\end{bmatrix} = \begin{bmatrix}0.25 & 0.75\\0.75 & 0.75\\1 & 0.25\\0.5 & 0.25\end{bmatrix} \qquad \mathbf{W}^2 = \begin{bmatrix}1 & 1 & 0 & 0\\0 & 0 & 1 & 1\end{bmatrix}$$

First Iteration

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1) = \mathbf{compet}\left(\begin{bmatrix} -\|{}_1\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|{}_2\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|{}_3\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|{}_4\mathbf{w}^1 - \mathbf{p}_1\| \end{bmatrix}\right)$$

$$\mathbf{a}^1 = \mathbf{compet}\left(\begin{bmatrix} -\left\|\begin{bmatrix}0.25 & 0.75\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \\ -\left\|\begin{bmatrix}0.75 & 0.75\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \\ -\left\|\begin{bmatrix}1.00 & 0.25\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \\ -\left\|\begin{bmatrix}0.50 & 0.25\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix} -0.354 \\ -0.791 \\ -1.25 \\ -0.901 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Second Layer

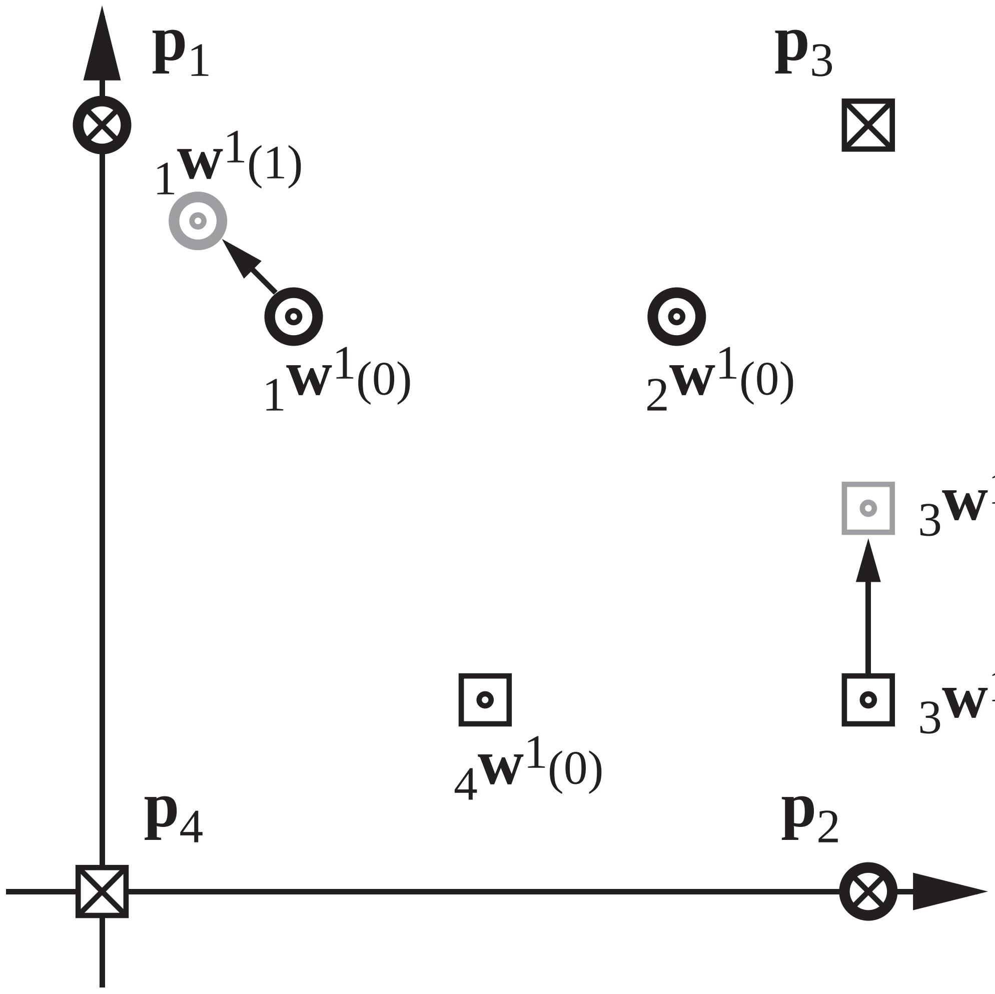
$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This is the correct class, therefore the weight vector is moved toward the input vector.

$${}_1\mathbf{w}^1(1) = {}_1\mathbf{w}^1(0) + \alpha(\mathbf{p}_1 - {}_1\mathbf{w}^1(0))$$

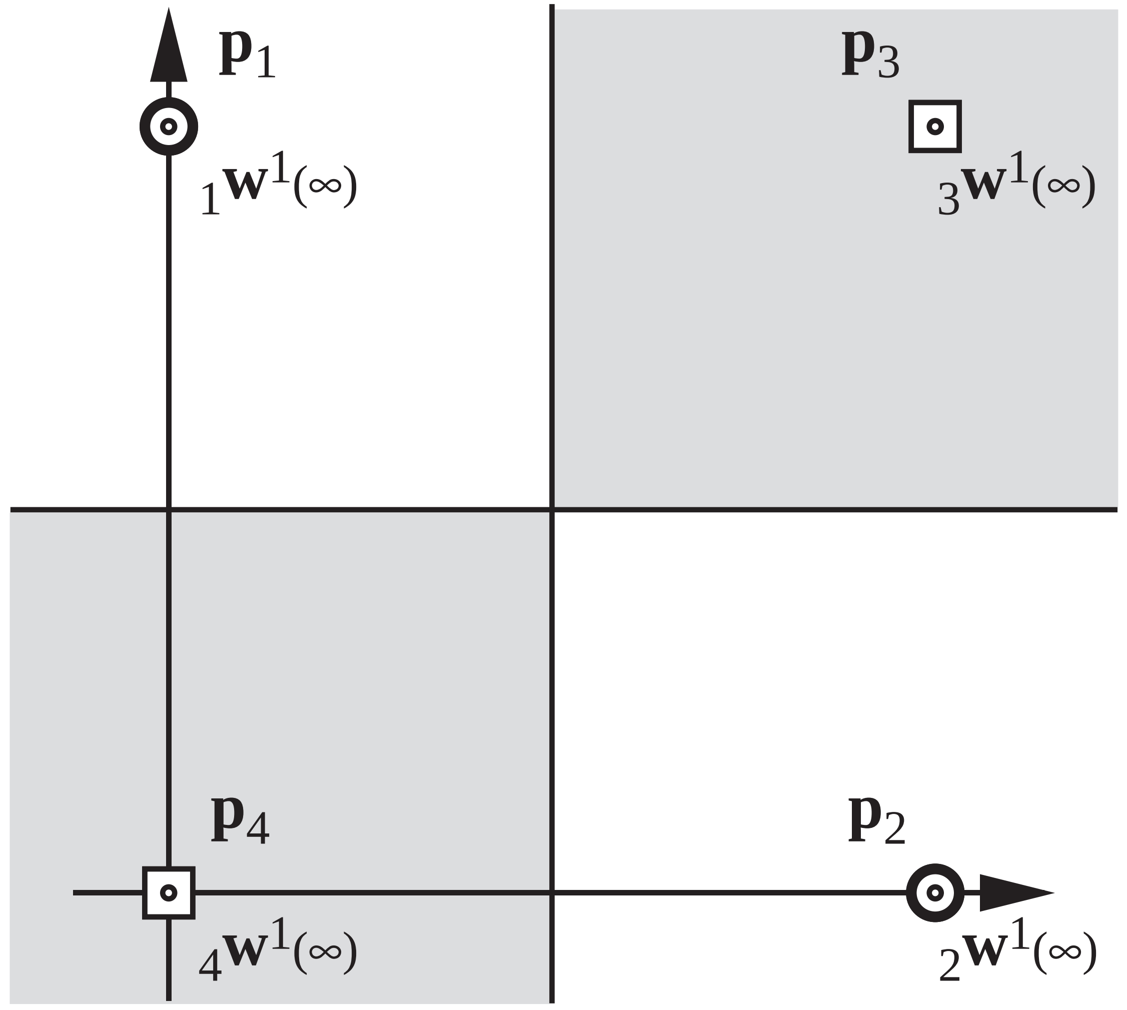
$${}_1\mathbf{w}^1(1) = \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix} + 0.5\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix}\right) = \begin{bmatrix} 0.125 \\ 0.875 \end{bmatrix}$$

Figure



$\mathbf{p}_1$
$\mathbf{p}_3$
${}_1\mathbf{w}^1(1)$
${}_1\mathbf{w}^1(0)$
${}_2\mathbf{w}^1(0)$
${}_3\mathbf{w}^1(2)$
${}_3\mathbf{w}^1(0)$
${}_4\mathbf{w}^1(0)$
$\mathbf{p}_4$
$\mathbf{p}_2$

Final Decision Regions

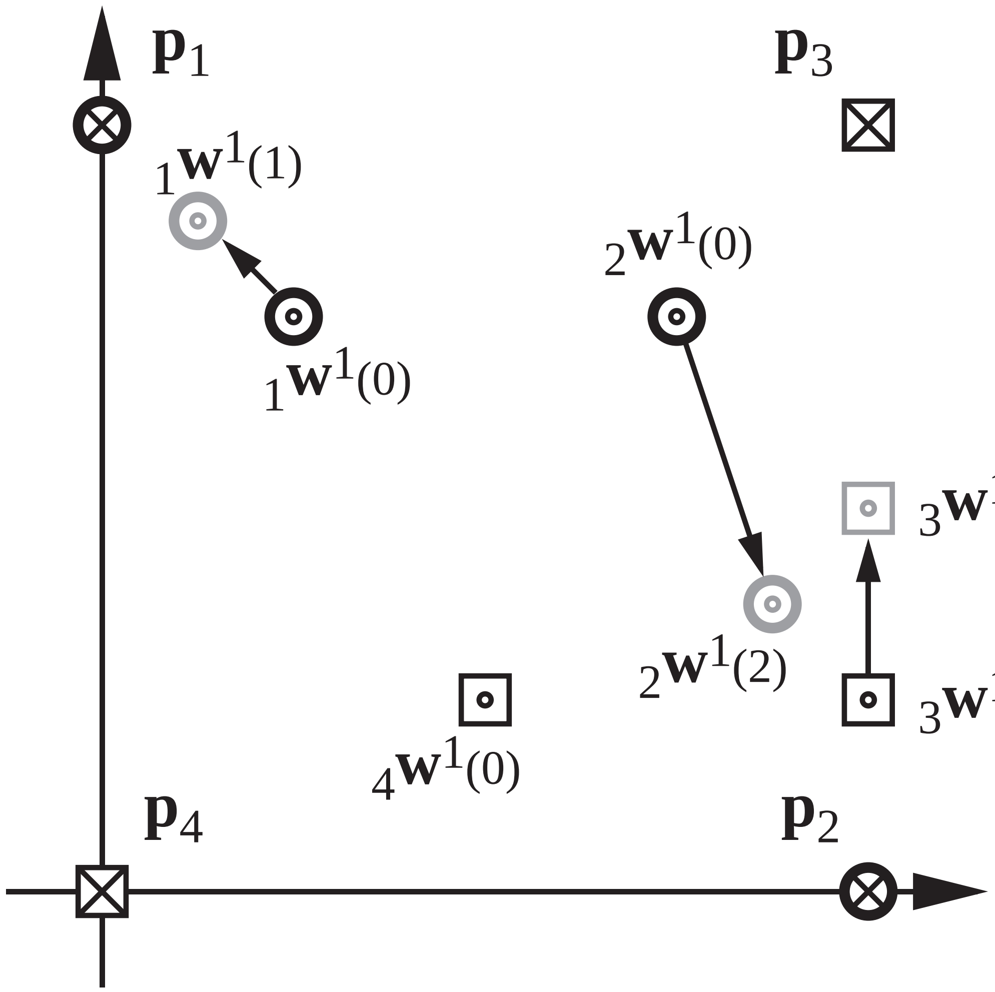

$\mathbf{p}_1$
${}_1\mathbf{w}^1(\infty)$
$\mathbf{p}_3$
${}_3\mathbf{w}^1(\infty)$
$\mathbf{p}_4$
${}_4\mathbf{w}^1(\infty)$
$\mathbf{p}_2$
${}_2\mathbf{w}^1(\infty)$

LVQ2

If the winning neuron in the hidden layer incorrectly classifies the current input, we move its weight vector away from the input vector, as before. However, we also adjust the weights of the closest neuron to the input vector that does classify it properly. The weights for this second neuron should be moved toward the input vector.

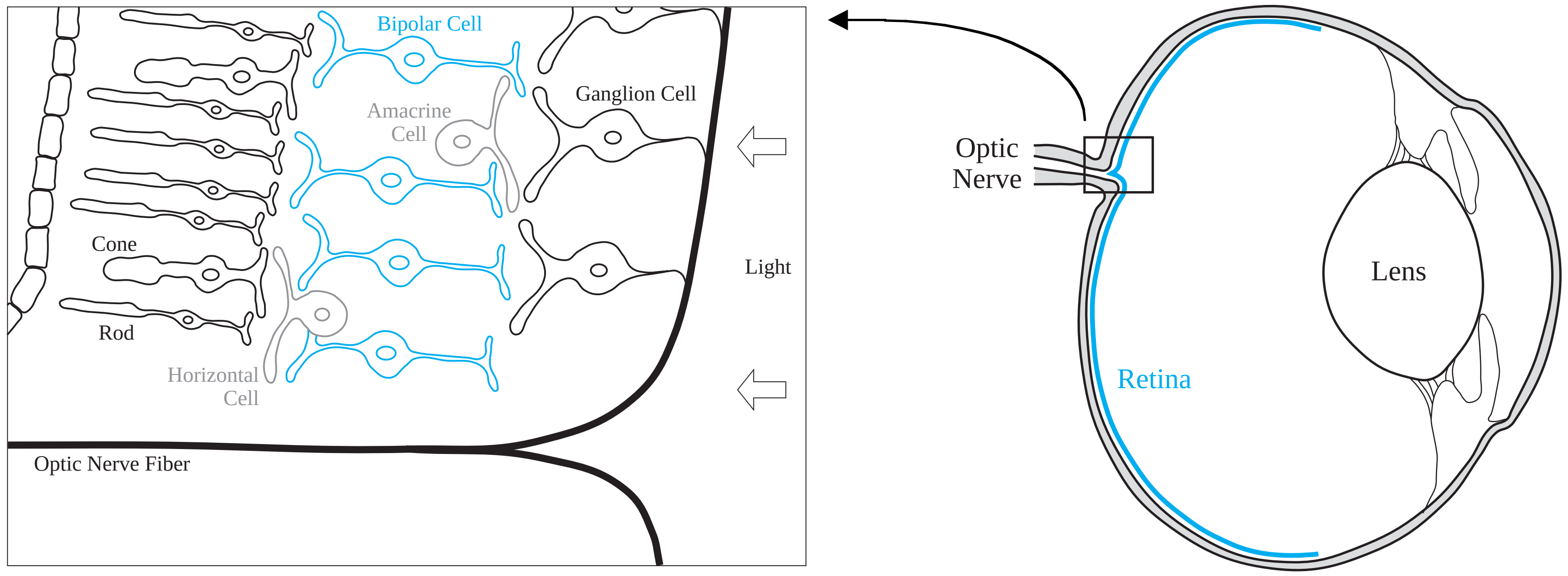
When the network correctly classifies an input vector, the weights of only one neuron are moved toward the input vector. However, if the input vector is incorrectly classified, the weights of two neurons are updated, one weight vector is moved away from the input vector, and the other one is moved toward the input vector. The resulting algorithm is called **LVQ2**.

LVQ2 Example



Grossberg Network

Biological Motivation: Vision



Eyeball and Retina

Layers of Retina

The retina is a part of the brain that covers the back inner wall of the eye and consists of three layers of neurons:

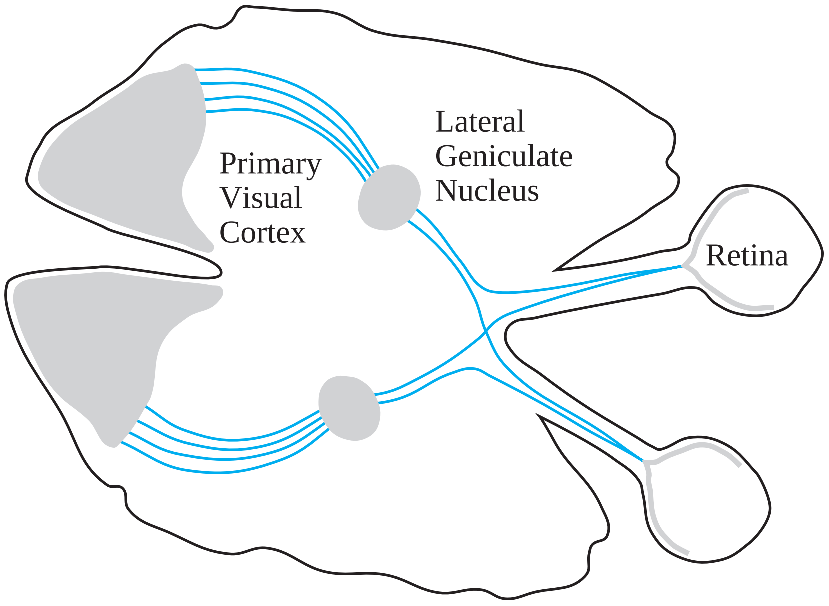
Outer Layer:
- Photoreceptors - convert light into electrical signals
  - Rods - allow us to see in dim light
  - Cones - fine detail and color

Middle Layer
- Bipolar Cells - link photoreceptors to third layer
- Horizontal Cells - link receptors with bipolar cells
- Amacrine Cells - link bipolar cells with ganglion cells

Final Layer
- Ganglion Cells - link retina to brain through optic nerve

15

Visual Pathway



Photograph of the Retina

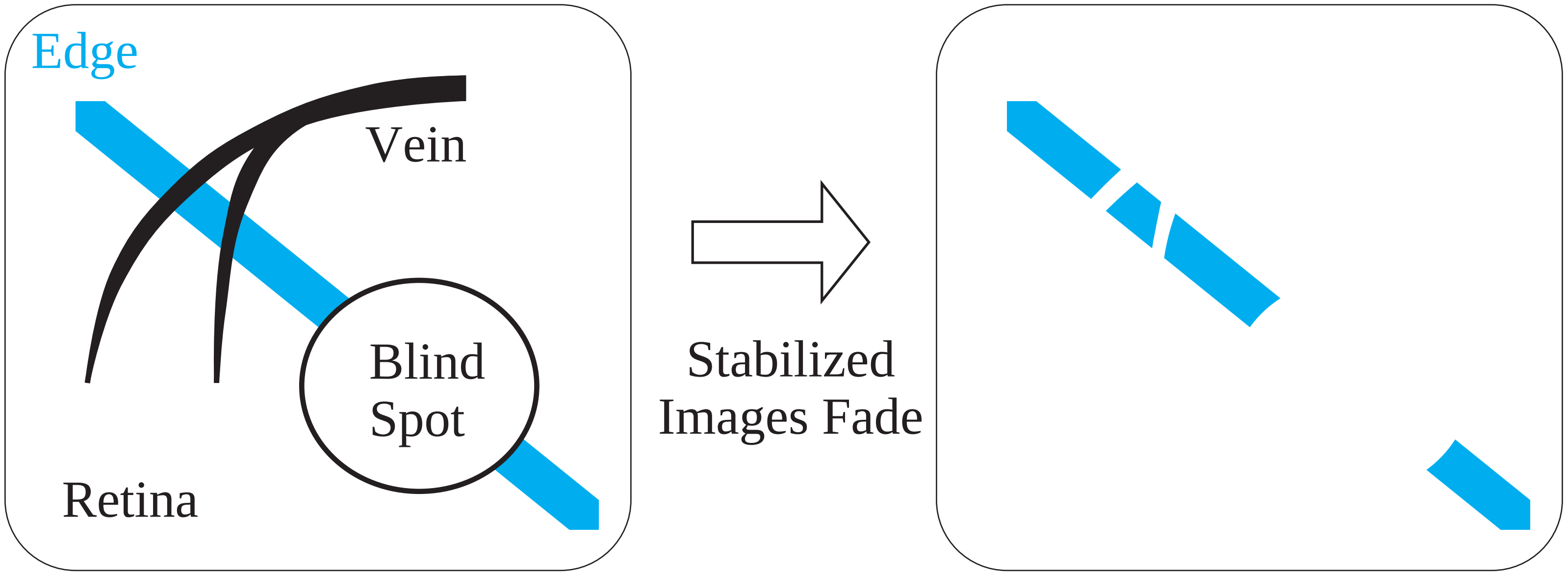
Blind Spot (Optic Disk)

Vein

Fovea

15

Imperfections in Retinal Uptake



15

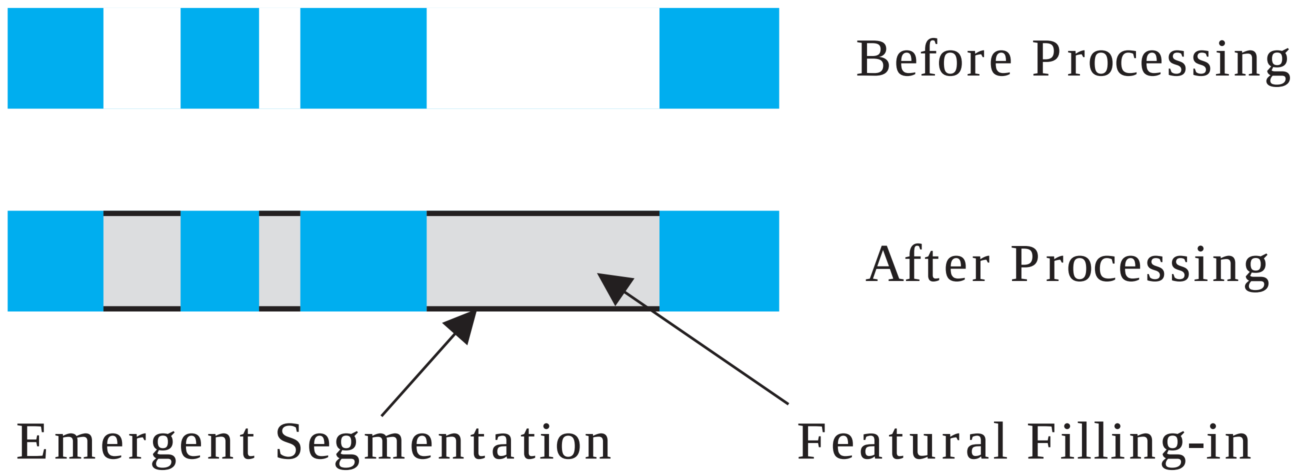
Compensatory Processing

Emergent Segmentation:

Complete missing boundaries.

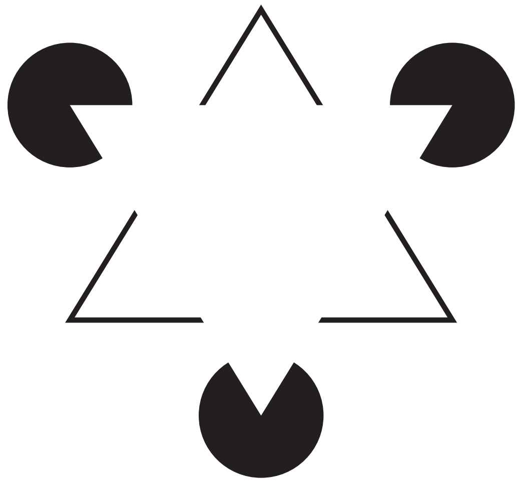
Featural Filling-In:

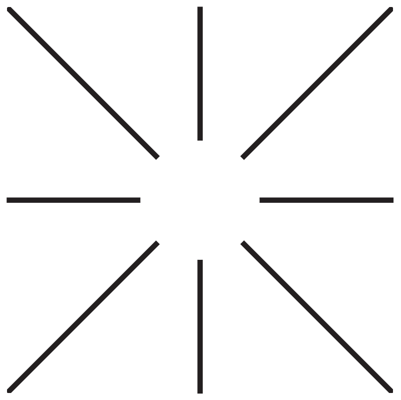
Fill in color and brightness.



15

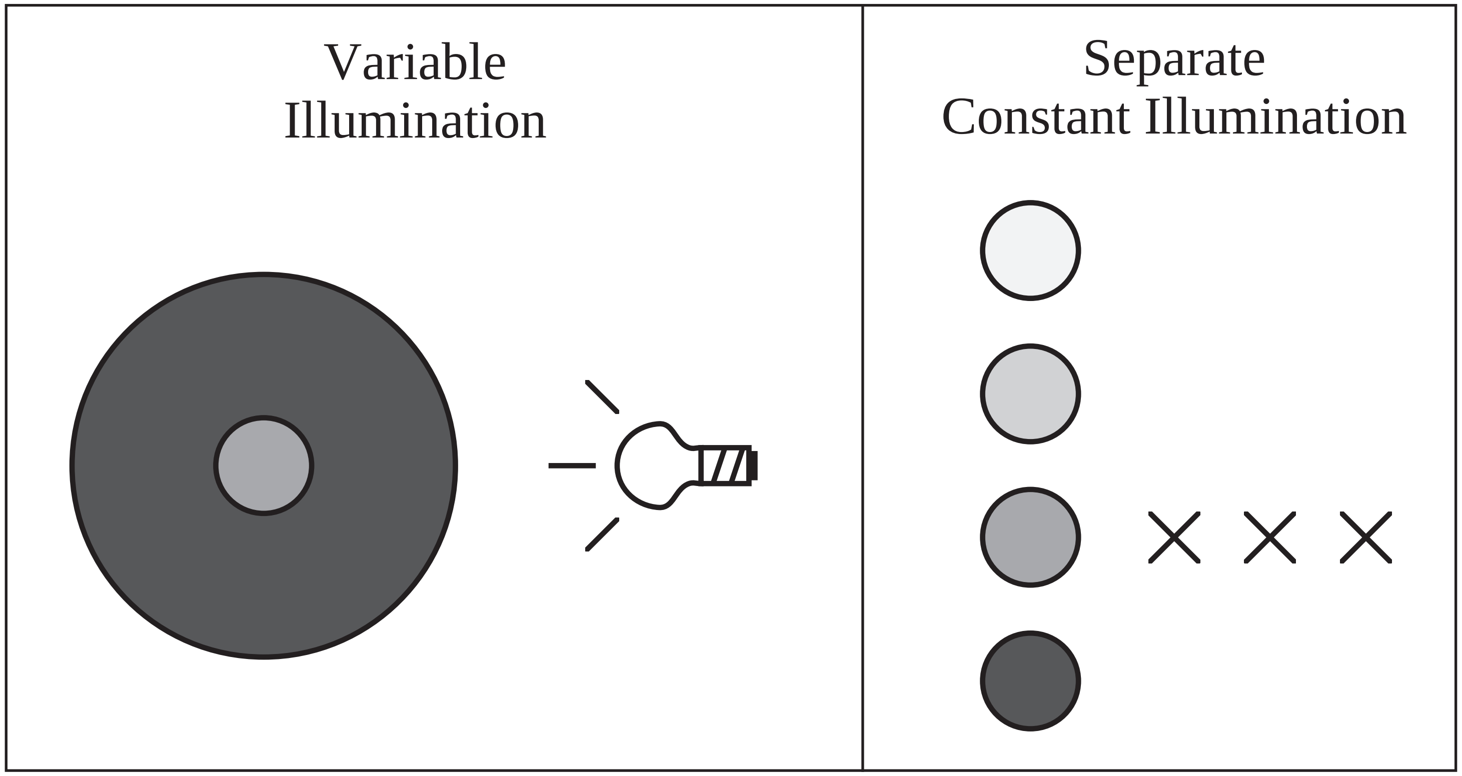
Visual Illusions





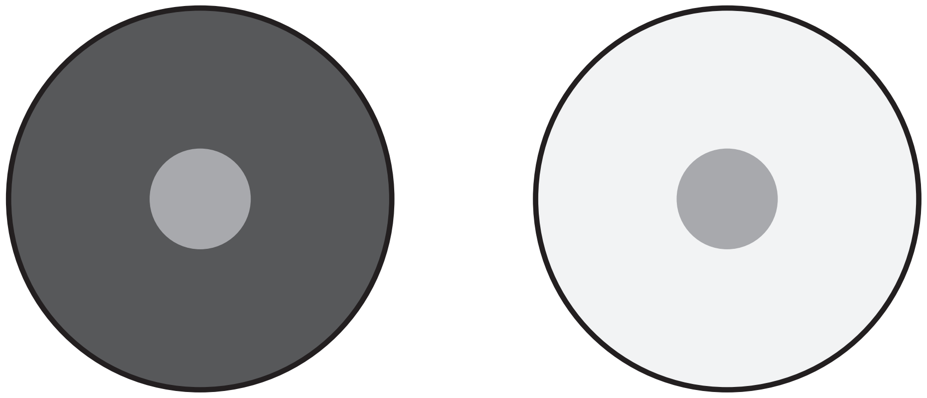
Illusions demostrate the compensatory processing of the visual system. Here we see a bright white triangle and a circle which do not actually exist in the figures.

Vision Normalization



The vision systems normalize scenes so that we are only aware of relative differences in brightness, not absolute brightness.

Brightness Contrast



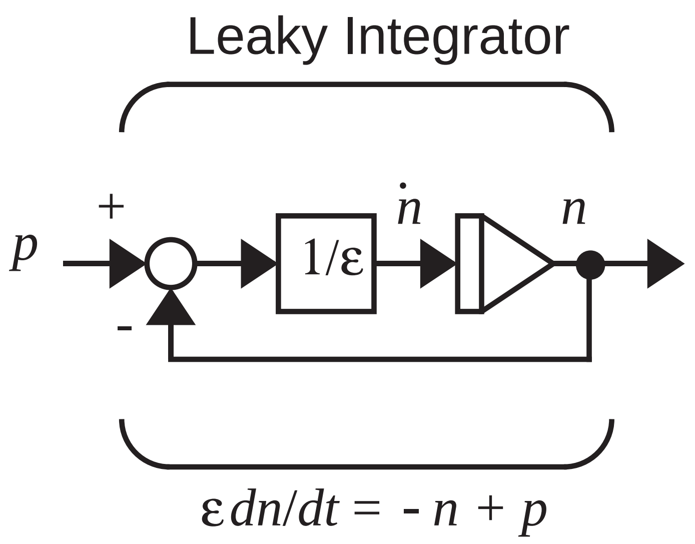
If you look at a point between the two circles, the small inner circle on the left will appear lighter than the small inner circle on the right, although they have the same brightness. It is relatively lighter than its surroundings.

The visual system normalizes the scene. We see relative intensities.

Leaky Integrator

(Building block for basic nonlinear model.)

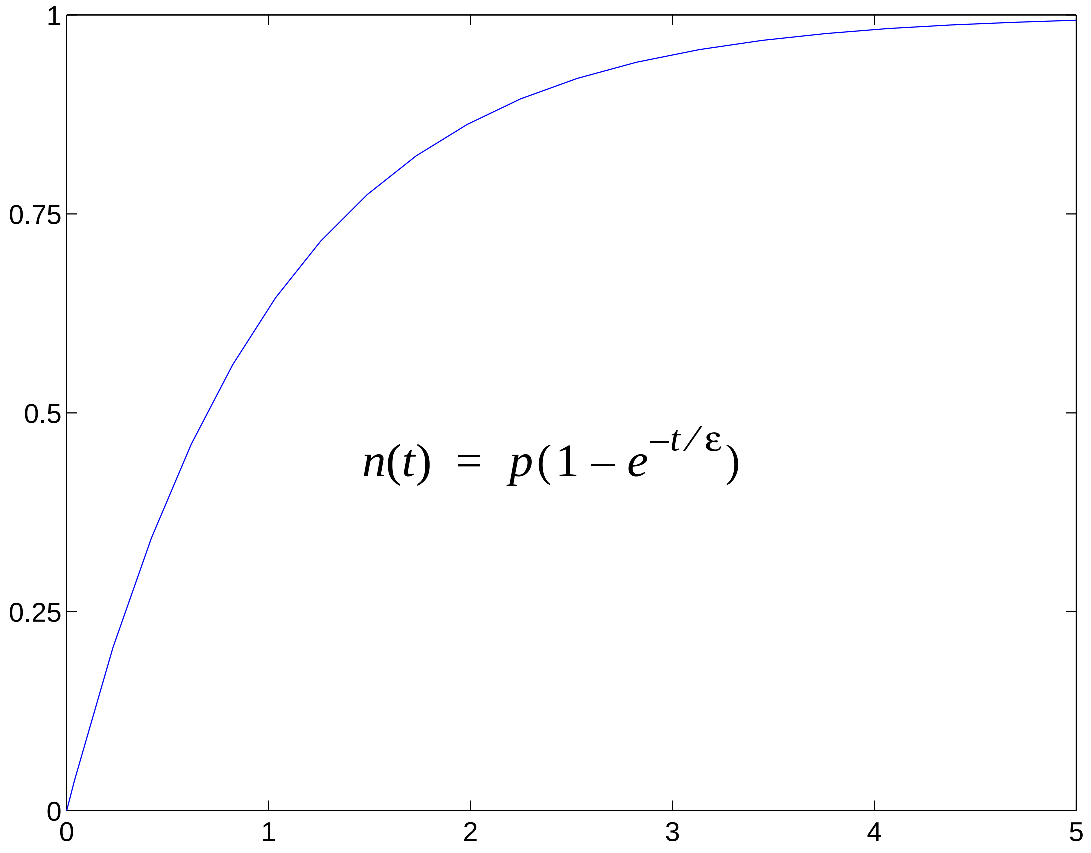
$$\varepsilon \frac{dn(t)}{dt} = -n(t) + p(t)$$



Leaky Integrator Response

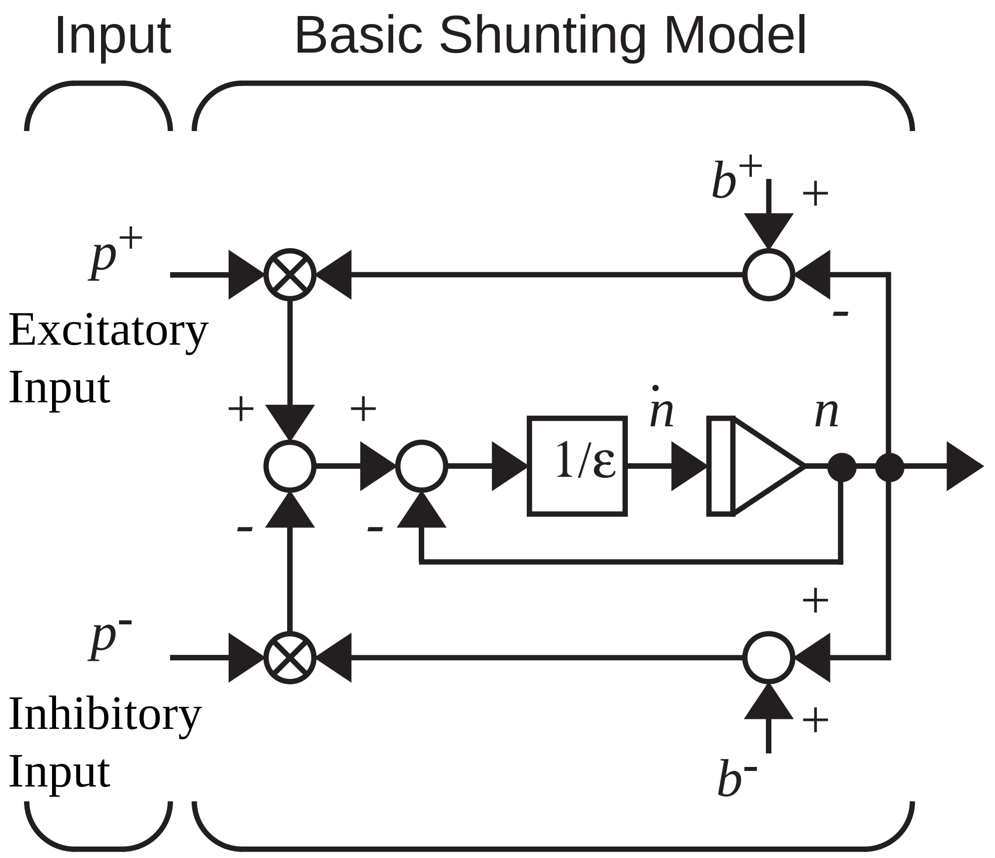
$$n(t) = e^{-t/\varepsilon}n(0) + \frac{1}{\varepsilon}\int_0^t e^{-(t-\tau)/\varepsilon}p(t-\tau)d\tau$$

For a constant input and zero initial conditions:

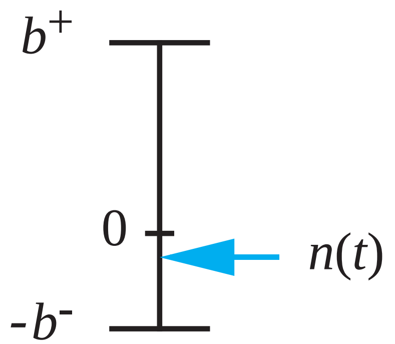


$$n(t) = p(1 - e^{-t/\varepsilon})$$

Shunting Model



$$\varepsilon\, dn/dt = -n + \underbrace{(b^+ - n)p^+}_{\substack{\text{Gain Control} \\ \text{(Sets upper limit)}}} - \underbrace{(n + b^-)p^-}_{\substack{\text{Gain Control} \\ \text{(Sets lower limit)}}}$$

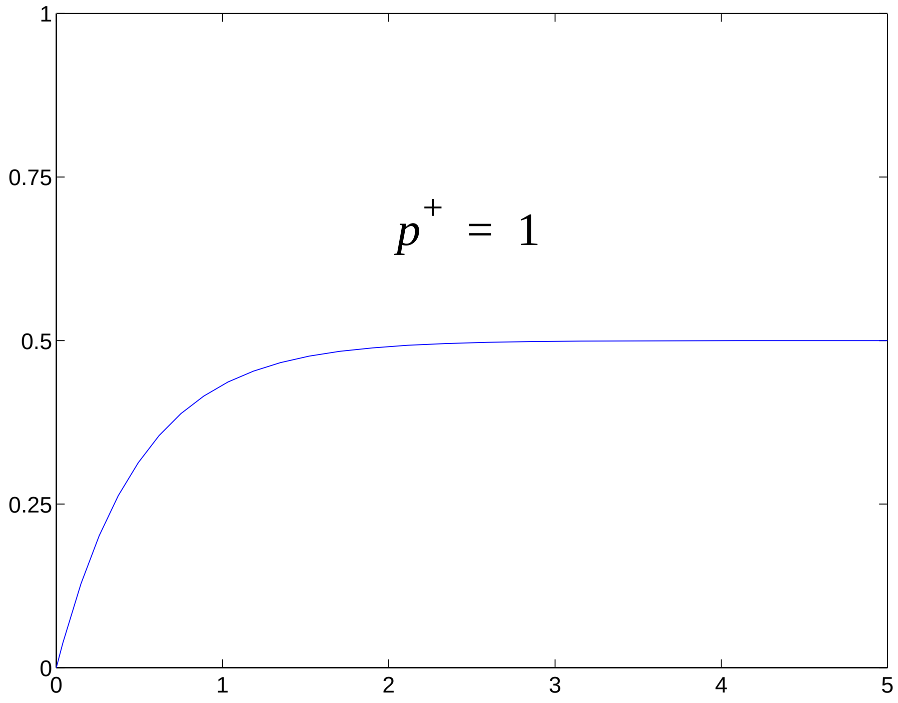


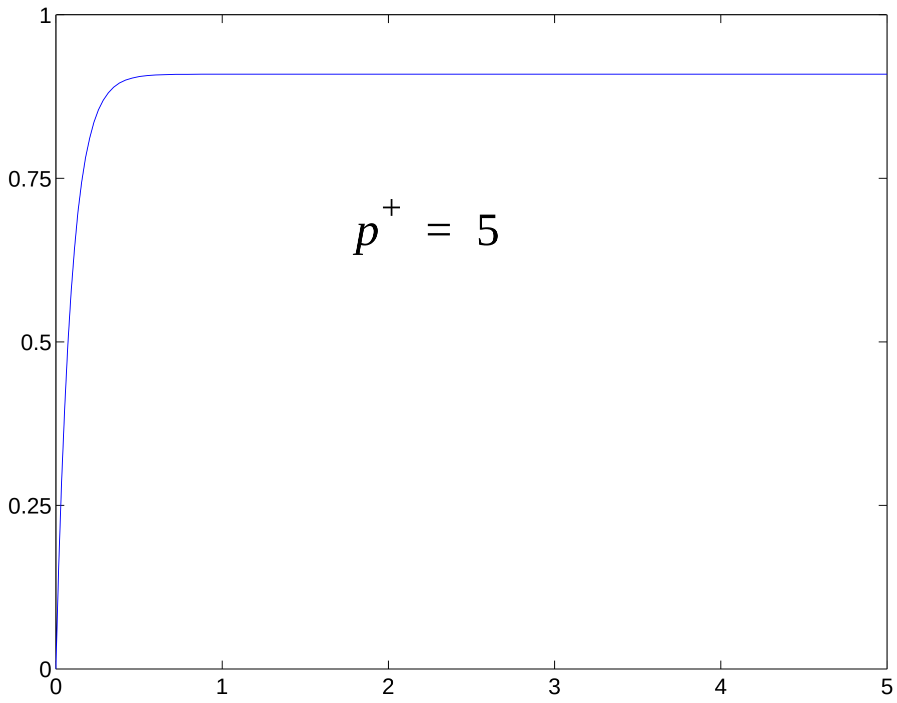
Shunting Model Response

$$\varepsilon \frac{dn(t)}{dt} = -n(t) + (b^{+} - n(t))p^{+} - (n(t) + b^{-})p^{-}$$

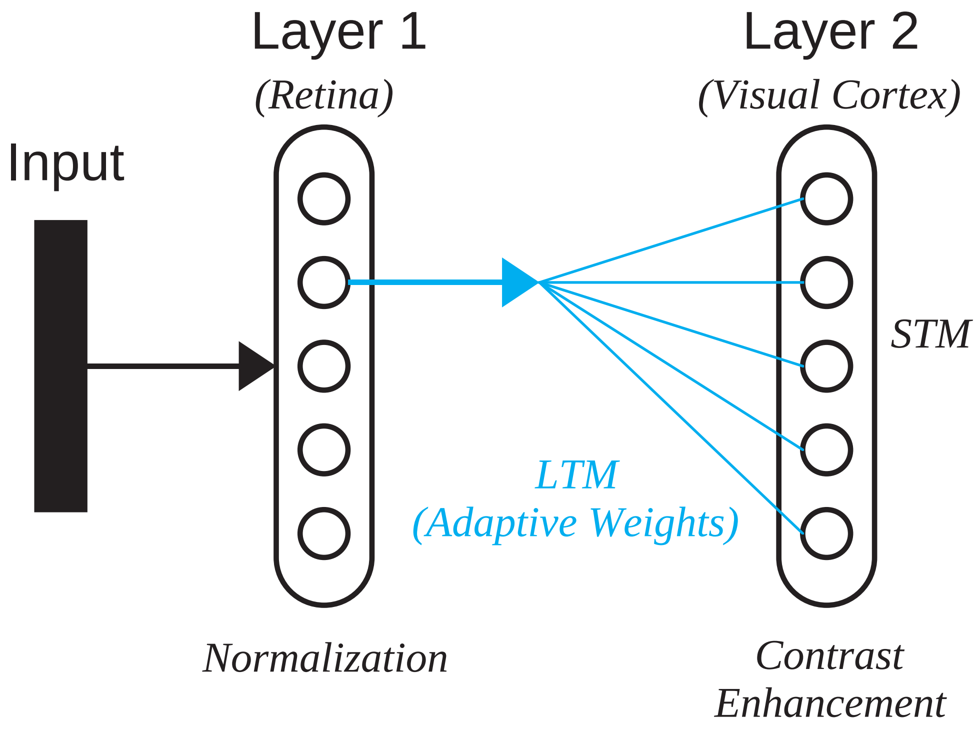
$$b^{+} = 1 \qquad b^{-} = 0 \qquad \varepsilon = 1 \qquad p^{-} = 0$$

Upper limit will be 1, and lower limit will be 0.



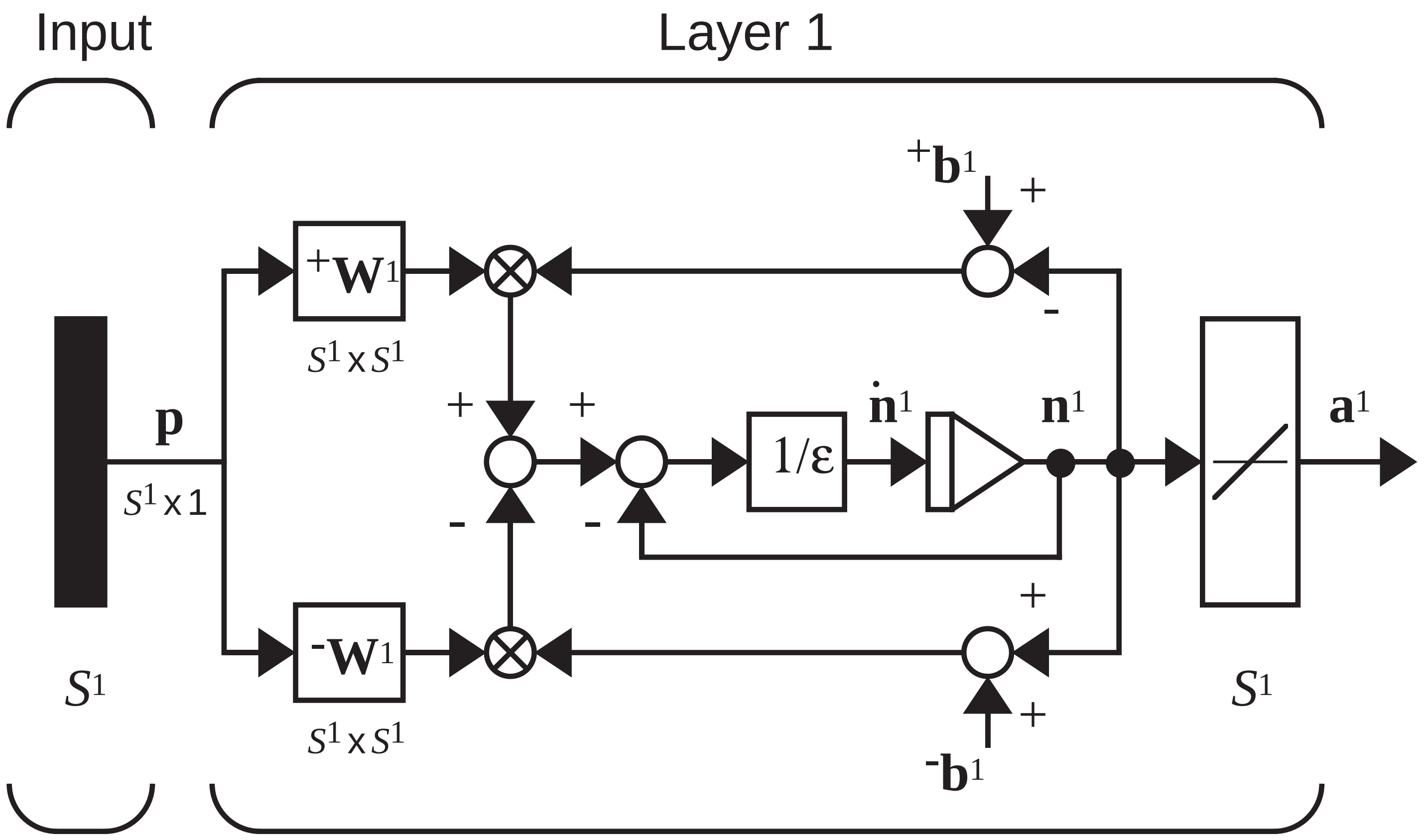


Grossberg Network



LTM - Long Term Memory (Network Weights)
STM - Short Term Memory (Network Outputs)

Layer 1



$$\varepsilon d\mathbf{n}^1/dt = -\mathbf{n}^1 + ({}^{+}\mathbf{b}^1 - \mathbf{n}^1)[{}^{+}\mathbf{W}^1]\mathbf{p} - (\mathbf{n}^1 + {}^{-}\mathbf{b}^1)[{}^{-}\mathbf{W}^1]\mathbf{p}$$

Operation of Layer 1

$$\varepsilon \frac{d\mathbf{n}^1(t)}{dt} = -\mathbf{n}^1(t) + ({}^{+}\mathbf{b}^1 - \mathbf{n}^1(t))[{}^{+}\mathbf{W}^1]\mathbf{p} - (\mathbf{n}^1(t) + {}^{-}\mathbf{b}^1)[{}^{-}\mathbf{W}^1]\mathbf{p}$$

$${}^{-}\mathbf{b}^1 = \mathbf{0}$$

$${}^{+}b_i^1 = {}^{+}b^1$$

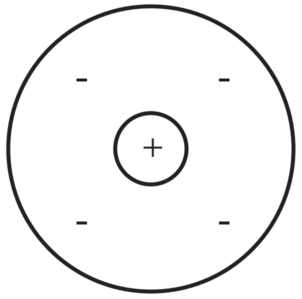
Excitatory Input

$$[{}^{+}\mathbf{W}^1]\mathbf{p} \qquad {}^{+}\mathbf{W}^1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Inhibitory Input

$$[{}^{-}\mathbf{W}^1]\mathbf{p} \qquad {}^{-}\mathbf{W}^1 = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 0 \end{bmatrix}$$

On-Center/
Off-Surround
Connection
Pattern



Normalizes the input while maintaining relative intensities.

Analysis of Normalization

Neuron *i* response:

$$\varepsilon \frac{dn_i^1(t)}{dt} = -n_i^1(t) + ({}^{+}b^1 - n_i^1(t))p_i - n_i^1(t)\sum_{j \neq i} p_j$$

At steady state:

$$0 = -n_i^1 + ({}^{+}b^1 - n_i^1)p_i - n_i^1 \sum_{j \neq i} p_j \quad \Rightarrow \quad n_i^1 = \frac{{}^{+}b^1 p_i}{1 + \sum_{j=1}^{S^1} p_j}$$

Define relative intensity:

$$\bar{p}_i = \frac{p_i}{P} \qquad \text{where} \qquad P = \sum_{j=1}^{S^1} p_j$$

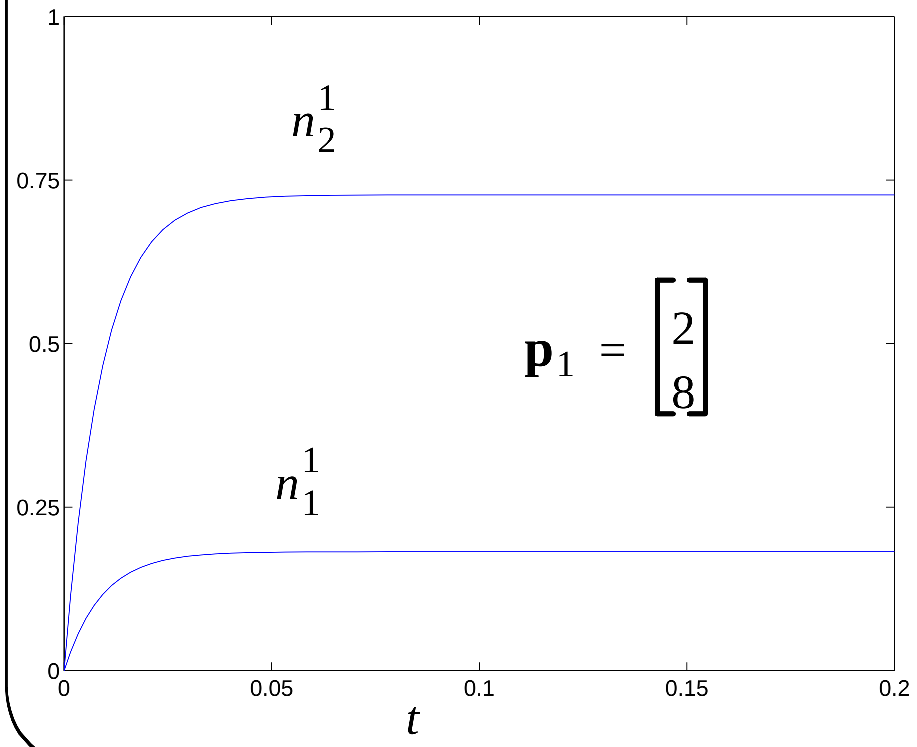
Steady state neuron activity:

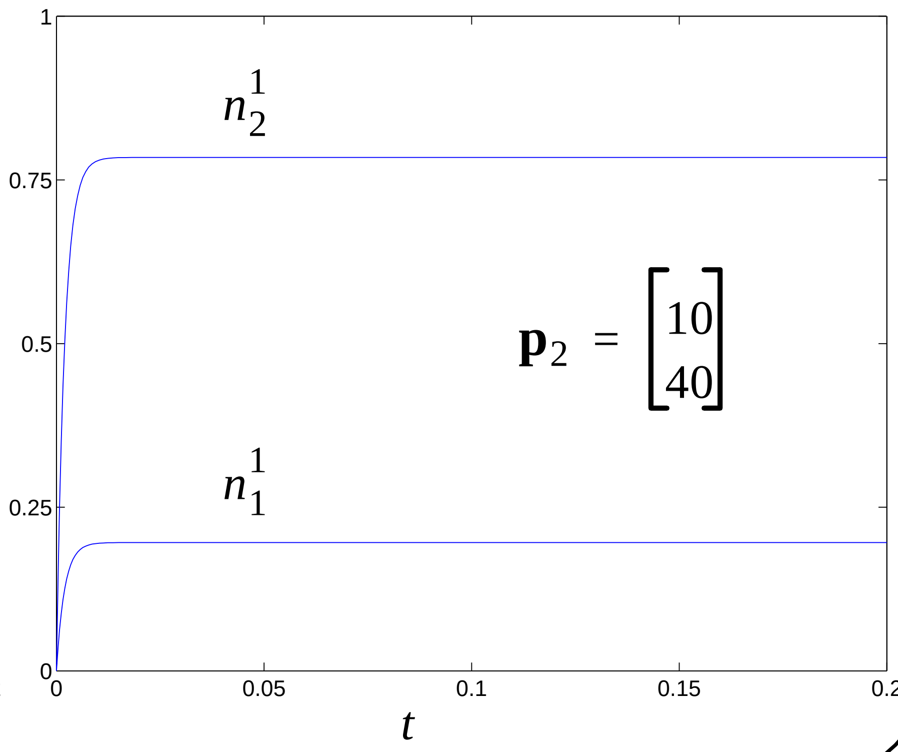
$$n_i^1 = \left(\frac{{}^{+}b^1 P}{1+P}\right)\bar{p}_i \qquad \text{Total activity:} \quad \sum_{j=1}^{S^1} n_j^1 = \sum_{j=1}^{S^1}\left(\frac{{}^{+}b^1 P}{1+P}\right)\bar{p}_j = \left(\frac{{}^{+}b^1 P}{1+P}\right) \leq {}^{+}b^1$$

Layer 1 Example

$$(0.1)\frac{dn_1^1(t)}{dt} = -n_1^1(t) + (1 - n_1^1(t))p_1 - n_1^1(t)p_2$$

$$(0.1)\frac{dn_2^1(t)}{dt} = -n_2^1(t) + (1 - n_2^1(t))p_2 - n_2^1(t)p_1$$

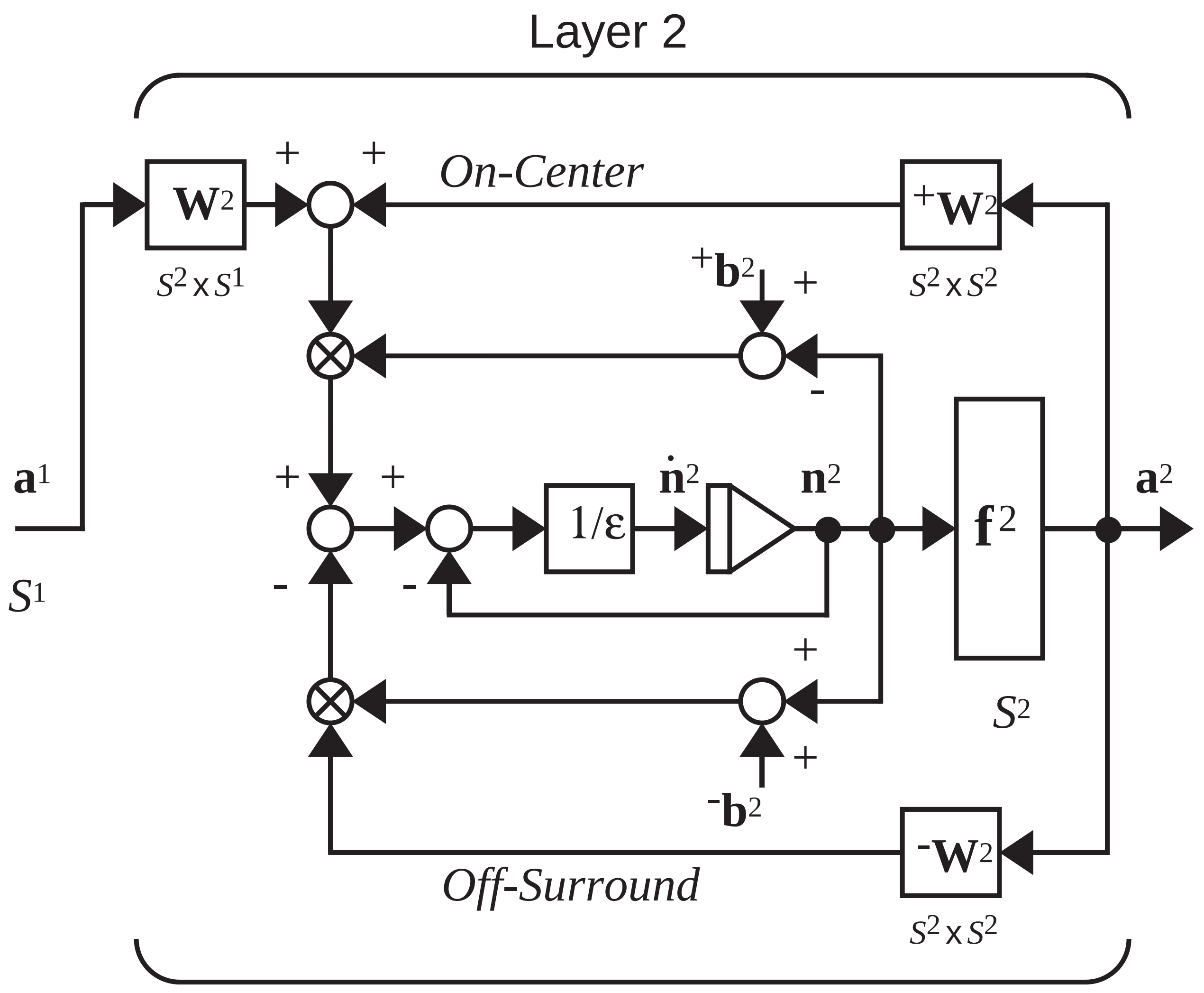




Characteristics of Layer 1

- The network is sensitive to relative intensities of the input pattern, rather than absolute intensities.
- The output of Layer 1 is a normalized version of the input pattern.
- The on-center/off-surround connection pattern and the nonlinear gain control of the shunting model produce the normalization effect.
- The operation of Layer 1 explains the brightness constancy and brightness contrast characteristics of the human visual system.

Layer 2



$$\varepsilon d\mathbf{n}^2/dt = -\mathbf{n}^2 + ({}^{+}\mathbf{b}^2 - \mathbf{n}^2)\{[{}^{+}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2) + \mathbf{W}^2\mathbf{a}^1\}$$
$$-(\mathbf{n}^2 + {}^{-}\mathbf{b}^2)[{}^{-}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2)$$

Layer 2 Operation

$$\varepsilon \frac{d\mathbf{n}^2(t)}{dt} = -\mathbf{n}^2(t) + ({}^{+}\mathbf{b}^2 - \mathbf{n}^2(t))\{[{}^{+}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2(t)) + \mathbf{W}^2\mathbf{a}^1\}$$
$$- (\mathbf{n}^2(t) + {}^{-}\mathbf{b}^2)[{}^{-}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2(t))$$

Excitatory Input:

$$\{[{}^{+}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2(t)) + \mathbf{W}^2\mathbf{a}^1\}$$

${}^{+}\mathbf{W}^2 = {}^{+}\mathbf{W}^1$ (On-center connections)

$\mathbf{W}^2$ (Adaptive weights)

Inhibitory Input:

$$[{}^{-}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2(t))$$

${}^{-}\mathbf{W}^2 = {}^{-}\mathbf{W}^1$ (Off-surround connections)

Layer 2 Example

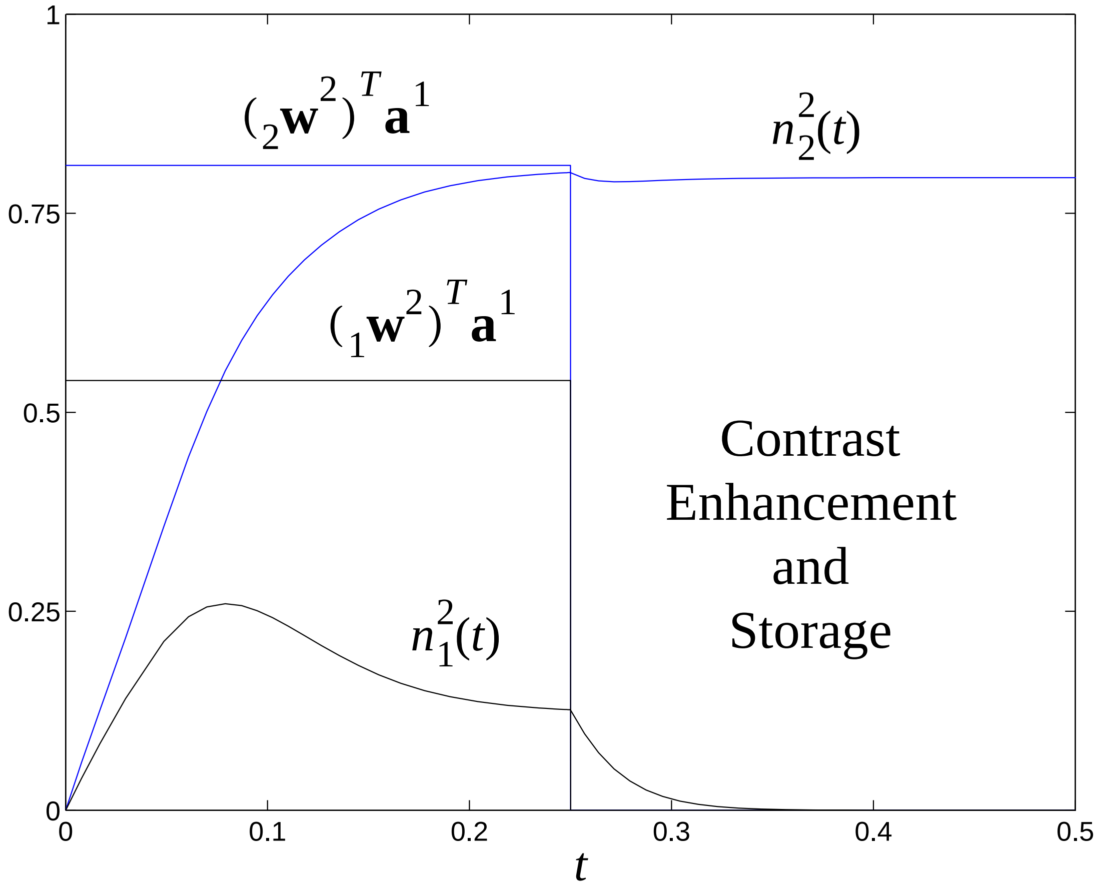
$$\varepsilon = 0.1 \quad {}^{+}\mathbf{b}^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad {}^{-}\mathbf{b}^2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad f^2(n) = \frac{10(n)^2}{1 + (n)^2} \quad \mathbf{W}^2 = \begin{bmatrix} ({}_1\mathbf{w}^2)^T \\ ({}_2\mathbf{w}^2)^T \end{bmatrix} = \begin{bmatrix} 0.9 & 0.45 \\ 0.45 & 0.9 \end{bmatrix}$$

$$(0.1)\frac{dn_1^2(t)}{dt} = -n_1^2(t) + (1 - n_1^2(t))\left\{ f^2(n_1^2(t)) + \overbrace{({}_1\mathbf{w}^2)^T \mathbf{a}^1}^{\text{Correlation between prototype 1 and input.}} \right\} - n_1^2(t) f^2(n_2^2(t))$$

$$(0.1)\frac{dn_2^2(t)}{dt} = -n_2^2(t) + (1 - n_2^2(t))\left\{ f^2(n_2^2(t)) + \overbrace{({}_2\mathbf{w}^2)^T \mathbf{a}^1}^{\text{Correlation between prototype 2 and input.}} \right\} - n_2^2(t) f^2(n_1^2(t)) \; .$$

Layer 2 Response

$$\mathbf{a}^1 = \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$$



Input to neuron 1:

$$({}_1\mathbf{w}^2)^T\mathbf{a}^1 = \begin{bmatrix} 0.9 & 0.45 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix} = 0.54$$

Input to neuron 2:

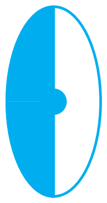
$$({}_2\mathbf{w}^2)^T\mathbf{a}^1 = \begin{bmatrix} 0.45 & 0.9 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix} = 0.81$$

Characteristics of Layer 2

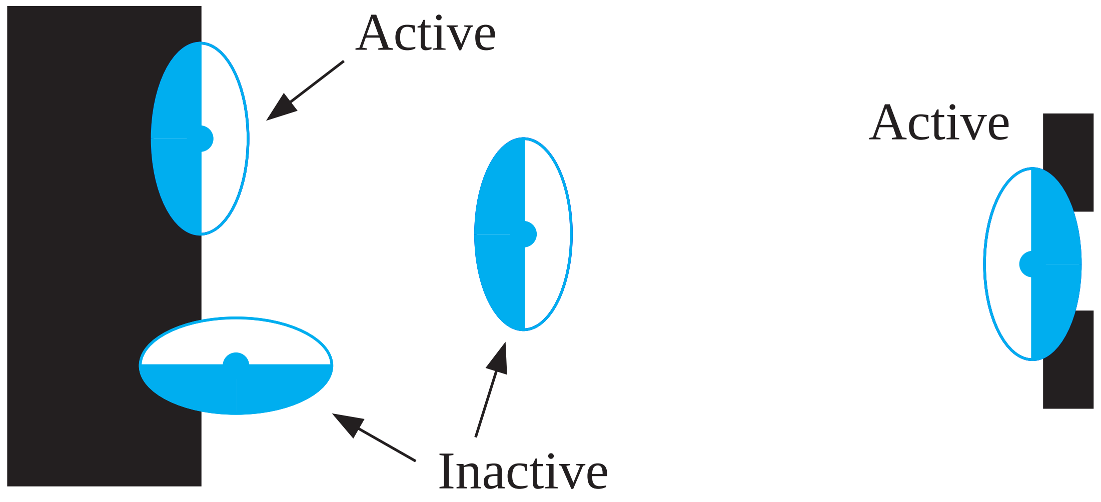
- As in the Hamming and Kohonen networks, the inputs to Layer 2 are the inner products between the prototype patterns (rows of the weight matrix $\mathbf{W}^2$) and the output of Layer 1 (normalized input pattern).
- The nonlinear feedback enables the network to store the output pattern (pattern remains after input is removed).
- The on-center/off-surround connection pattern causes contrast enhancement (large inputs are maintained, while small inputs are attenuated).

15

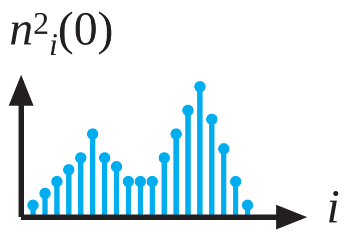
Oriented Receptive Field

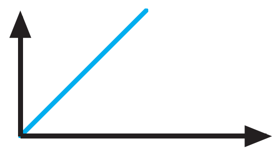
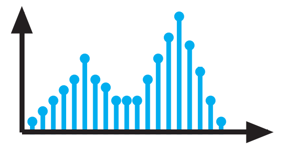
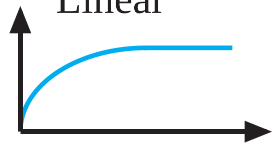
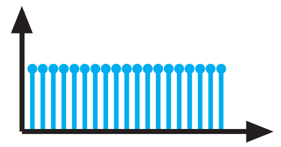
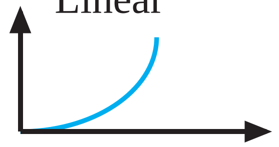
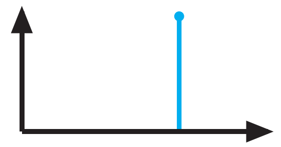
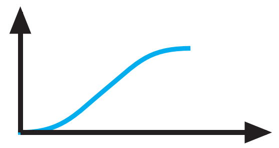
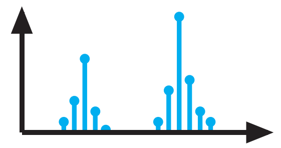


When an oriented receptive field is used, instead of an on-center/off-surround receptive field, the emergent segmentation problem can be understood.



Choice of Transfer Function



$f^2(n)$	Stored Pattern $\mathbf{n}^2(\infty)$	Comments
Linear 		Perfect storage of any pattern, but amplifies noise.
Slower than Linear 		Amplifies noise, reduces contrast.
Faster than Linear 		Winner-take-all, suppresses noise, quantizes total activity.
Sigmoid 		Supresses noise, contrast enhances, not quantized.

Adaptive Weights

Hebb Rule with Decay

$$\frac{dw^2_{i,j}(t)}{dt} = \alpha\{-w^2_{i,j}(t) + n^2_i(t)n^1_j(t)\}$$

Instar Rule
(Gated Learning)

$$\frac{dw^2_{i,j}(t)}{dt} = \alpha n^2_i(t)\{-w^2_{i,j}(t) + n^1_j(t)\}$$

Learn when $n_i^2(t)$ is active.

Vector Instar Rule

$$\frac{d[{}_i\mathbf{w}^2(t)]}{dt} = \alpha n^2_i(t)\{-[{}_i\mathbf{w}^2(t)] + \mathbf{n}^1(t)\}$$

Example

$$\frac{dw^2_{1,1}(t)}{dt} = n^2_1(t)\{-w^2_{1,1}(t) + n^1_1(t)\}$$

$$\frac{dw^2_{1,2}(t)}{dt} = n^2_1(t)\{-w^2_{1,2}(t) + n^1_2(t)\}$$

$$\frac{dw^2_{2,1}(t)}{dt} = n^2_2(t)\{-w^2_{2,1}(t) + n^1_1(t)\}$$

$$\frac{dw^2_{2,2}(t)}{dt} = n^2_2(t)\{-w^2_{2,2}(t) + n^1_2(t)\}$$

Response of Adaptive Weights

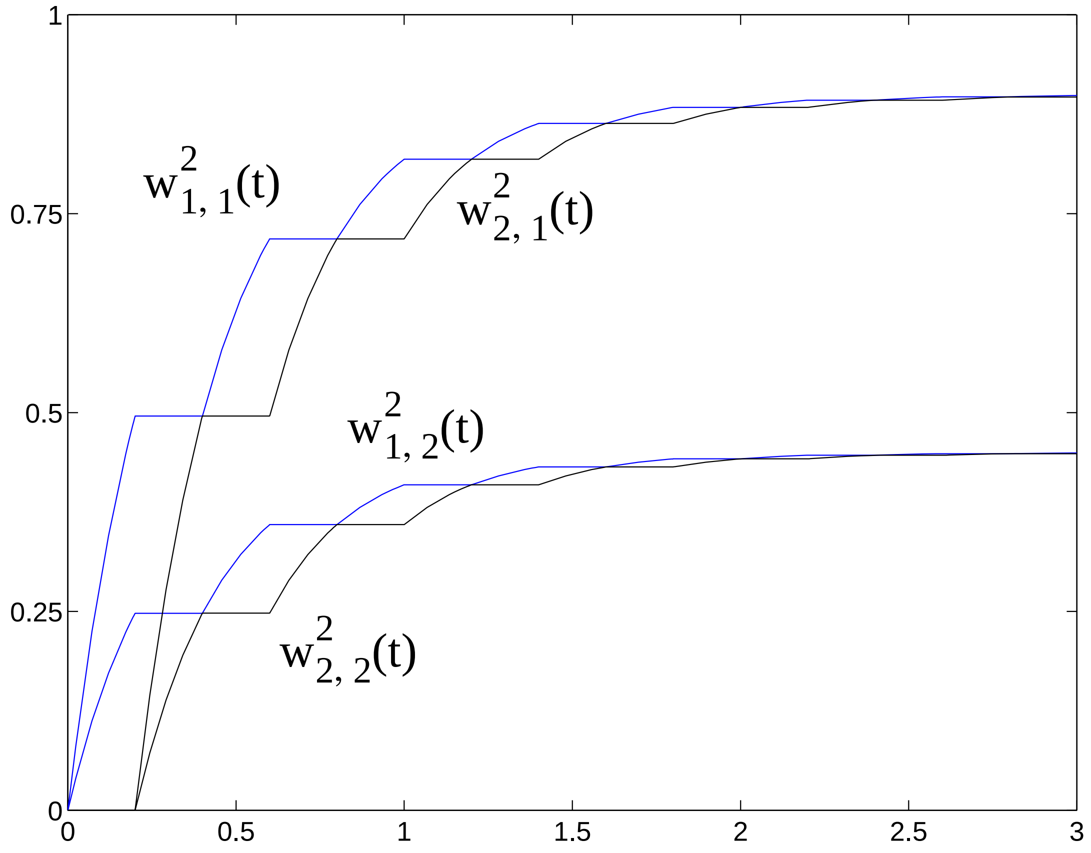
Two different input patterns are alternately presented to the network for periods of 0.2 seconds at a time.

For Pattern 1:

$$\mathbf{n}^1 = \begin{bmatrix} 0.9 \\ 0.45 \end{bmatrix} \qquad \mathbf{n}^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

For Pattern 2:

$$\mathbf{n}^1 = \begin{bmatrix} 0.45 \\ 0.9 \end{bmatrix} \qquad \mathbf{n}^2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



The first row of the weight matrix is updated when $n_1^2(t)$ is active, and the second row of the weight matrix is updated when $n_2^2(t)$ is active.

Relation to Kohonen Law

Grossberg Learning (Continuous-Time)

$$\frac{d[{}_i\mathbf{w}^2(t)]}{dt} = \alpha n_i^2(t)\{-[{}_i\mathbf{w}^2(t)] + \mathbf{n}^1(t)\}$$

Euler Approximation for the Derivative

$$\frac{d[{}_i\mathbf{w}^2(t)]}{dt} \approx \frac{{}_i\mathbf{w}^2(t+\Delta t) - {}_i\mathbf{w}^2(t)}{\Delta t}$$

Discrete-Time Approximation to Grossberg Learning

$${}_i\mathbf{w}^2(t+\Delta t) = {}_i\mathbf{w}^2(t) + \alpha(\Delta t) n_i^2(t)\{-{}_i\mathbf{w}^2(t) + \mathbf{n}^1(t)\}$$

Relation to Kohonen Law

Rearrange Terms

$$ {}_i\mathbf{w}^2(t+\Delta t) = \{1-\alpha(\Delta t)n_i^2(t)\}\,{}_i\mathbf{w}^2(t) + \alpha(\Delta t)n_i^2(t)\{\mathbf{n}^1(t)\} $$

Assume Winner-Take-All Competition

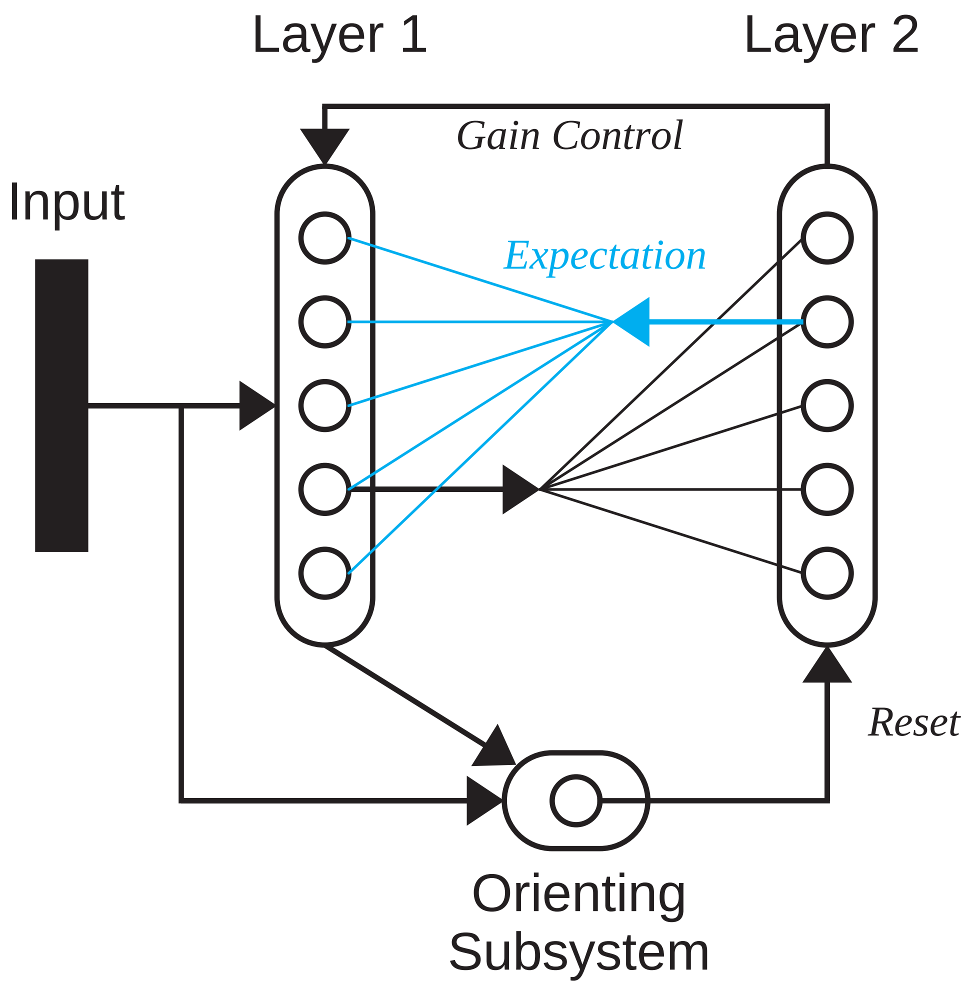
$$ {}_{i*}\mathbf{w}^2(t+\Delta t) = \{1-\alpha'\}\,{}_{i*}\mathbf{w}^2(t) + \{\alpha\}'\mathbf{n}^1(t) \qquad \text{where} \qquad \alpha' = \alpha(\Delta t)n_{i*}^2(t) $$

Compare to Kohonen Rule

$$ {}_{i*}\mathbf{w}(q) = (1-\alpha)\,{}_{i*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q) $$

Adaptive Resonance Theory (ART)

Basic ART Architecture



ART Subsystems

Layer 1
- Normalization
- Comparison of input pattern and expectation

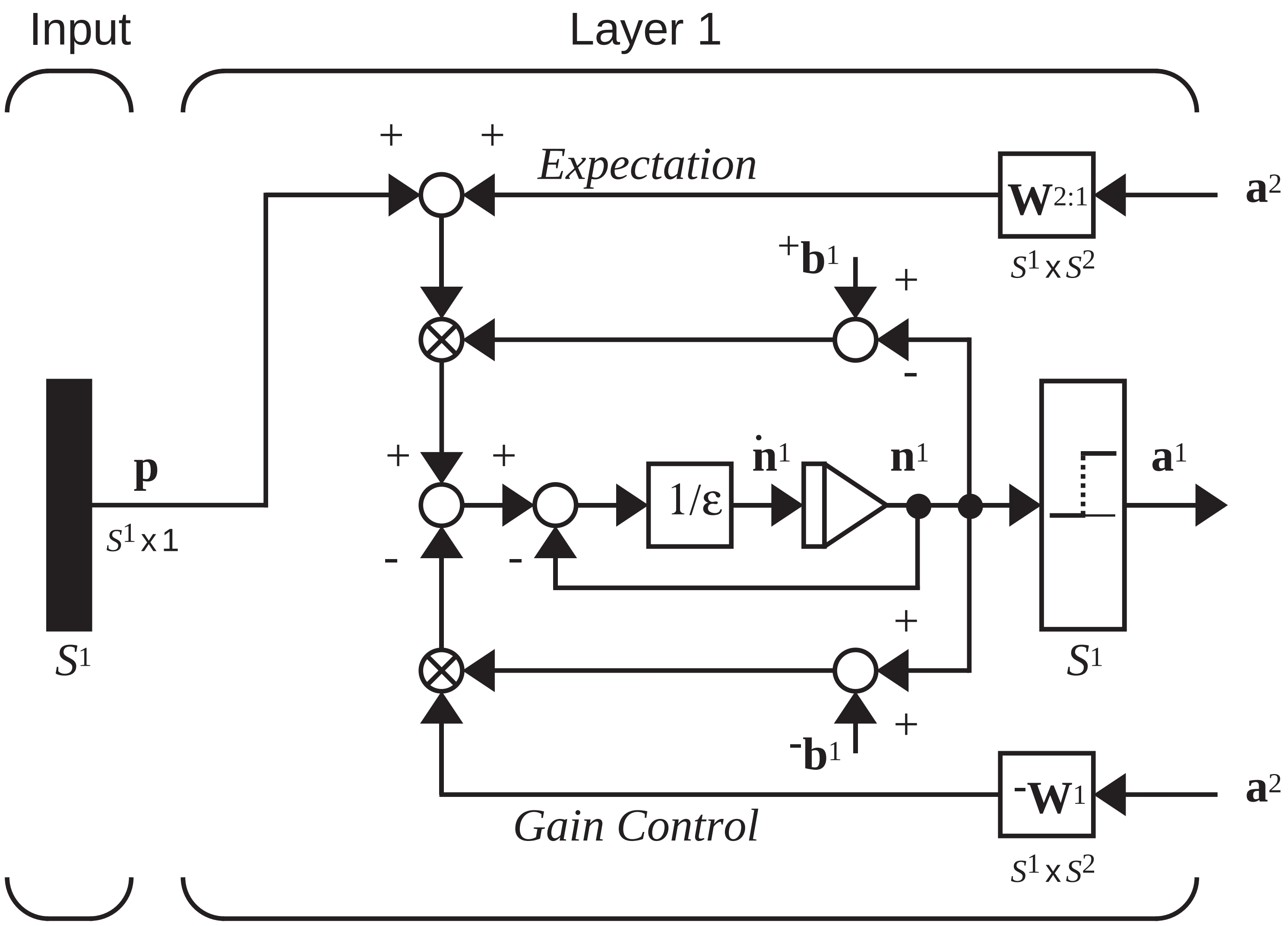
L1-L2 Connections (Instars)
- Perform clustering operation.
- Each row of $W^{1:2}$ is a prototype pattern.

Layer 2
- Competition, contrast enhancement

L2-L1 Connections (Outstars)
- Expectation
- Perform pattern recall.
- Each column of $W^{2:1}$ is a prototype pattern

Orienting Subsystem
- Causes a reset when expectation does not match input
- Disables current winning neuron

Layer 1



$$\varepsilon\, d\mathbf{n}^1/dt = -\mathbf{n}^1 + ({}^{+}\mathbf{b}^1 - \mathbf{n}^1)\{\mathbf{p} + \mathbf{W}^{2:1}\mathbf{a}^2\} - (\mathbf{n}^1 + {}^{-}\mathbf{b}^1)[{}^{-}\mathbf{W}^1]\mathbf{a}^2$$

Layer 1 Operation

Shunting Model

$$\varepsilon \frac{d\mathbf{n}^1(t)}{dt} = -\mathbf{n}^1(t) + ({}^{+}\mathbf{b}^1 - \mathbf{n}^1(t))\underbrace{\{\mathbf{p} + \mathbf{W}^{2:1}\mathbf{a}^2(t)\}}_{\substack{\text{Excitatory Input}\\\text{(Comparison with Expectation)}}} - (\mathbf{n}^1(t) + {}^{-}\mathbf{b}^1)\underbrace{[{}^{-}\mathbf{W}^1]\mathbf{a}^2(t)}_{\substack{\text{Inhibitory Input}\\\text{(Gain Control)}}}$$

$$\mathbf{a}^1 = \mathbf{hardlim}^{+}(\mathbf{n}^1)$$

$$hardlim^{+}(n) = \begin{cases} 1, & n > 0 \\ 0, & n \le 0 \end{cases}$$

Excitatory Input to Layer 1

$$\mathbf{p} + \mathbf{W}^{2:1}\mathbf{a}^2(t)$$

Suppose that neuron *j* in Layer 2 has won the competition:

$$\mathbf{W}^{2:1}\mathbf{a}^2 = \begin{bmatrix} \mathbf{w}_1^{2:1} & \mathbf{w}_2^{2:1} & \dots & \mathbf{w}_j^{2:1} & \dots & \mathbf{w}_{S^2}^{2:1} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} = \mathbf{w}_j^{2:1} \qquad (j\text{th column of } \mathbf{W}^{2:1})$$

Therefore the excitatory input is the sum of the input pattern and the L2-L1 expectation:

$$\mathbf{p} + \mathbf{W}^{2:1}\mathbf{a}^2 = \mathbf{p} + \mathbf{w}_j^{2:1}$$

Inhibitory Input to Layer 1

Gain Control

$$[{}^{-}\mathbf{W}^1]\mathbf{a}^2(t)$$

$$ {}^{-}\mathbf{W}^1 = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} $$

The gain control will be one when Layer 2 is active (one neuron has won the competition), and zero when Layer 2 is inactive (all neurons having zero output).

Steady State Analysis: Case I

$$\varepsilon \frac{dn_i^1}{dt} = -n_i^1 + ({}^+b^1 - n_i^1)\left\{p_i + \sum_{j=1}^{S^2} w_{i,j}^{2:1} a_j^2\right\} - (n_i^1 + {}^-b^1)\sum_{j=1}^{S^2} a_j^2$$

Case I: Layer 2 inactive (each $a^2{}_j = 0$)

$$\varepsilon \frac{dn_i^1}{dt} = -n_i^1 + ({}^+b^1 - n_i^1)\{p_\mathrm{i}\}$$

In steady state:

$$0 = -n_i^1 + ({}^+b^1 - n_i^1)p_i = -(1+p_i)n_i^1 + {}^+b^1 p_i \quad \Longrightarrow \quad n_i^1 = \frac{{}^+b^1 p_i}{1+p_i}$$

Therefore, if Layer 2 is inactive:

$$\boxed{\mathbf{a}^1 = \mathbf{p}}$$

Steady State Analysis: Case II

Case II: Layer 2 active (one $a^2_j = 1$)

$$\varepsilon \frac{dn_i^1}{dt} = -n_i^1 + ({}^{+}b^1 - n_i^1)\{p_i + w_{i,j}^{2:1}\} - (n_i^1 + {}^{-}b^1)$$

In steady state:

$$\begin{aligned} 0 &= -n_i^1 + ({}^{+}b^1 - n_i^1)\{p_i + w_{i,j}^{2:1}\} - (n_i^1 + {}^{-}b^1) \\ &= -(1 + p_i + w_{i,j}^{2:1} + 1)n_i^1 + ({}^{+}b^1(p_i + w_{i,j}^{2:1}) - {}^{-}b^1) \end{aligned} \Longrightarrow n_i^1 = \frac{{}^{+}b^1(p_i + w_{i,j}^{2:1}) - {}^{-}b^1}{2 + p_i + w_{i,j}^{2:1}}$$

We want Layer 1 to combine the input vector with the expectation from Layer 2, using a logical AND operation:

$n^1_i<0$, if either $w^{2:1}_{i,j}$ or p_i is equal to zero.

$n^1_i>0$, if both $w^{2:1}_{i,j}$ or p_i are equal to one.

$$\left.\begin{aligned} {}^{+}b^1(2) - {}^{-}b^1 > 0 \\ {}^{+}b^1 - {}^{-}b^1 < 0 \end{aligned}\right\} \quad {}^{+}b^1(2) > {}^{-}b^1 > {}^{+}b^1$$

Therefore, if Layer 2 is active, and the biases satisfy these conditions:

$$\boxed{\mathbf{a}^1 = \mathbf{p} \cap \mathbf{w}_j^{2:1}}$$

Layer 1 Summary

If Layer 2 is inactive (each $a^2_j = 0$)

$$\boxed{\mathbf{a}^1 = \mathbf{p}}$$

If Layer 2 is active (one $a^2_j = 1$)

$$\boxed{\mathbf{a}^1 = \mathbf{p} \cap \mathbf{w}_j^{2:1}}$$

Layer 1 Example

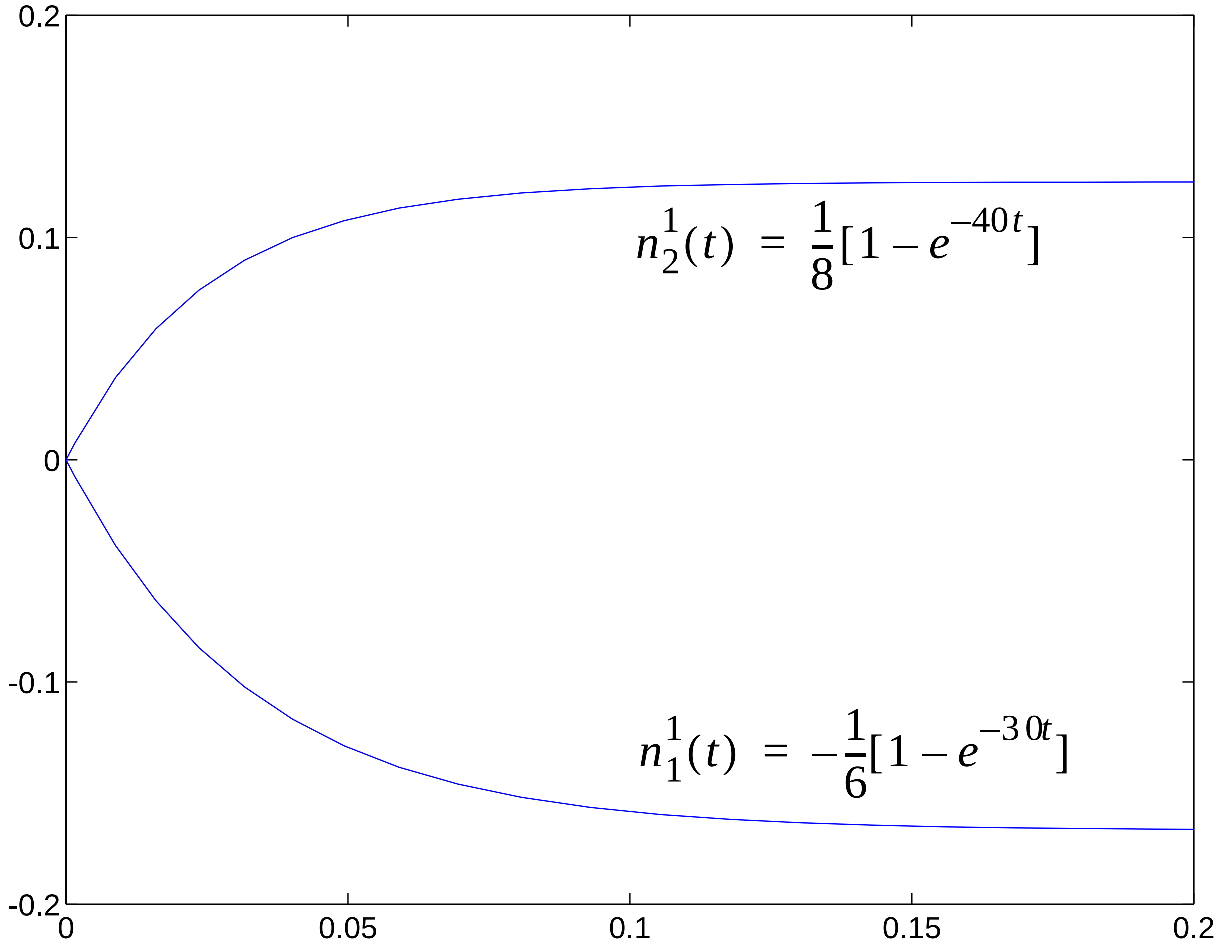
$\varepsilon = 1,\ {}^{+}b^{1} = 1$ and ${}^{-}b^{1} = 1.5$ $\quad \mathbf{W}^{2:1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Assume that Layer 2 is active, and neuron 2 won the competition.

$$\left.\begin{aligned} (0.1)\frac{dn_1^1}{dt} &= -n_1^1 + (1 - n_1^1)\{p_1 + w_{1,2}^{2:1}\} - (n_1^1 + 1.5) \\ &= -n_1^1 + (1 - n_1^1)\{0 + 1\} - (n_1^1 + 1.5) = -3n_1^1 - 0.5 \end{aligned}\right\} \quad \frac{dn_1^1}{dt} = -30n_1^1 - 5$$

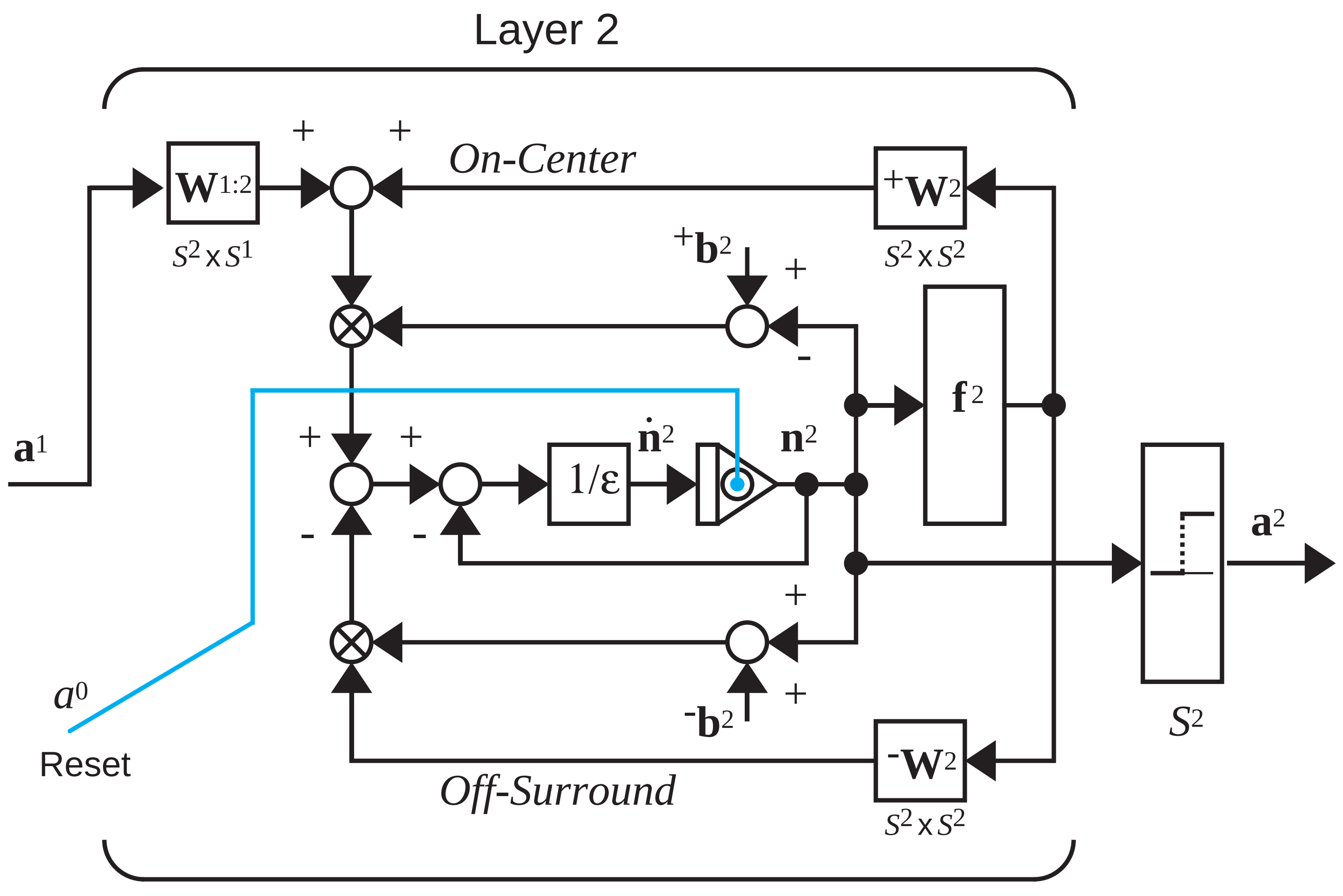
$$\left.\begin{aligned} (0.1)\frac{dn_2^1}{dt} &= -n_2^1 + (1 - n_2^1)\{p_2 + w_{2,2}^{2:1}\} - (n_2^1 + 1.5) \\ &= -n_2^1 + (1 - n_2^1)\{1 + 1\} - (n_2^1 + 1.5) = -4n_2^1 + 0.5 \end{aligned}\right\} \quad \frac{dn_2^1}{dt} = -40n_2^1 + 5$$

Example Response



$$\mathbf{p} \cap \mathbf{w}_2^{2:1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cap \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \mathbf{a}^1$$

Layer 2



$$\varepsilon\, d\mathbf{n}^2/dt = -\mathbf{n}^2 + ({}^{+}\mathbf{b}^2 - \mathbf{n}^2)\{[{}^{+}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2) + \mathbf{W}^{1:2}\mathbf{a}^1\}$$
$$- (\mathbf{n}^2 + {}^{-}\mathbf{b}^2)[{}^{-}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2)$$

Layer 2 Operation

Shunting Model

$$\varepsilon \frac{d\mathbf{n}^2(t)}{dt} = -\mathbf{n}^2(t)$$

$$\boxed{} + ({}^{+}\mathbf{b}^2 - \mathbf{n}^2(t))\underbrace{\{\overbrace{[{}^{+}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2(t))}^{\text{On-Center Feedback}} + \overbrace{\mathbf{W}^{1:2}\mathbf{a}^1}^{\text{Adaptive Instars}}\}}_{\text{Excitatory Input}}$$

$$-(\mathbf{n}^2(t) + {}^{-}\mathbf{b}^2)\underbrace{\overbrace{[{}^{-}\mathbf{W}^2]\mathbf{f}^2(\mathbf{n}^2(t))}^{\text{Off-Surround Feedback}}}_{\text{Inhibitory Input}}$$

Layer 2 Example

$$\varepsilon = 0.1 \qquad {}^{+}\mathbf{b}^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad {}^{-}\mathbf{b}^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{W}^{1:2} = \begin{bmatrix} ({}_1\mathbf{w}^{1:2})^T \\ ({}_2\mathbf{w}^{1:2})^T \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 1 & 0 \end{bmatrix}$$

$$f^2(n) = \begin{cases} 10(n)^2, & n \geq 0 \\ 0\,, & n < 0 \end{cases}$$

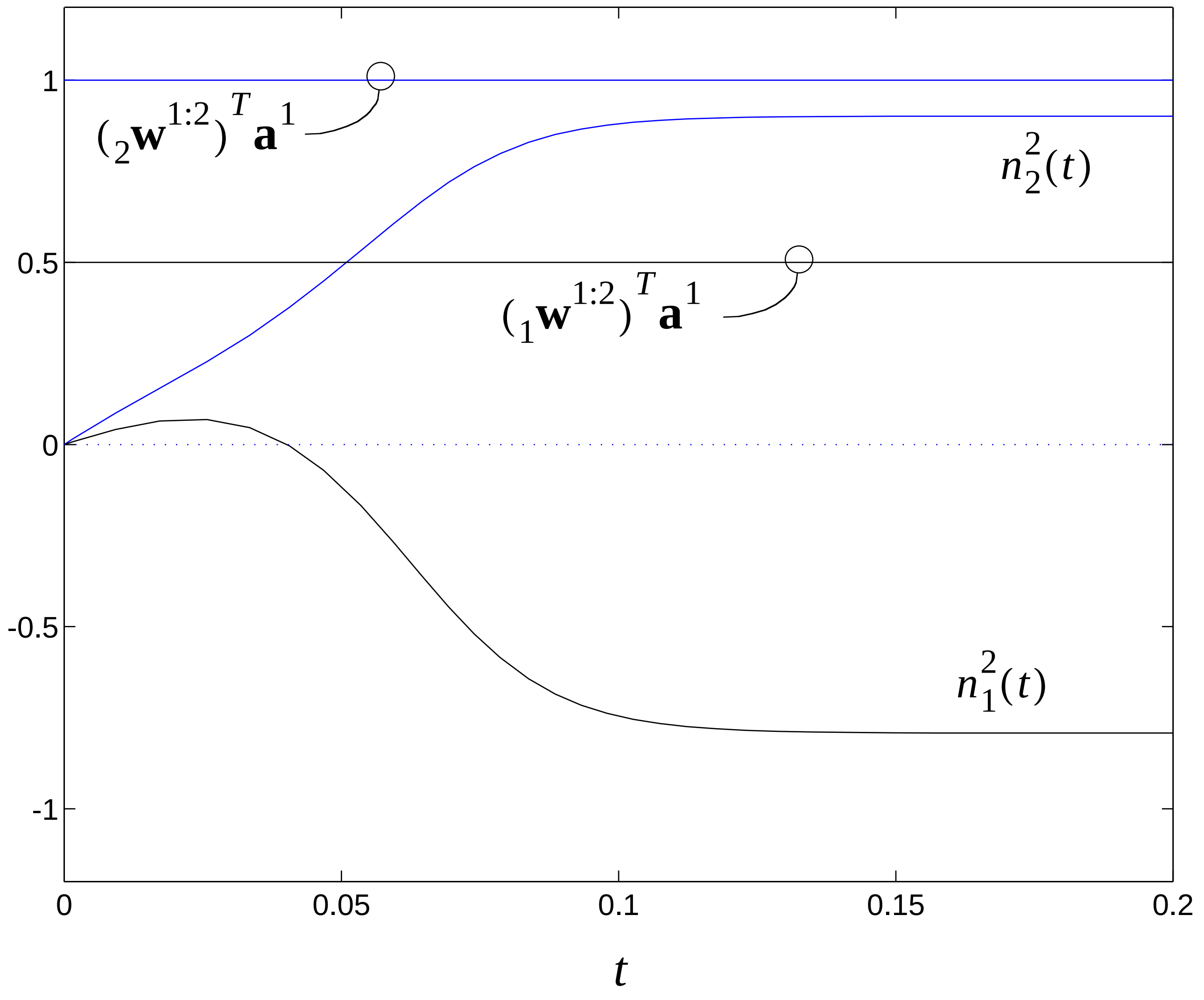
(Faster than linear, winner-take-all)

$$(0.1)\frac{dn_1^2(t)}{dt} = -n_1^2(t) + (1 - n_1^2(t))\left\{ f^2(n_1^2(t)) + ({}_1\mathbf{w}^{1:2})^T \mathbf{a}^1 \right\} - (n_1^2(t) + 1) f^2(n_2^2(t))$$

$$(0.1)\frac{\mathrm{d}n_2^2(t)}{\mathrm{d}t} = -n_2^2(t) + (1 - n_2^2(t))\left\{ f^2(n_2^2(t)) + ({}_2\mathbf{w}^{1:2})^T \mathbf{a}^1 \right\} - (n_2^2(t) + 1) f^2(n_1^2(t)) \,.$$

Example Response

$\mathbf{a}^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$



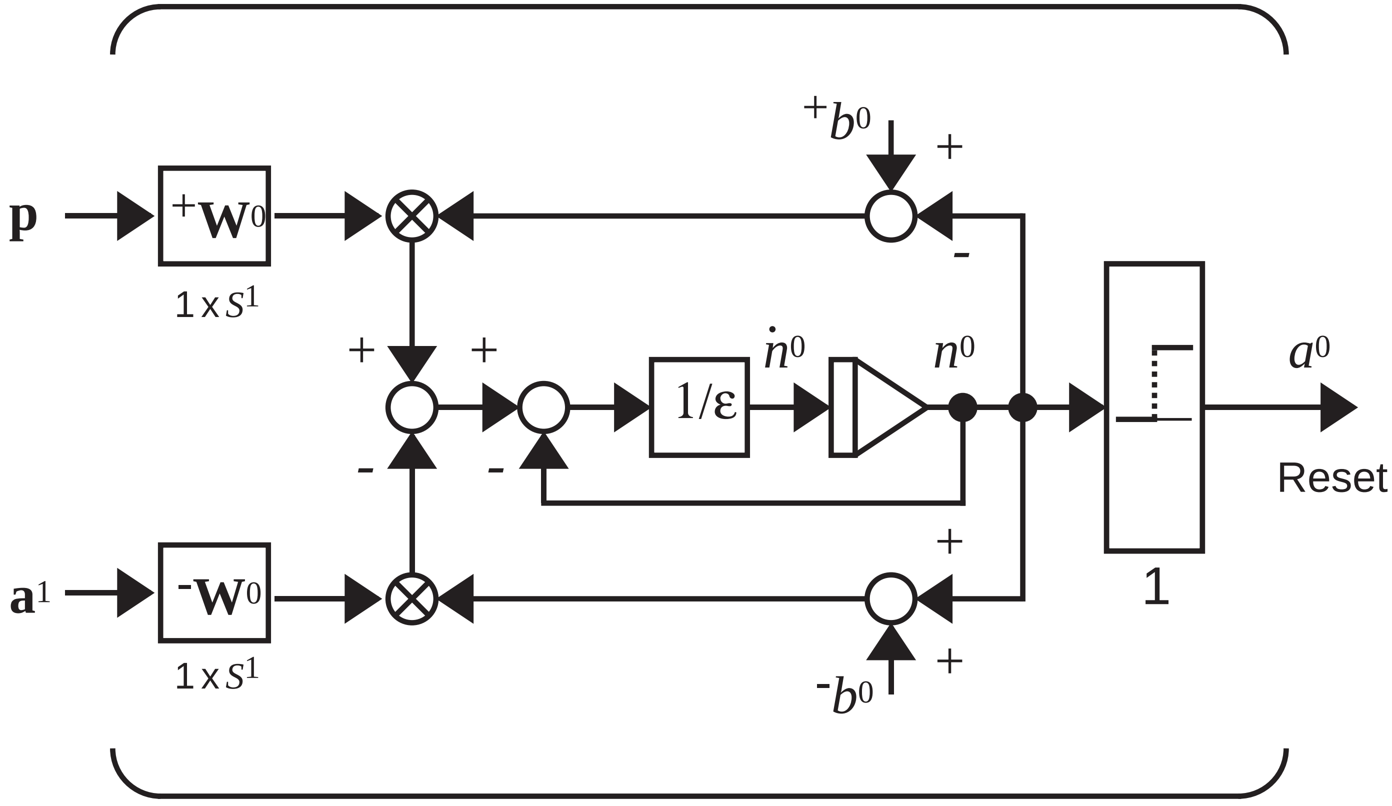
$\mathbf{a}^2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Layer 2 Summary

$$a_i^2 = \begin{cases} 1\,, & \text{if}((_{\text{i}}\mathbf{w}^{1:2})^T\mathbf{a}^1 = max[(_j\mathbf{w}^{1:2})^T\mathbf{a}^1]) \\ 0\,, & \text{otherwise} \end{cases}$$

Orienting Subsystem

Orienting Subsystem



$$\varepsilon\, dn^0/dt = -n^0 + (^{+}b^0 - n^0)[^{+}\mathbf{W}^0]\mathbf{p} - (n^0 + {}^{-}b^0)[^{-}\mathbf{W}^0]\mathbf{a}^1$$

Purpose: Determine if there is a sufficient match between the L2-L1 expectation ($\mathbf{a}^1$) and the input pattern (**p**).

Orienting Subsystem Operation

$$\varepsilon\frac{dn^0(t)}{dt} = -n^0(t) + ({}^{+}b^0 - n^0(t))\underbrace{\{{}^{+}\mathbf{W}^0\mathbf{p}\}}_{\text{Excitatory Input}} - (n^0(t) + {}^{-}b^0)\underbrace{\{{}^{-}\mathbf{W}^0\mathbf{a}^1\}}_{\text{Inhibitory Input}}$$

Excitatory Input

$$\rightarrow \quad {}^{+}\mathbf{W}^0\mathbf{p} = \begin{bmatrix}\alpha & \alpha & \dots & \alpha\end{bmatrix}\mathbf{p} = \alpha\sum_{j=1}^{S^1} p_j = \alpha\|\mathbf{p}\|^2$$

Inhibitory Input

$$\rightarrow \quad {}^{-}\mathbf{W}^0\mathbf{a}^1 = \begin{bmatrix}\beta & \beta & \dots & \beta\end{bmatrix}\mathbf{a}^1 = \beta\sum_{j=1}^{S^1} a_j^1(t) = \beta\|\mathbf{a}^1\|^2$$

When the excitatory input is larger than the inhibitory input, the Orienting Subsystem will be driven on.

Steady State Operation

$$0 = -n^0 + ({}^{+}b^0 - n^0)\{\alpha\|\mathbf{p}\|^2\} - (n^0 + {}^{-}b^0)\left\{\beta\left\|\mathbf{a}^1\right\|^2\right\}$$

$$= -(1 + \alpha\|\mathbf{p}\|^2 + \beta\left\|\mathbf{a}^1\right\|^2)n^0 + {}^{+}b^0(\alpha\|\mathbf{p}\|^2) - {}^{-}b^0(\beta\left\|\mathbf{a}^1\right\|^2)$$

$$n^0 = \frac{{}^{+}b^0(\alpha\|\mathbf{p}\|^2) - {}^{-}b^0(\beta\left\|\mathbf{a}^1\right\|^2)}{(1 + \alpha\|\mathbf{p}\|^2 + \beta\left\|\mathbf{a}^1\right\|^2)}$$

Let $\quad {}^{+}b^0 = {}^{-}b^0 = 1$

$$n^0 > 0 \qquad \text{if} \qquad \frac{\left\|\mathbf{a}^1\right\|^2}{\|\mathbf{p}\|^2} < \frac{\alpha}{\beta} = \underline{\rho}$$

Vigilance

RESET

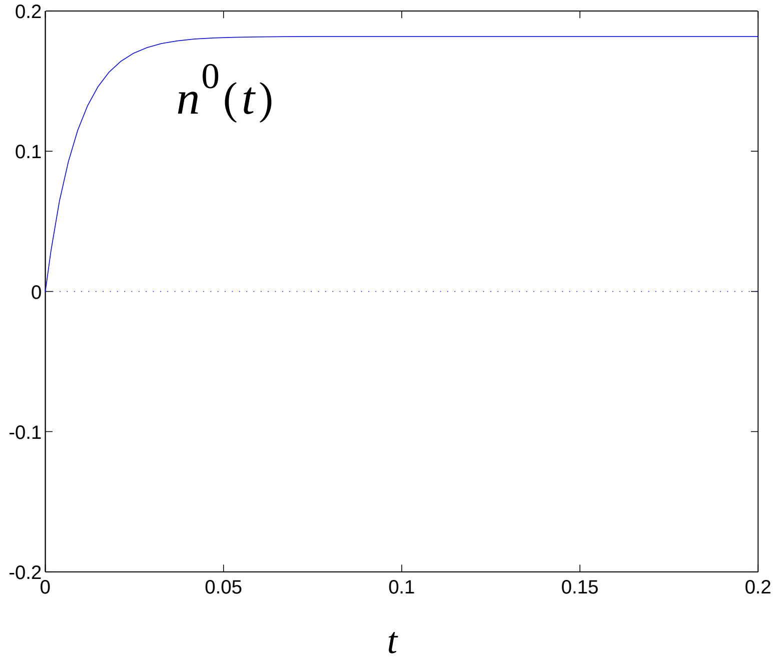
Since $\mathbf{a}^1 = \mathbf{p} \cap \mathbf{w}_j^{2:1}$, a reset will occur when there is enough of a mismatch between $\mathbf{p}$ and $\mathbf{w}_j^{2:1}$.

Orienting Subsystem Example

$\varepsilon = 0.1,\ \alpha = 3,\ \beta = 4\ (\rho = 0.75)$ $\quad \mathbf{p} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \mathbf{a}^1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$$(0.1)\frac{dn^0(t)}{dt} = -n^0(t) + (1 - n^0(t))\{3(p_1 + p_2)\} - (n^0(t) + 1)\{4(a_1^1 + a_2^1)\}$$

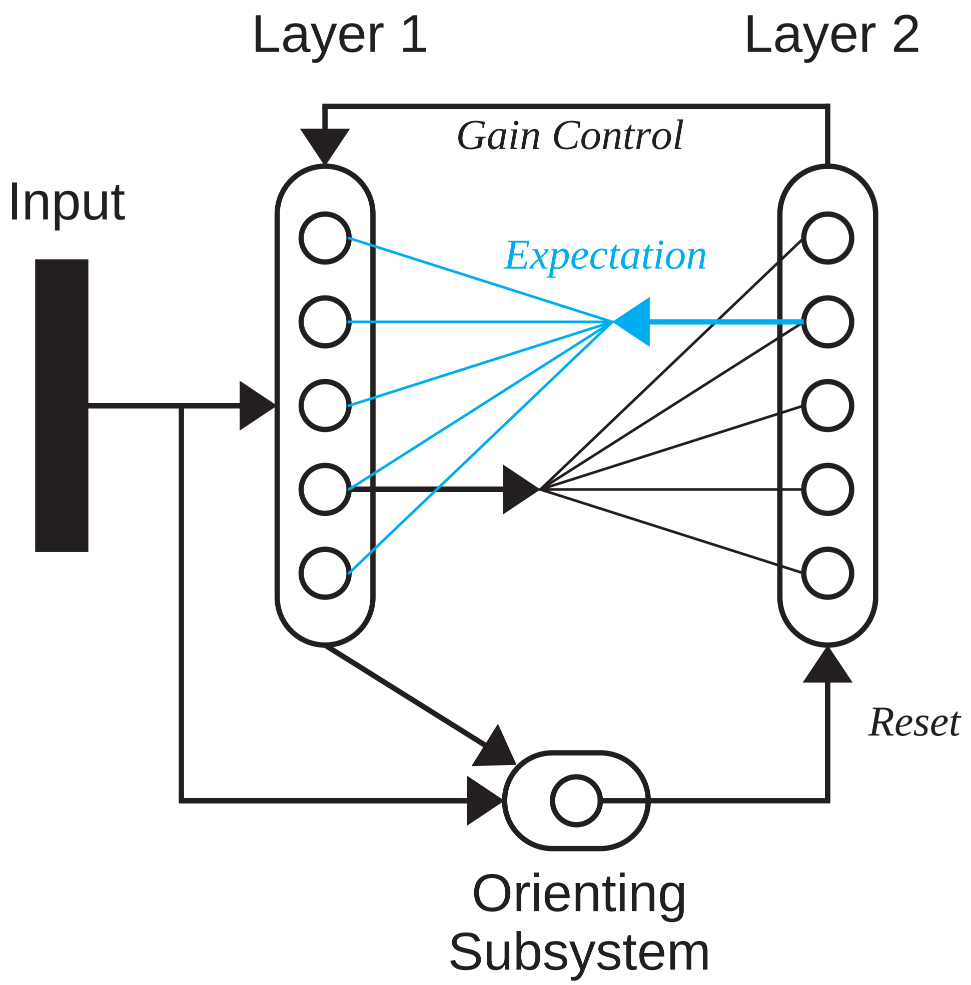
$$\frac{dn^0(t)}{dt} = -110n^0(t) + 20$$



Orienting Subsystem Summary

$$a^0 = \begin{cases} 1\ , & \text{if}\left[\left\|\mathbf{a}^1\right\|^2 / \left\|\mathbf{p}\right\|^2 < \rho\right] \\ 0\ , & \text{otherwise} \end{cases}$$

Learning Laws: L1-L2 and L2-L1



The ART1 network has two separate learning laws: one for the L1-L2 connections (instars) and one for the L2-L1 connections (outstars).

Both sets of connections are updated at the same time - when the input and the expectation have an adequate match.

The process of matching, and subsequent adaptation is referred to as resonance.

Subset/Superset Dilemma

Suppose that $\mathbf{W}^{1:2} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ so the prototypes are ${}_1\mathbf{w}^{1:2} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ ${}_2\mathbf{w}^{1:2} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

We say that ${}_1\mathbf{w}^{1:2}$ is a subset of ${}_2\mathbf{w}^{1:2}$, because ${}_2\mathbf{w}^{1:2}$ has a 1 wherever ${}_1\mathbf{w}^{1:2}$ has a 1.

If the output of layer 1 is $\mathbf{a}^1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ then the input to Layer 2 will be

$$\mathbf{W}^{1:2}\mathbf{a}^1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

Both prototype vectors have the same inner product with $\mathbf{a}^1$, even though the first prototype is identical to $\mathbf{a}^1$ and the second prototype is not. This is called the *Subset/Superset* dilemma.

Subset/Superset Solution

Normalize the prototype patterns.

$$\mathbf{W}^{1:2} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

$$\mathbf{W}^{1:2}\mathbf{a}^{1} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{2}{3} \end{bmatrix}$$

Now we have the desired result; the first prototype has the largest inner product with the input.

L1-L2 Learning Law

Instar Learning with Competition

$$\frac{d[{}_i\mathbf{w}^{1:2}(t)]}{dt} = a_i^2(t)[\{{}^{+}\mathbf{b} - {}_i\mathbf{w}^{1:2}(t)\}\zeta[{}^{+}\mathbf{W}]\mathbf{a}^1(t) - \{{}_i\mathbf{w}^{1:2}(t) + {}^{-}\mathbf{b}\}[{}^{-}\mathbf{W}]\mathbf{a}^1(t)] ,$$

where

$$\underbrace{{}^{+}\mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{\text{Upper Limit Bias}} \quad \underbrace{{}^{-}\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\text{Lower Limit Bias}} \quad \underbrace{{}^{+}\mathbf{W} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}}_{\text{On-Center Connections}} \quad \underbrace{{}^{-}\mathbf{W} = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 0 \end{bmatrix}}_{\text{Off-Surround Connections}}$$

When neuron *i* of Layer 2 is active, ${}_i\mathbf{w}^{1:2}$ is moved in the direction of $\mathbf{a}^1$. The elements of ${}_i\mathbf{w}^{1:2}$ compete, and therefore ${}_i\mathbf{w}^{1:2}$ is normalized.

Fast Learning

$$\frac{dw^{1:2}_{i,j}(t)}{dt} = a^2_i(t)\left[(1 - w^{1:2}_{i,j}(t))\zeta a^1_j(t) - w^{1:2}_{i,j}(t)\sum_{k \neq j} a^1_k(t)\right]$$

For *fast learning* we assume that the outputs of Layer 1 and Layer 2 remain constant until the weights reach steady state.

Assume that $a^2_i(t) = 1$, and solve for the steady state weight:

$$0 = \left[(1 - w^{1:2}_{i,j})\zeta a^1_j - w^{1:2}_{i,j}\sum_{k \neq j} a^1_k\right]$$

Case I: $a^1_j = 1$

$$\left.0 = (1 - w^{1:2}_{i,j})\zeta - w^{1:2}_{i,j}(\|\mathbf{a}^1\|^2 - 1) = -(\zeta + \|\mathbf{a}^1\|^2 - 1)w^{1:2}_{i,j} + \zeta \right\} \quad w^{1:2}_{i,j} = \frac{\zeta}{\zeta + \|\mathbf{a}^1\|^2 - 1}$$

Case II: $a^1_j = 0$

$$\left.0 = -w^{1:2}_{i,j}\|\mathbf{a}^1\|^2 \right\} \quad w^{1:2}_{i,j} = 0$$

Summary

$$\boxed{{}_i\mathbf{w}^{1:2} = \frac{\zeta \mathbf{a}^1}{\zeta + \|\mathbf{a}^1\|^2 - 1}}$$

Learning Law: L2-L1

Outstar

$$\frac{d[\mathbf{w}_j^{2:1}(t)]}{dt} = a_j^2(t)[-\mathbf{w}_j^{2:1}(t) + \mathbf{a}^1(t)]$$

Fast Learning

Assume that $a^2_j(t) = 1$, and solve for the steady state weight:

$$\mathbf{0} = -\mathbf{w}_j^{2:1} + \mathbf{a}^1 \quad \text{or} \quad \mathbf{w}_j^{2:1} = \mathbf{a}^1$$

Column j of $\mathbf{W}^{2:1}$ converges to the output of Layer 1, which is a combination of the input pattern and the previous prototype pattern. The prototype pattern is modified to incorporate the current input pattern.

ART1 Algorithm Summary

0) All elements of the initial $\mathbf{W}^{2:1}$ matrix are set to 1. All elements of the initial $\mathbf{W}^{1:2}$ matrix are set to $\zeta/(\zeta+S^1-1)$.

1) Input pattern is presented. Since Layer 2 is not active,

$$\mathbf{a}^1 = \mathbf{p}$$

2) The input to Layer 2 is computed, and the neuron with the largest input is activated.

$$a_i^2 = \begin{cases} 1 \,, & \text{if}(({}_i\mathbf{w}^{1:2})^T\mathbf{a}^1 = max[({}_k\mathbf{w}^{1:2})^T\mathbf{a}^1]) \\ 0 \,, & \text{otherwise} \end{cases}$$

In case of a tie, the neuron with the smallest index is the winner.

3) The L2-L1 expectation is computed.

$$\mathbf{W}^{2:1}\mathbf{a}^2 = \mathbf{w}_j^{2:1}$$

Summary Continued

4) Layer 1 output is adjusted to include the L2-L1 expectation.

$$\mathbf{a}^1 = \mathbf{p} \cap \mathbf{w}_j^{2:1}$$

5) The orienting subsystem determines match between the expectation and the input pattern.

$$a^0 = \begin{cases} 1\,, & \text{if}\left[\left\|\mathbf{a}^1\right\|^2 / \left\|\mathbf{p}\right\|^2 < \rho\right] \\ 0\,, & \text{otherwise} \end{cases}$$

6) If $a^0 = 1$, then set $a^2_j = 0$, inhibit it until resonance, and return to Step 1. If $a^0 = 0$, then continue with Step 7.

7) Resonance has occured. Update row j of $\mathbf{W}^{1:2}$.

$${}_j\mathbf{w}^{1:2} = \frac{\zeta \mathbf{a}^1}{\zeta + \left\|\mathbf{a}^1\right\|^2 - 1}$$

8) Update column j of $\mathbf{W}^{2:1}$.

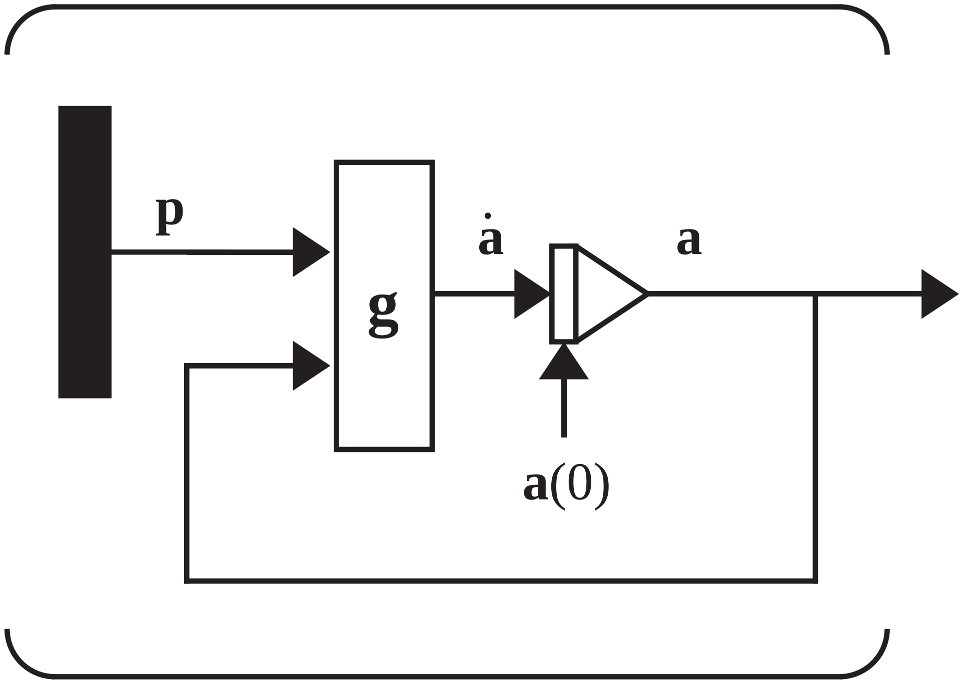
$$\mathbf{w}_j^{2:1} = \mathbf{a}^1$$

9) Remove input, restore inhibited neurons, and return to Step 1.

Stability

Recurrent Networks

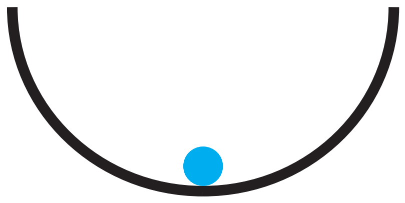
Nonlinear Recurrent Network



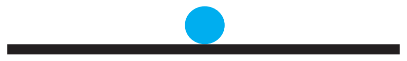
$$d\mathbf{a}(t)/dt = \mathbf{g}(\mathbf{a}(t), \mathbf{p}(t), t)$$

Types of Stability

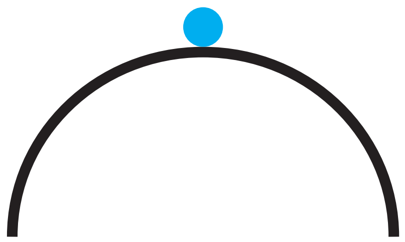
A ball bearing, with dissipative friction, in a gravity field:



Asymptotically Stable

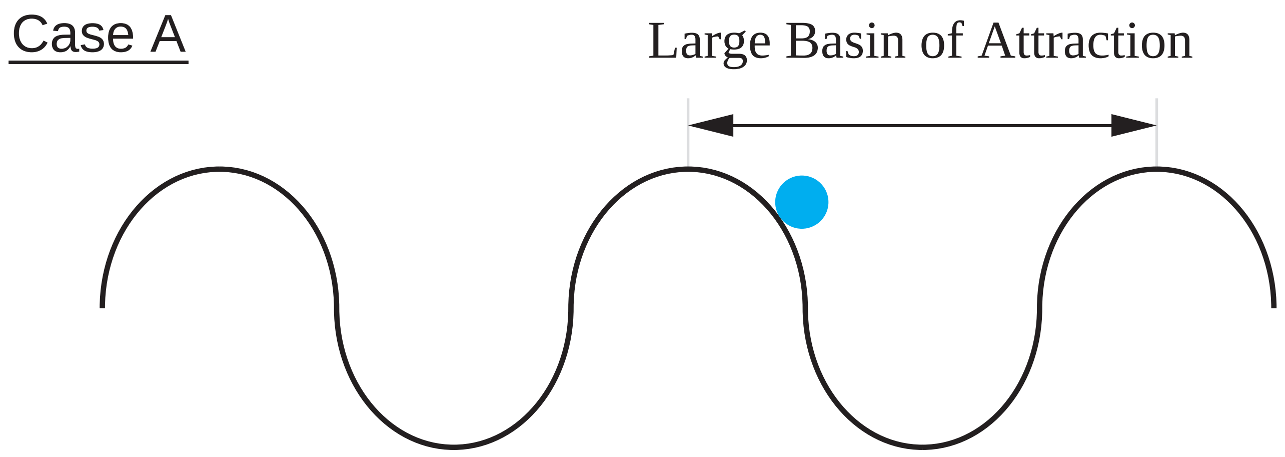


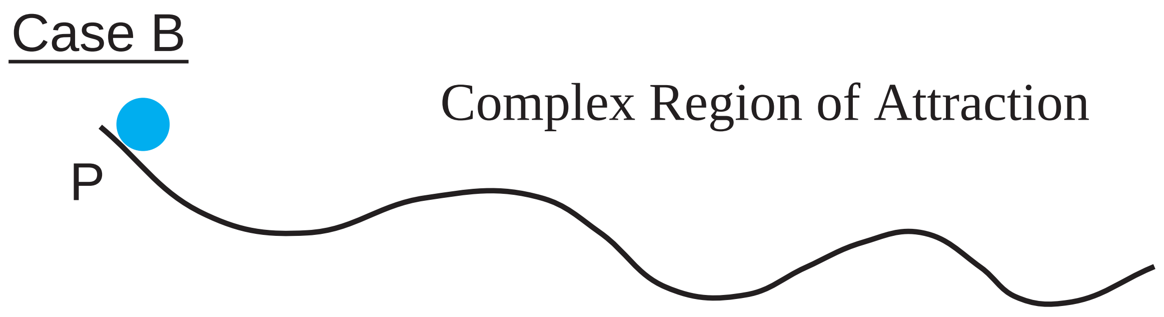
Stable in the Sense of Lyapunov



Unstable

Basins of Attraction





In the Hopfield network we want the prototype patterns to be stable points with large basins of attraction.

Lyapunov Stability

$$\frac{d}{dt}\mathbf{a}(t) = \mathbf{g}(\mathbf{a}(t),\mathbf{p}(t), t)$$

Eqilibrium Point:

An equilibrium point is a point $\mathbf{a}^*$ where $d\mathbf{a}/dt = \mathbf{0}$.

Stability (in the sense of Lyapunov):

The origin is a stable equilibrium point if for any given value $\varepsilon > 0$ there exists a number $\delta(\varepsilon) > 0$ such that if $\|\mathbf{a}(0)\| < \delta$, then the resulting motion, $\mathbf{a}(t)$, satisfies $\|\mathbf{a}(t)\| < \varepsilon$ for $t > 0$.

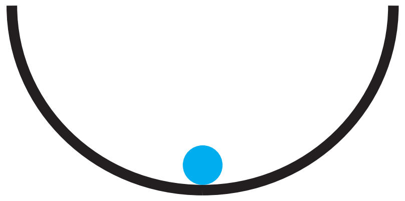
17

Asymptotic Stability

$$\frac{d}{dt}\mathbf{a}(t) = \mathbf{g}(\mathbf{a}(t), \mathbf{p}(t), t)$$

Asymptotic Stability:

The origin is an asymptotically stable equilibrium point if there exists a number $\delta > 0$ such that if $\|\mathbf{a}(0)\| < \delta$, then the resulting motion, $\mathbf{a}(t)$, satisfies $\|\mathbf{a}(t)\| \to 0$ as $t \to \infty$.



Definite Functions

Positive Definite:

A scalar function $V(\mathbf{a})$ is positive definite if $V(\mathbf{0}) = 0$ and $V(\mathbf{a}) > 0$ for $\mathbf{a} \neq 0$.

Positive Semidefinite:

A scalar function $V(\mathbf{a})$ is positive semidefinite if $V(\mathbf{0}) = 0$ and $V(\mathbf{a}) \geq 0$ for all $\mathbf{a}$.

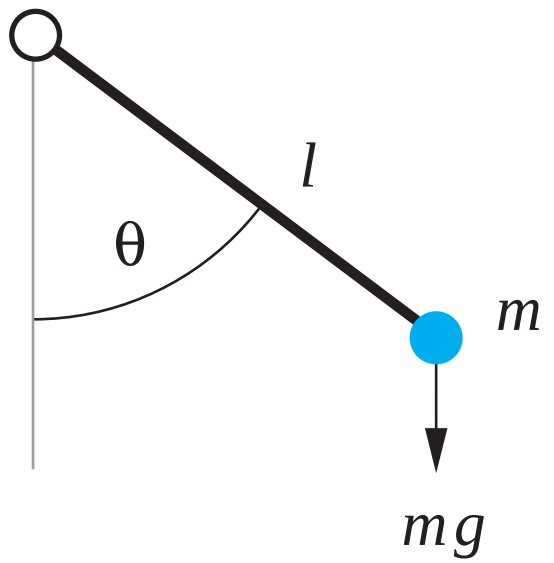
Lyapunov Stability Theorem

$$\frac{d\mathbf{a}}{dt} = \mathbf{g}(\mathbf{a})$$

Theorem 1: Lyapunov Stability Theorem

If a positive definite function $V(\mathbf{a})$ can be found such that $dV(\mathbf{a})/dt$ is negative semidefinite, then the origin $(\mathbf{a} = \mathbf{0})$ is stable for the above system. If a positive definite function $V(\mathbf{a})$ can be found such that $dV(\mathbf{a})/dt$ is negative definite, then the origin $(\mathbf{a} = \mathbf{0})$ is asymptotically stable. In each case, $V(\mathbf{a})$ is called a Lyapunov function of the system.

Pendulum Example



$$ml\frac{d^2\theta}{dt^2} + c\frac{d\theta}{dt} + mg\sin(\theta) = 0$$

State Variable Model

$$a_1 = \theta \qquad \frac{da_1}{dt} = a_2$$

$$a_2 = \frac{d\theta}{dt} \qquad \frac{da_2}{dt} = -\frac{g}{l}\sin(a_1) - \frac{c}{ml}a_2$$

Equilibrium Point

Check: $\mathbf{a} = \mathbf{0}$

$$\frac{da_1}{dt} = a_2 = 0$$

$$\frac{da_2}{dt} = -\frac{g}{l}\sin(a_1) - \frac{c}{ml}a_2 = -\frac{g}{l}\sin(0) - \frac{c}{ml}(0) = 0$$

Therefore the origin is an equilibrium point.

17

Lyapunov Function (Energy)

$$V(\mathbf{a}) = \underbrace{\frac{1}{2}ml^2(a_2)^2}_{\text{Kinetic Energy}} + \underbrace{mgl(1 - \cos(a_1))}_{\text{Potential Energy}}$$ (Positive Definite)

Check the derivative of the Lyapunov function:

$$\frac{d}{dt}V(\mathbf{a}) = [\nabla V(\mathbf{a})]^T g(\mathbf{a}) = \frac{\partial V}{\partial a_1}\left(\frac{da_1}{dt}\right) + \frac{\partial V}{\partial a_2}\left(\frac{da_2}{dt}\right)$$

$$\frac{d}{dt}V(\mathbf{a}) = (mgl\sin(a_1))a_2 + (ml^2 a_2)\left(-\frac{g}{l}\sin(a_1) - \frac{c}{ml}a_2\right)$$

$$\frac{d}{dt}V(\mathbf{a}) = -cl(a_2)^2 \le 0$$

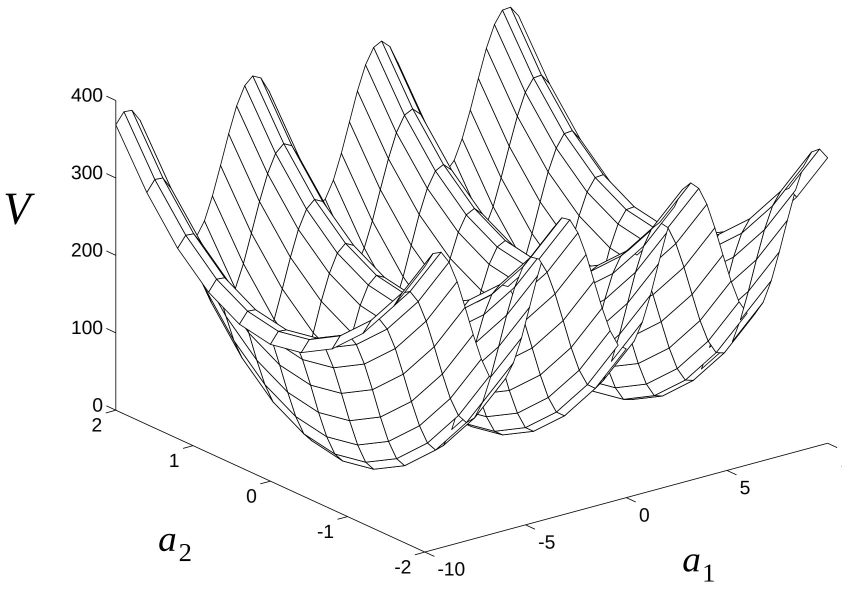
The derivative is negative semidefinite, which proves that the origin is stable in the sense of Lyapunov (at least).

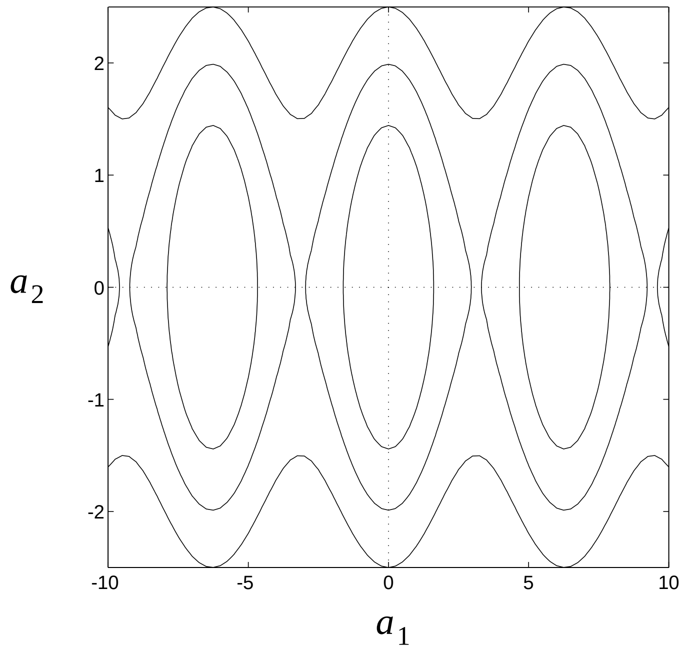
Numerical Example

$$g = 9.8, \quad m = 1, \quad l = 9.8, \quad c = 1.96$$

$$\frac{da_1}{dt} = a_2 \qquad \frac{da_2}{dt} = -\sin(a_1) - 0.2a_2$$

$$V = (9.8)^2\left[\frac{1}{2}(a_2)^2 + (1 - \cos(a_1))\right] \qquad \frac{dV}{dt} = -(19.208)(a_2)^2$$



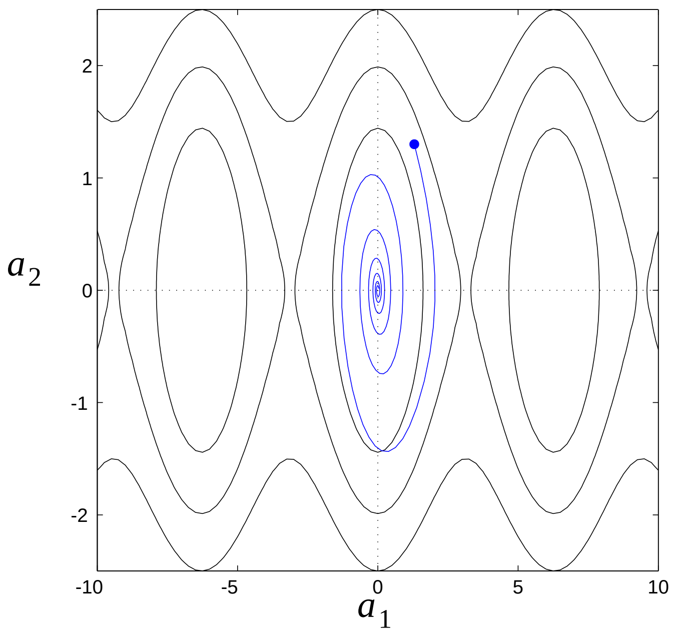


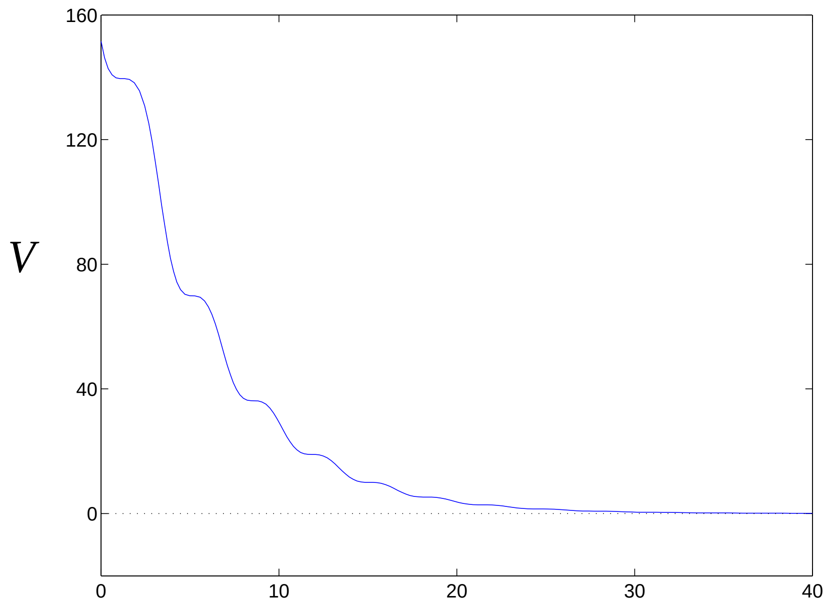
Pendulum Response

a_1

$$\mathbf{a}(0) = \begin{bmatrix} 1.3 \\ 1.3 \end{bmatrix}$$

a_2





17 Definitions (Lasalle's Theorem)

Lyapunov Function

Let $V(\mathbf{a})$ be a continuously differentiable function from $\Re^n$ to $\Re$. If G is any subset of $\Re^n$, we say that V is a Lyapunov function on G for the system $d\mathbf{a}/dt = \mathbf{g}(\mathbf{a})$ if

$$\frac{dV(\mathbf{a})}{dt} = (\nabla V(\mathbf{a}))^T \mathbf{g}(\mathbf{a})$$

does not change sign on G.

Set Z

$$Z = \{\mathbf{a}: dV(\mathbf{a})/dt = 0, \ \mathbf{a} \text{ in the closure of } G\}$$

Definitions

Invariant Set

A set of points in $\Re^n$ is invariant with respect to $d\mathbf{a}/dt = \mathbf{g}(\mathbf{a})$ if every solution of $d\mathbf{a}/dt = \mathbf{g}(\mathbf{a})$ starting in that set remains in the set for all time.

Set L

L is defined as the largest invariant set in Z.

Lasalle’s Invariance Theorem

Theorem 2: Lasalle’s Invariance Theorem

If V is a Lyapunov function on G for $d\mathbf{a}/dt = \mathbf{g}(\mathbf{a})$, then each solution $\mathbf{a}(t)$ that remains in G for all $t > 0$ approaches $L^{\circ} = L \cup \{\infty\}$ as $t \to \infty$. (G is a basin of attraction for L, which has all of the stable points.) If all trajectories are bounded, then $\mathbf{a}(t) \to L$ as $t \to \infty$.

Corollary 1: Lasalle’s Corollary

Let G be a component (one connected subset) of

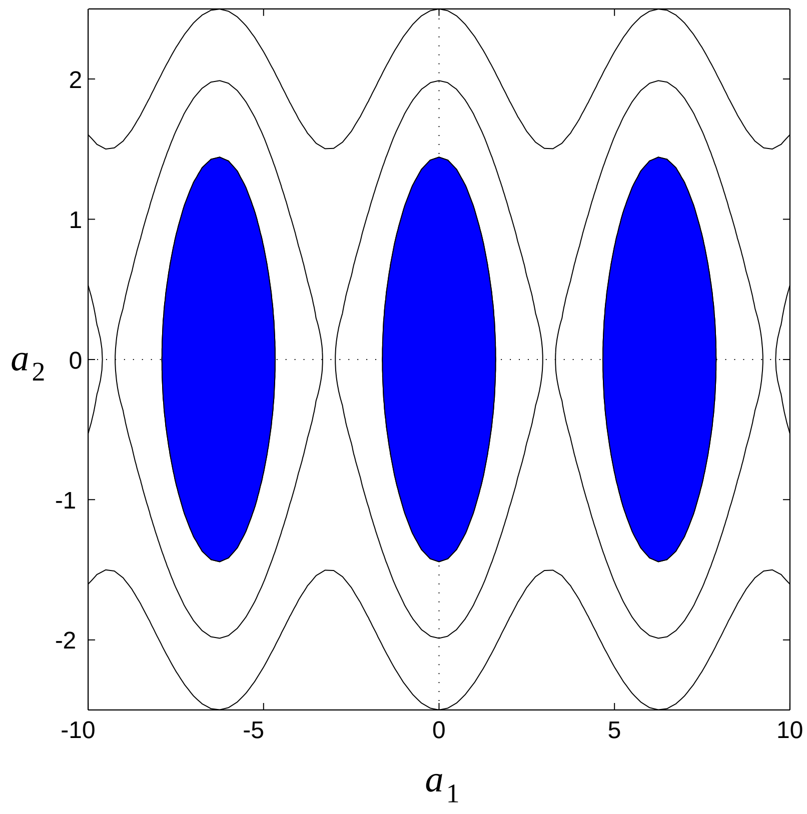
$$\Omega_{\eta} = \{\mathbf{a}: V(\mathbf{a}) < \eta\}.$$

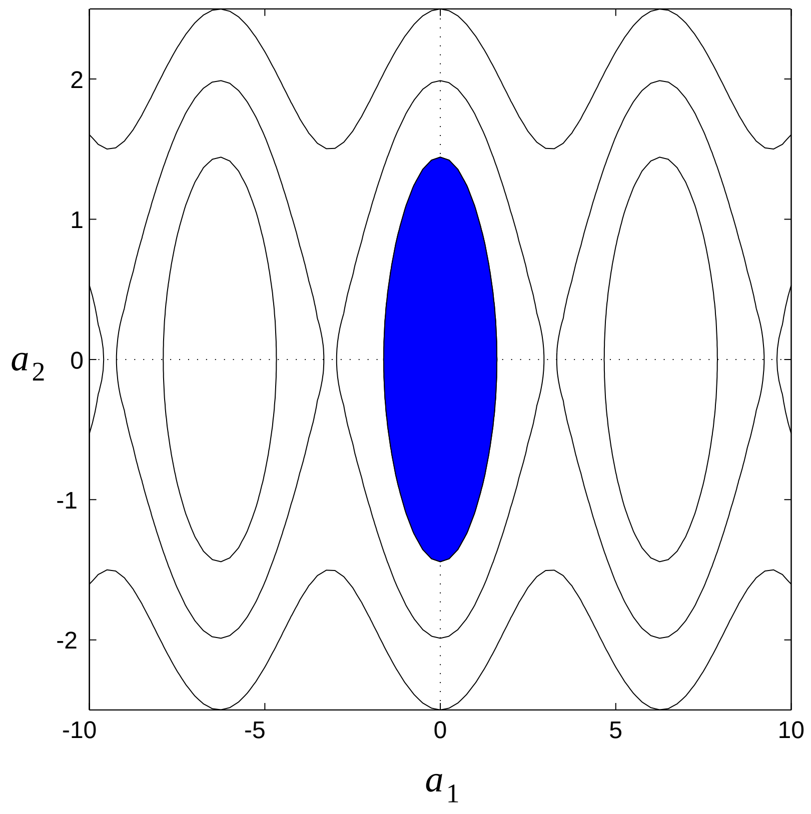
Assume that G is bounded, $dV(\mathbf{a})/dt \leq 0$ on the set G, and let the set $L^{\circ} = \text{closure}(L \cup G)$ be a subset of G. Then L° is an attractor, and G is in its region of attraction.

Pendulum Example

$\Omega_{100} = \{\mathbf{a}: V(\mathbf{a}) \le 100\}$

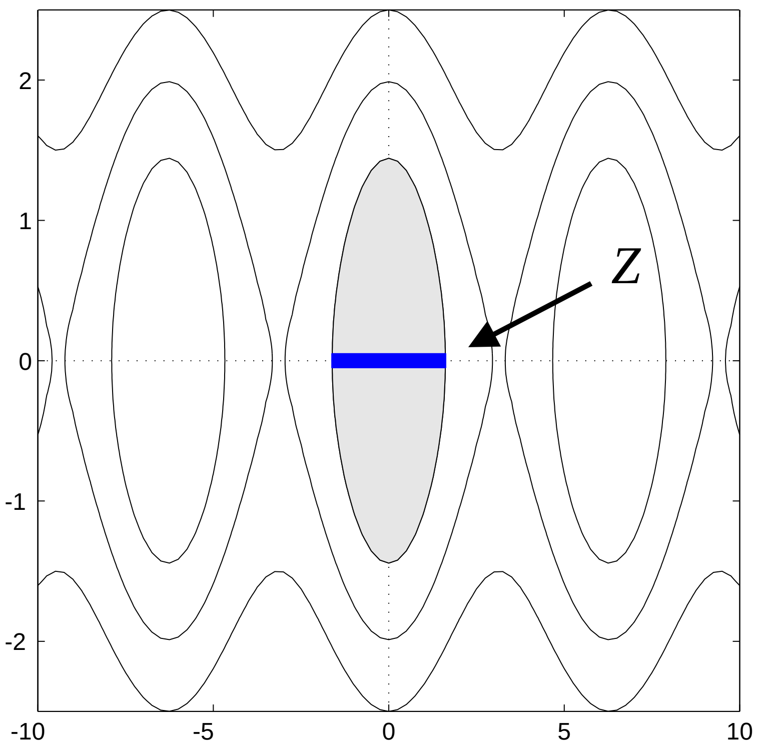
G = One component of Ω_{100}.





Invariant and Attractor Sets

$Z = \{\mathbf{a}: dV(\mathbf{a})/dt = 0,\ \mathbf{a} \text{ in the closure of } G\} = \{\mathbf{a}: a_2 = 0, \mathbf{a} \text{ in the closure of } G\}$



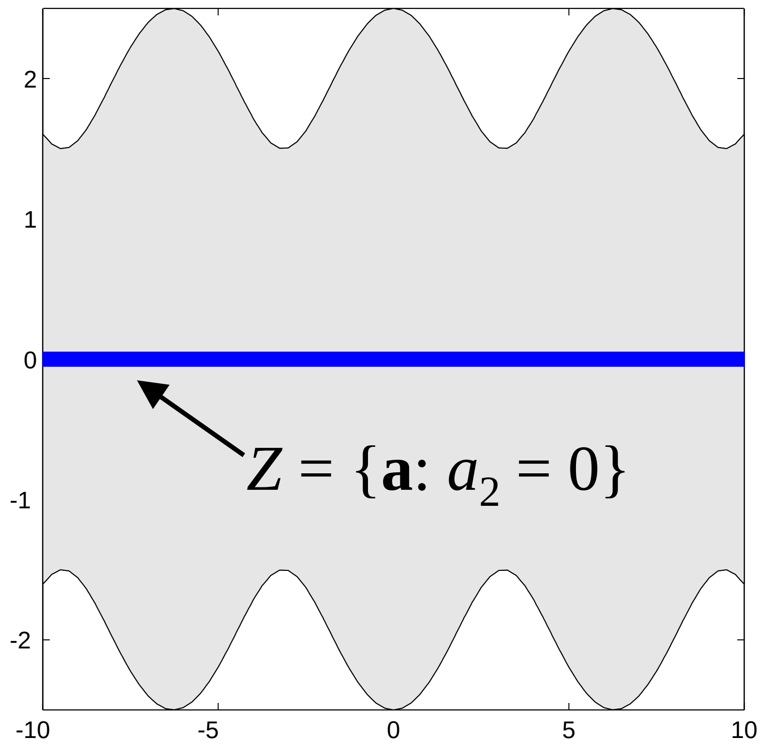
$L = \{\mathbf{a}: \mathbf{a} = 0\}$

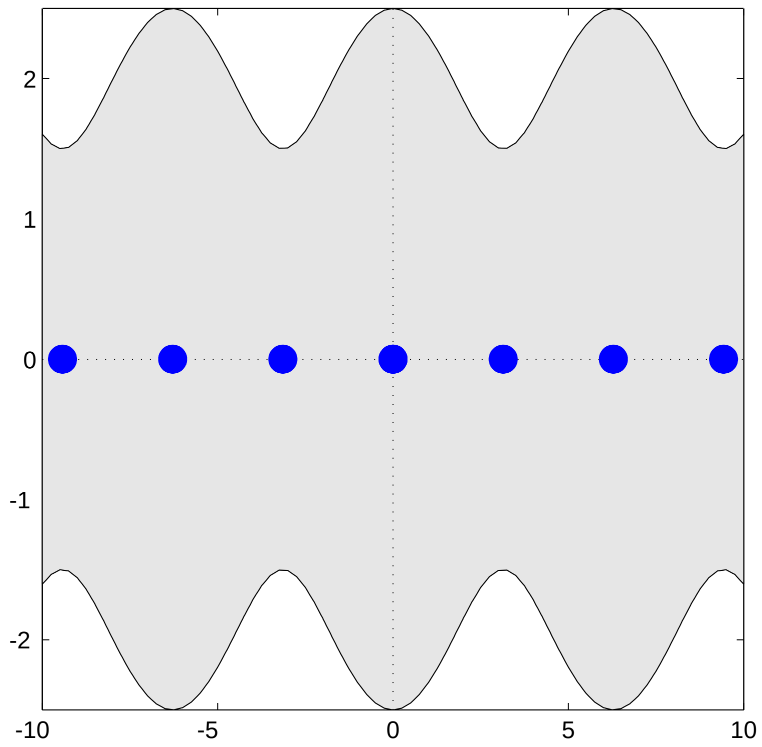
17

Larger G Set

$G = \Omega_{300} = \{\mathbf{a}: V(\mathbf{a}) \le 300\}$

$L^{\circ} = L = \{\mathbf{a}: a_1 = \pm n\pi, a_2 = 0\}$

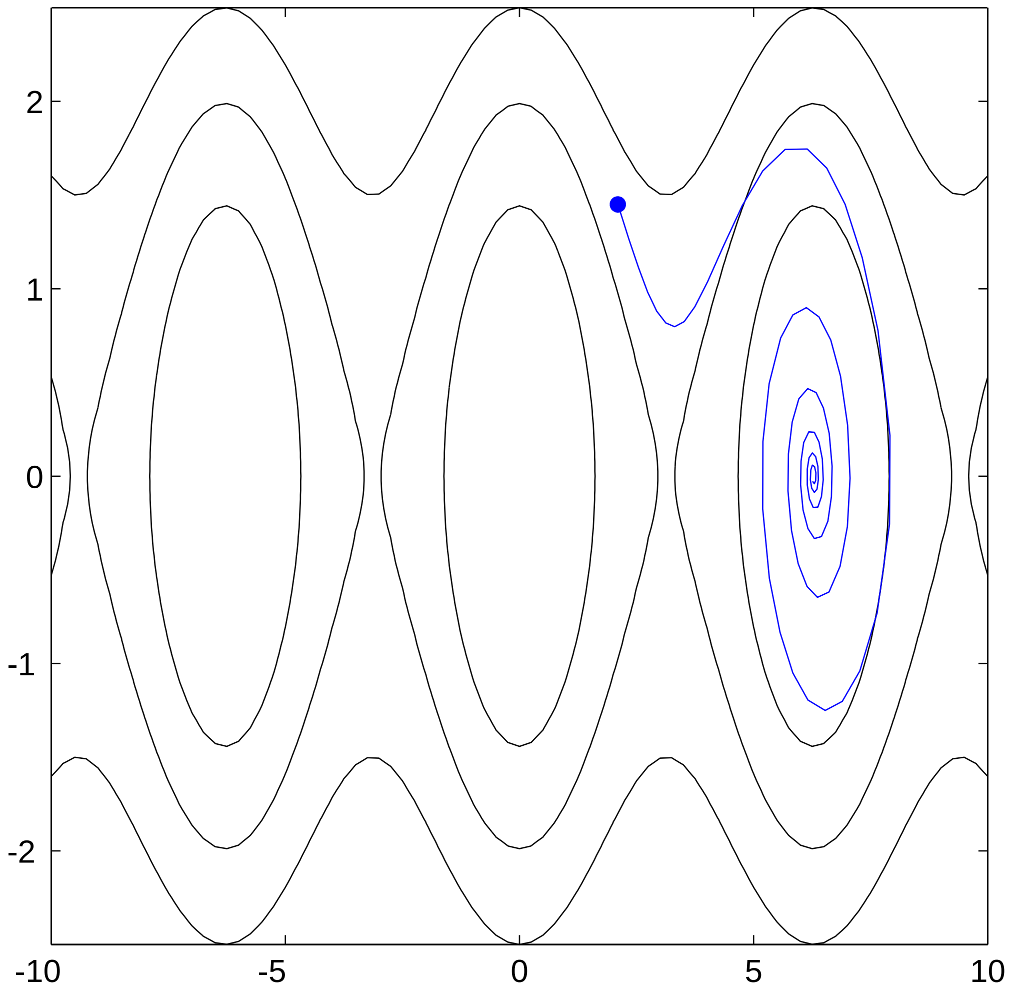




For this choice of G we can say little about where the trajectorywill converge.

17

Pendulum Trajectory



17

Comments

We want G to be as large as possible, because that will indicate the region of attraction. However, we want to choose V so that the set Z, which will contain the attractor set, is as small as possible.

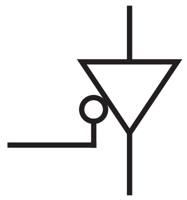
$V = 0$ is a Lyapunov function for all of $\Re^n$, but it gives no information since $Z = \Re^n$.

If V_1 and V_2 are Lyapunov functions on G, and dV_1/dt and dV_2/dt have the same sign, then $V_1 + V_2$ is also a Lyapunov function, and $Z = Z_1 \cap Z_2$. If Z is smaller than Z_1 or Z_2, then V is a “better” Lyapunov function than either V_1 or V_2. V is always at least as good as either V_1 or V_2.

Hopfield Network

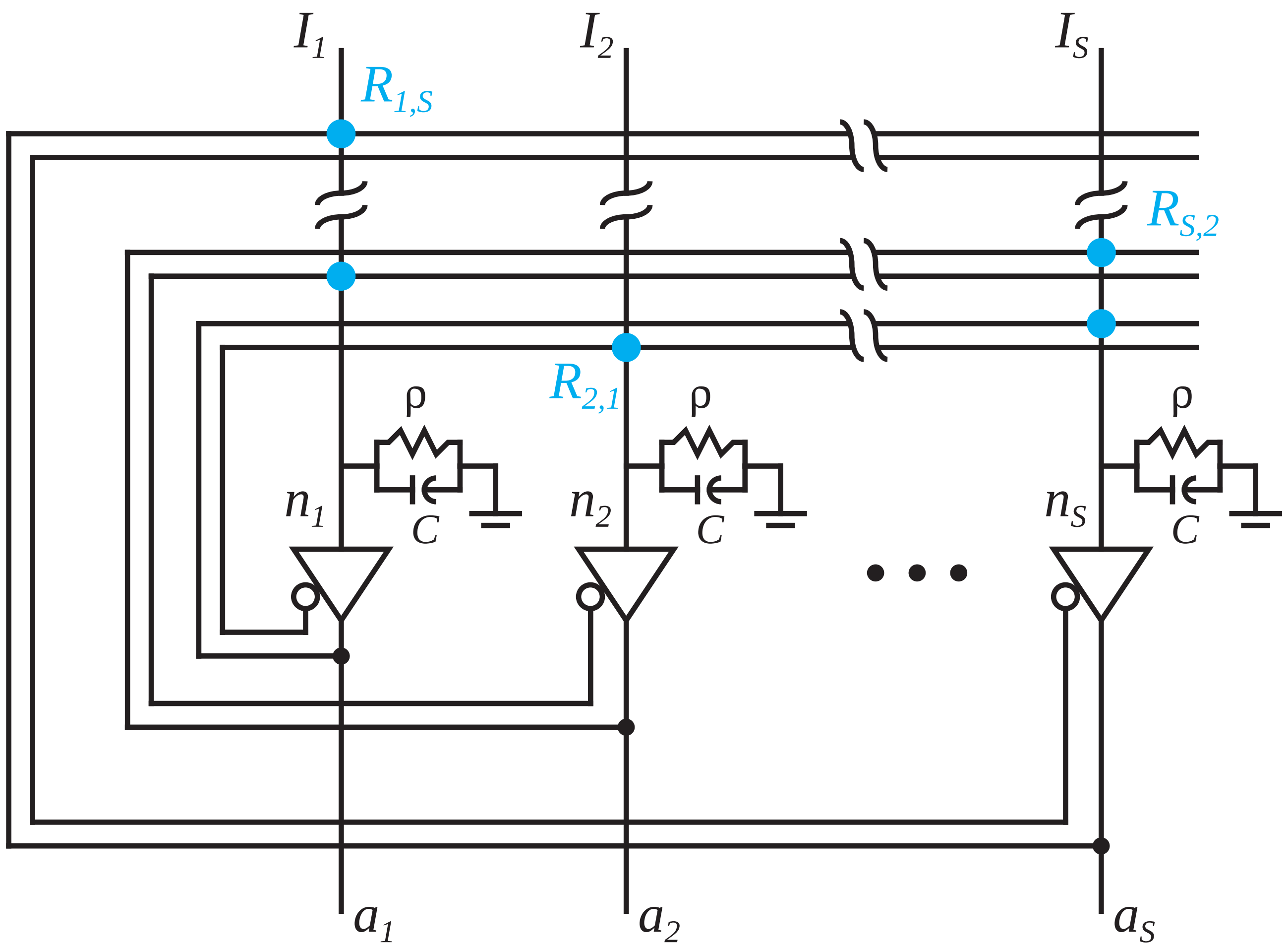
Hopfield Model

Amplifier



Resistor

●



Equations of Operation

$$C\frac{dn_i(t)}{dt} = \sum_{j=1}^{S} T_{i,j}a_j(t) - \frac{n_i(t)}{R_i} + I_i$$

n_i - input voltage to the *i*th amplifier
a_i - output voltage of the *i*th amplifier
C - amplifier input capacitance
I_i - fixed input current to the *i*th amplifier

$$\left|T_{i,j}\right| = \frac{1}{R_{i,j}} \qquad \frac{1}{R_i} = \frac{1}{\rho} + \sum_{j=1}^{S}\frac{1}{R_{i,j}} \qquad n_i = f^{-1}(a_i) \qquad a_i = f(n_i)$$

Network Format

$$R_i C \frac{dn_i(t)}{dt} = \sum_{j=1}^{S} R_i T_{i,j} a_j(t) - n_i(t) + R_i I_i$$

Define:

$$\varepsilon = R_i C \qquad w_{i,j} = R_i T_{i,j} \qquad b_i = R_i I_i$$

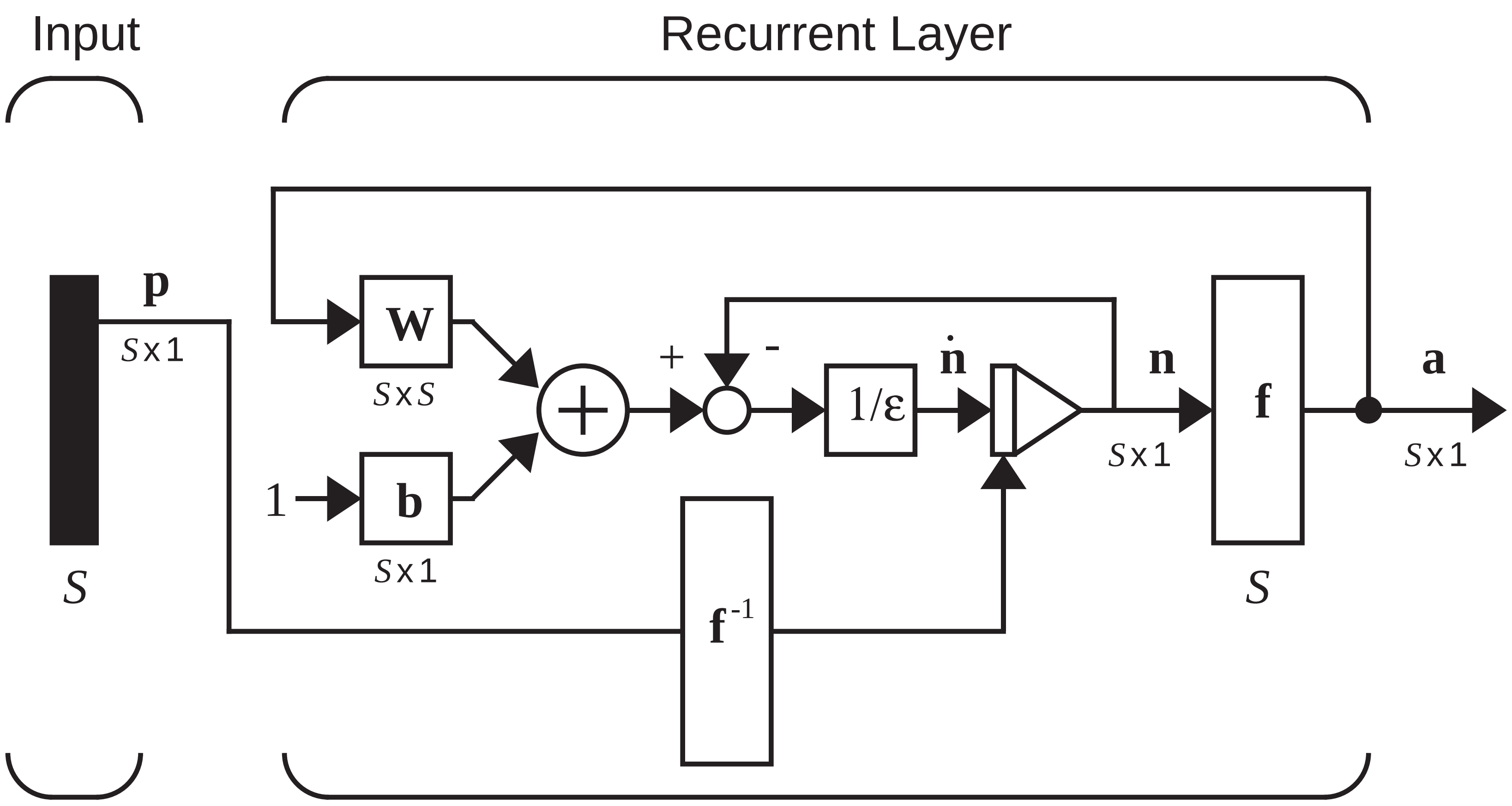
$$\varepsilon \frac{dn_i(t)}{dt} = -n_i(t) + \sum_{j=1}^{S} w_{i,j} a_j(t) + b_i$$

Vector Form:

$$\varepsilon \frac{d\mathbf{n}(t)}{dt} = -\mathbf{n}(t) + \mathbf{W}\mathbf{a}(t) + \mathbf{b}$$

$$\mathbf{a}(t) = \mathbf{f}(\mathbf{n}(t))$$

Hopfield Network



$$\mathbf{n}(0) = \mathbf{f}^{-1}(\mathbf{p}),\ \ (\mathbf{a}(0) = \mathbf{p}) \qquad \varepsilon\, d\mathbf{n}/dt = -\mathbf{n} + \mathbf{W}\mathbf{f}(\mathbf{n}) + \mathbf{b}$$

Lyapunov Function

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} + \sum_{i=1}^{S}\left\{\int_0^{a_i} f^{-1}(u)\,du\right\} - \mathbf{b}^T\mathbf{a}$$

Individual Derivatives

First Term:

$$\frac{d}{dt}\left\{-\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a}\right\} = -\frac{1}{2}\nabla[\mathbf{a}^T\mathbf{W}\mathbf{a}]^T\frac{d\mathbf{a}}{dt} = -[\mathbf{W}\mathbf{a}]^T\frac{d\mathbf{a}}{dt} = -\mathbf{a}^T\mathbf{W}\frac{d\mathbf{a}}{dt}$$

Second Term:

$$\frac{d}{dt}\left\{\int_0^{a_i} f^{-1}(u)du\right\} = \frac{d}{da_i}\left\{\int_0^{a_i} f^{-1}(u)du\right\}\frac{da_i}{dt} = f^{-1}(a_i)\frac{da_i}{dt} = n_i\frac{da_i}{dt}$$

$$\frac{d}{dt}\left[\sum_{i=1}^{S}\left\{\int_0^{a_i} f^{-1}(u)du\right\}\right] = \mathbf{n}^T\frac{d\mathbf{a}}{dt}$$

Third Term:

$$\frac{d}{dt}\{-\mathbf{b}^T\mathbf{a}\} = -\nabla[\mathbf{b}^T\mathbf{a}]^T\frac{d\mathbf{a}}{dt} = -\mathbf{b}^T\frac{d\mathbf{a}}{dt}$$

Complete Lyapunov Derivative

$$\frac{d}{dt}V(\mathbf{a}) = -\mathbf{a}^T\mathbf{W}\frac{d\mathbf{a}}{dt} + \mathbf{n}^T\frac{d\mathbf{a}}{dt} - \mathbf{b}^T\frac{d\mathbf{a}}{dt} = [-\mathbf{a}^T\mathbf{W} + \mathbf{n}^T - \mathbf{b}^T]\frac{d\mathbf{a}}{dt}$$

From the system equations we know:

$$[-\mathbf{a}^T\mathbf{W} + \mathbf{n}^T - \mathbf{b}^T] = -\varepsilon\left[\frac{d\mathbf{n}(t)}{dt}\right]^T$$

So the derivative can be written:

$$\frac{d}{dt}V(\mathbf{a}) = -\varepsilon\left[\frac{d\mathbf{n}(t)}{dt}\right]^T\frac{d\mathbf{a}}{dt} = -\varepsilon\sum_{i=1}^{S}\left(\frac{dn_i}{dt}\right)\left(\frac{da_i}{dt}\right) = -\varepsilon\sum_{i=1}^{S}\left(\frac{dn_i}{dt}\right)\left(\frac{da_i}{dt}\right)$$

$$= -\varepsilon\sum_{i=1}^{S}\left(\frac{d}{da_i}[f^{-1}(a_i)]\right)\left(\frac{da_i}{dt}\right)^2$$

If $\frac{d}{da_i}[f^{-1}(a_i)] > 0$ then $\frac{d}{dt}V(\mathbf{a}) \le 0$

Invariant Sets

$$Z = \{\mathbf{a}:\ dV(\mathbf{a})/dt = 0,\ \ \mathbf{a} \text{ in the closure of } G\}$$

$$\frac{d}{dt}V(\mathbf{a}) = -\varepsilon \sum_{i=1}^{S} \left(\frac{d}{da_i}[f^{-1}(a_i)]\right)\left(\frac{da_i}{dt}\right)^2$$

This will be zero only if the neuron outputs are not changing:

$$\frac{d\mathbf{a}}{dt} = \mathbf{0}$$

Therefore, the system energy is not changing only at the equilibrium points of the circuit. Thus, all points in *Z* are potential attractors:

$$L = Z$$

Example

$$a = f(n) = \frac{2}{\pi}\tan^{-1}\left(\frac{\gamma\pi n}{2}\right) \qquad n = \frac{2}{\gamma\pi}\tan\left(\frac{\pi}{2}a\right)$$

$$\left.\begin{array}{l} R_{1,2} = R_{2,1} = 1 \\ \\ T_{1,2} = T_{2,1} = 1 \end{array}\right\} \quad \mathbf{W} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\varepsilon = R_i C = 1$$

$$\gamma = 1.4$$

$$\left.I_1 = I_2 = 0\right\} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

18

Example Lyapunov Function

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} + \sum_{i=1}^{S}\left\{\int_0^{a_i} f^{-1}(u)\,du\right\} - \mathbf{b}^T\mathbf{a}$$

$$-\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} = -\frac{1}{2}\begin{bmatrix} a_1 & a_2 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = -a_1 a_2$$

$$\int_0^{a_i} f^{-1}(u)\,du = \frac{2}{\gamma\pi}\int_0^{a_i}\tan\left(\frac{\pi}{2}u\right)du = \frac{2}{\gamma\pi}\left[-\log\left[\cos\left(\frac{\pi}{2}u\right)\right]\frac{2}{\pi}\right]_0^{a_i} = -\frac{4}{\gamma\pi^2}\log\left[\cos\left(\frac{\pi}{2}a_i\right)\right]$$

$$V(\mathbf{a}) = -a_1 a_2 - \frac{4}{1.4\pi^2}\left[\log\left\{\cos\left(\frac{\pi}{2}a_1\right)\right\} + \log\left\{\cos\left(\frac{\pi}{2}a_2\right)\right\}\right]$$

Example Network Equations

$$\frac{d\mathbf{n}}{dt} = -\mathbf{n} + \mathbf{W}\mathbf{f}(\mathbf{n}) = -\mathbf{n} + \mathbf{W}\mathbf{a}$$

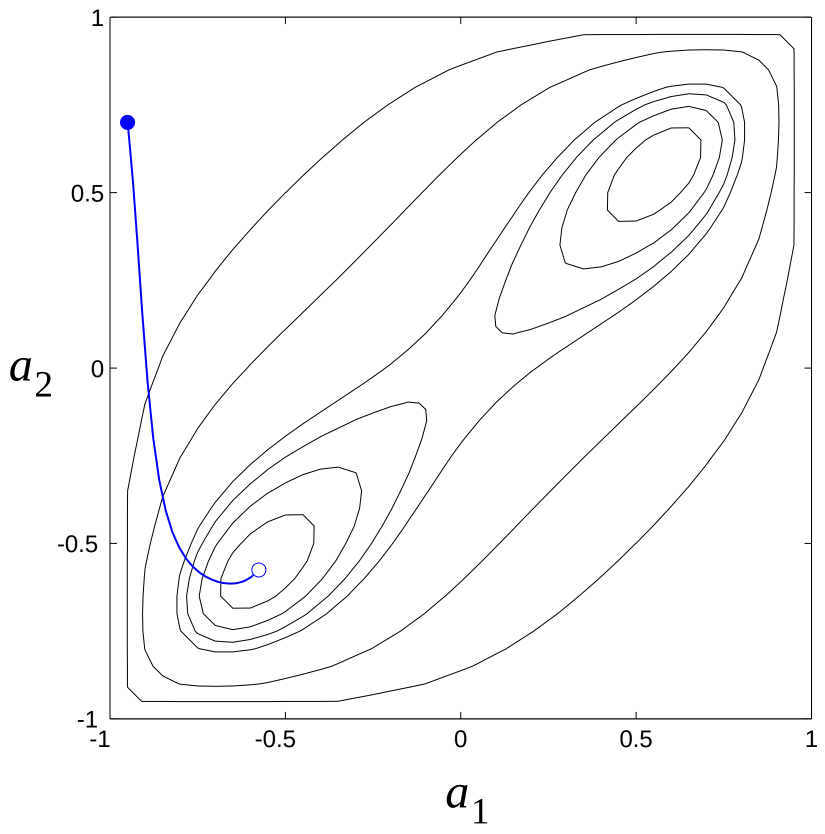
$$dn_1/dt = a_2 - n_1$$

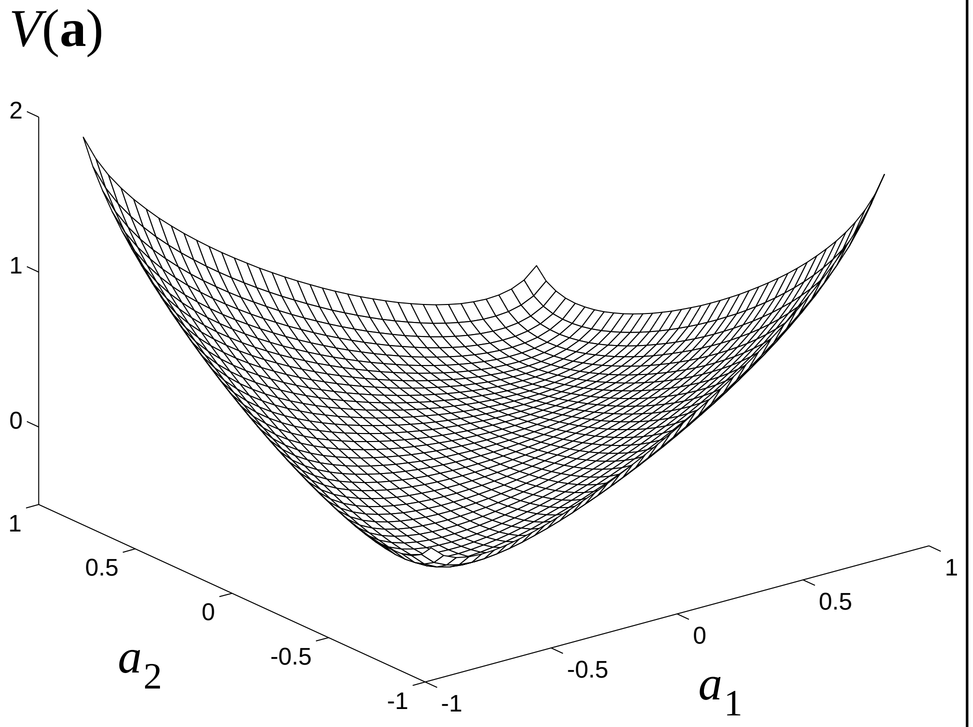
$$dn_2/dt = a_1 - n_2$$

$$a_1 = \frac{2}{\pi}\tan^{-1}\left(\frac{1.4\pi}{2}n_1\right)$$

$$a_2 = \frac{2}{\pi}\tan^{-1}\left(\frac{1.4\pi}{2}n_2\right)$$

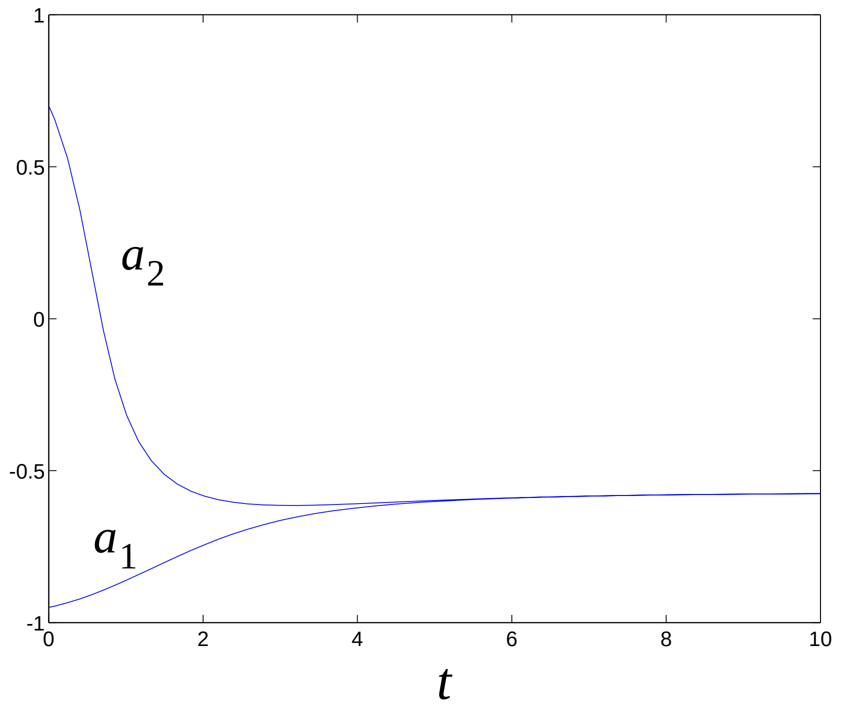
Lyapunov Function and Trajectory

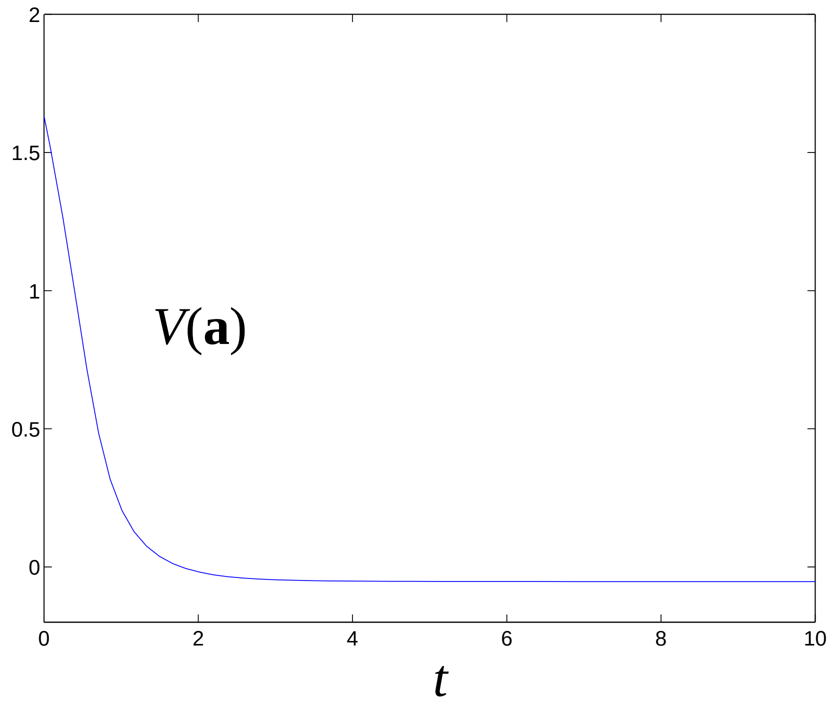




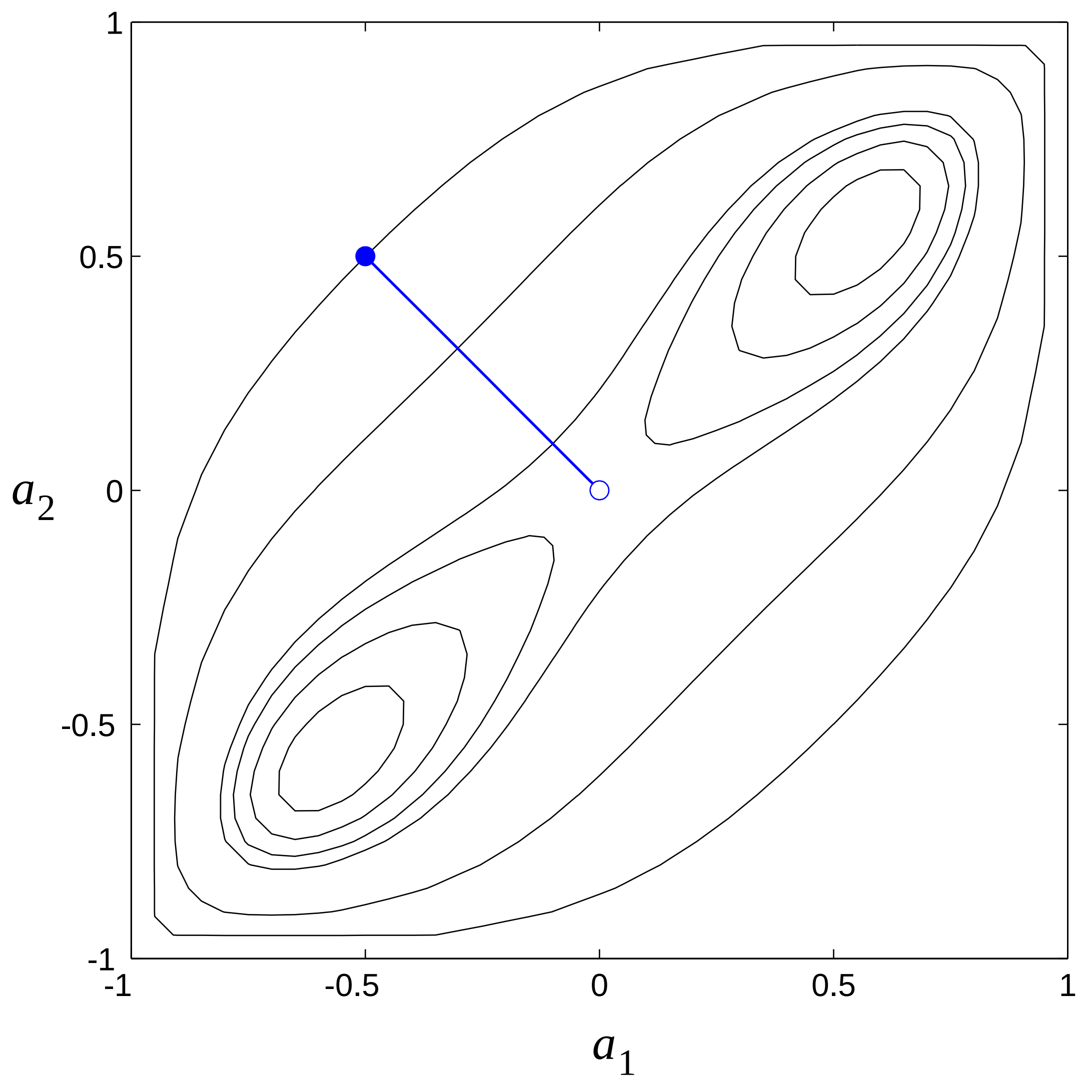
18

Time Response





Convergence to a Saddle Point



Hopfield Attractors

The potential attractors of the Hopfield network satisfy:

$$\frac{d\mathbf{a}}{dt} = \mathbf{0}$$

How are these points related to the minima of $V(\mathbf{a})$? The minima must satisfy:

$$\nabla V = \left[\frac{\partial V}{\partial a_1}\ \frac{\partial V}{\partial a_2}\ \dots\ \frac{\partial V}{\partial a_S}\right]^T = \mathbf{0}$$

Where the Lyapunov function is given by:

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} + \sum_{i=1}^{S}\left\{\int_0^{a_i} f^{-1}(u)\,du\right\} - \mathbf{b}^T\mathbf{a}$$

Hopfield Attractors

Using previous results, we can show that:

$$\nabla V(\mathbf{a}) = [-\mathbf{W}\mathbf{a} + \mathbf{n} - \mathbf{b}] = -\varepsilon\left[\frac{d\mathbf{n}(t)}{dt}\right]$$

The *i*th element of the gradient is therefore:

$$\frac{\partial}{\partial \mathrm{a}_\mathrm{i}} V(\mathbf{a}) = -\varepsilon \frac{dn_i}{dt} = -\varepsilon \frac{d}{dt}([f^{-1}(a_i)]) = -\varepsilon \frac{d}{d\,a_i}[f^{-1}(a_i)]\frac{da_i}{d\,t}$$

Since the transfer function and its inverse are monotonic increasing:

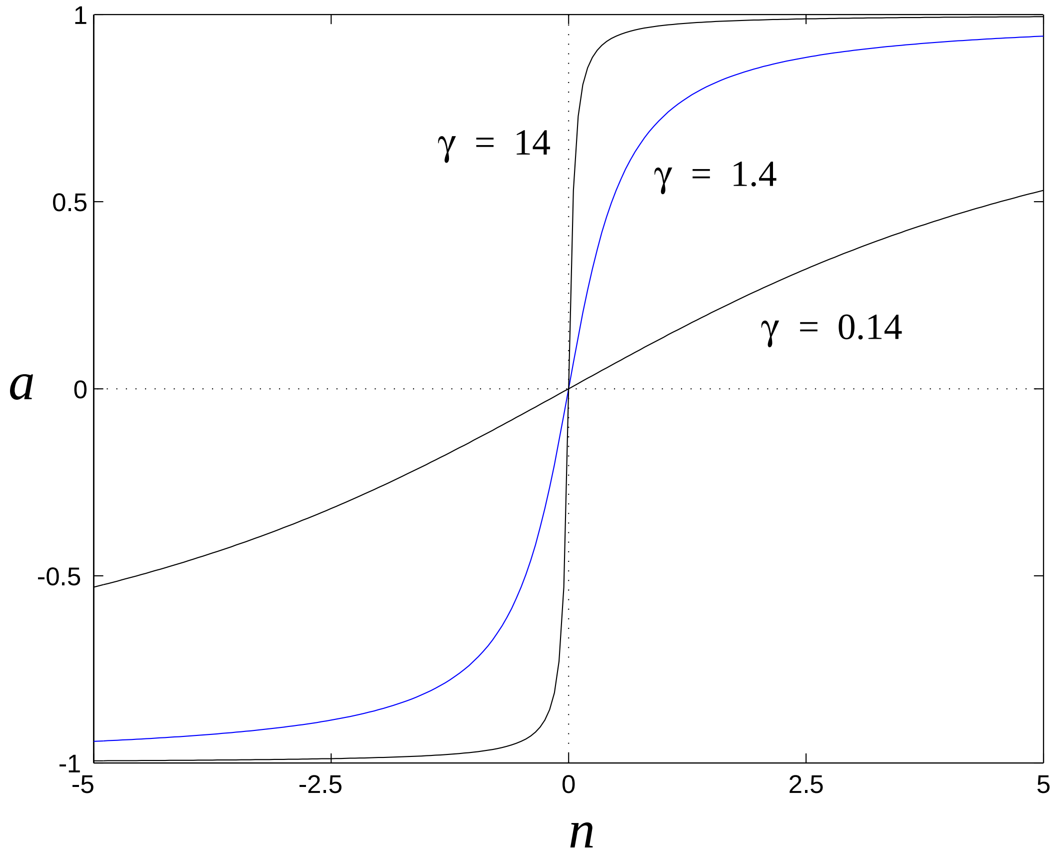
$$\frac{d}{da_i}[f^{-1}(a_i)] > 0$$

All points for which $\frac{d\mathbf{a}(t)}{dt} = \mathbf{0}$ will also satisfy $\nabla V(\mathbf{a}) = \mathbf{0}$

Therefore all attractors will be stationary points of $V(\mathbf{a})$.

Effect of Gain

$$a = f(n) = \frac{2}{\pi}\tan^{-1}\left(\frac{\gamma\pi n}{2}\right)$$



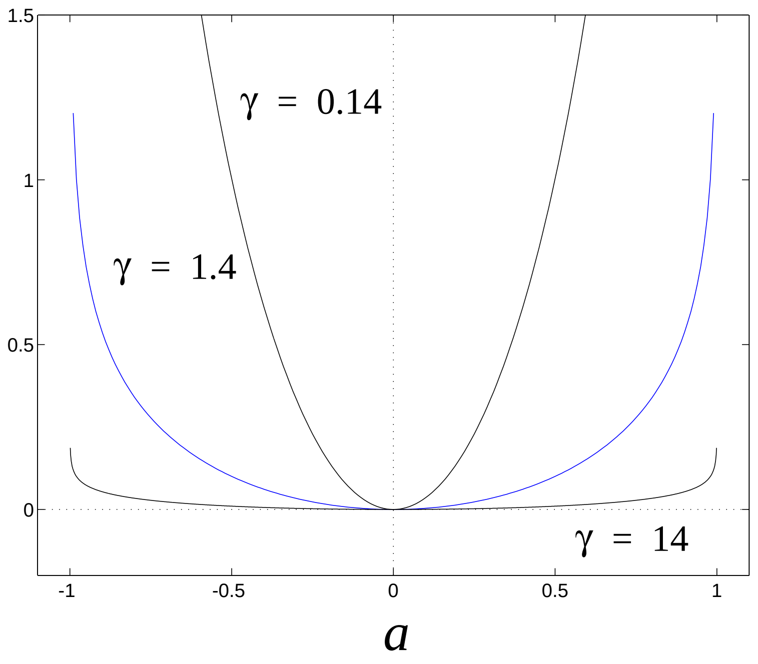
Lyapunov Function

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} + \sum_{i=1}^{S}\underbrace{\left\{\int_0^{a_i} f^{-1}(u)\,du\right\}}_{} - \mathbf{b}^T\mathbf{a}$$

$$f^{-1}(u) = \frac{2}{\gamma\pi}\tan\left(\frac{\pi u}{2}\right)$$

$$\int_0^{a_i} f^{-1}(u)\,du = \frac{2}{\gamma\pi}\left[\frac{2}{\pi}\log\left(\cos\left(\frac{\pi a_i}{2}\right)\right)\right] = -\frac{4}{\gamma\pi^2}\log\left[\cos\left(\frac{\pi a_i}{2}\right)\right]$$

$$-\frac{4}{\gamma\pi^2}\log\left[\cos\left(\frac{\pi a}{2}\right)\right]$$



High Gain Lyapunov Function

As $\gamma \rightarrow \infty$ the Lyapunov function reduces to:

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} - \mathbf{b}^T\mathbf{a}$$

The high gain Lyapunov function is quadratic:

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} - \mathbf{b}^T\mathbf{a} = \frac{1}{2}\mathbf{a}^T\mathbf{A}\mathbf{a} + \mathbf{d}^T\mathbf{a} + c$$

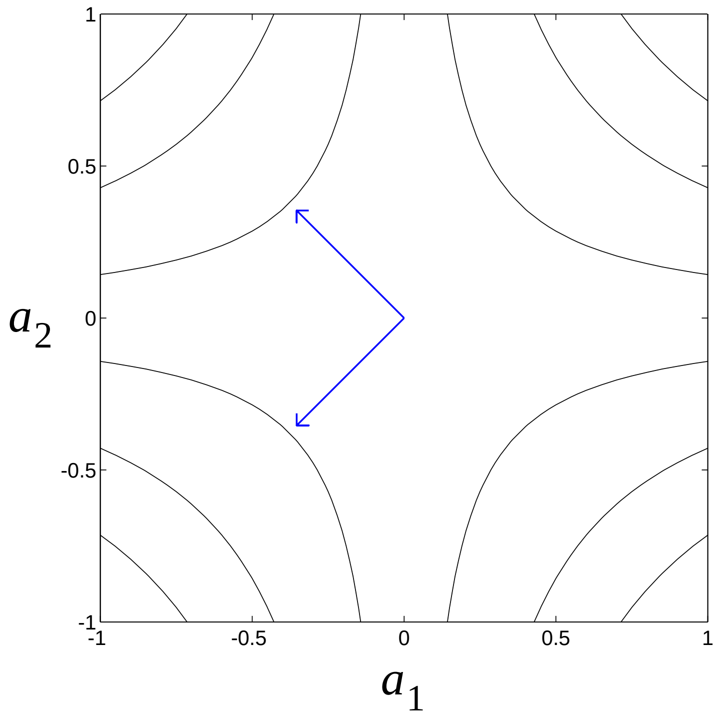
where

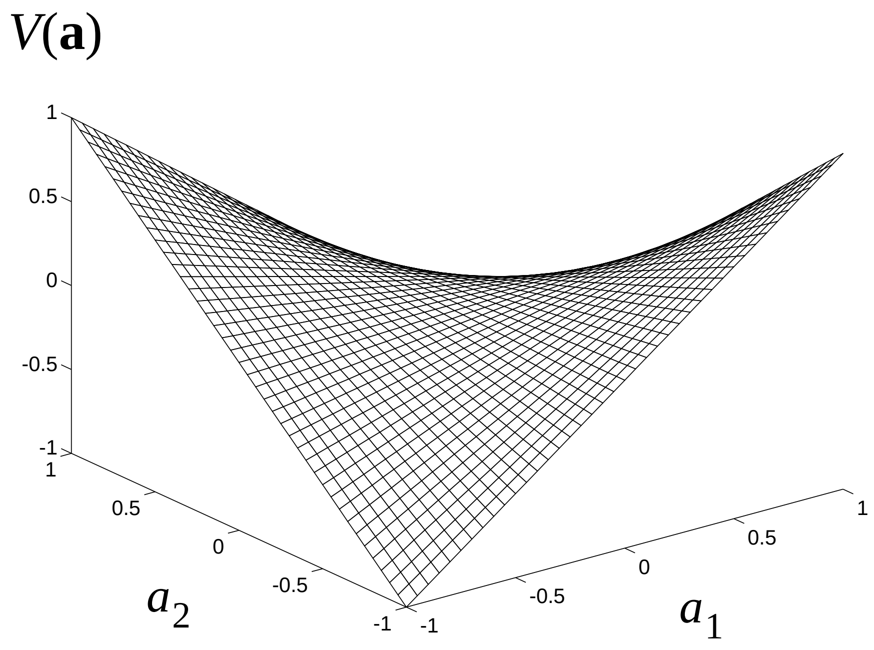
$$\nabla^2 V(\mathbf{a}) = \mathbf{A} = -\mathbf{W} \qquad \mathbf{d} = -\mathbf{b} \qquad c = 0$$

Example

$$\nabla^2 V(\mathbf{a}) = -\mathbf{W} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \qquad \left|\nabla^2 V(\mathbf{a}) - \lambda \mathbf{I}\right| = \begin{vmatrix} -\lambda & -1 \\ -1 & -\lambda \end{vmatrix} = \lambda^2 - 1 = (\lambda + 1)(\lambda - 1)$$

$$\lambda_1 = -1 \qquad \mathbf{z}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad\qquad \lambda_2 = 1 \qquad \mathbf{z}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$





Hopfield Design

The Hopfield network will minimize the following Lyapunov function:

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} - \mathbf{b}^T\mathbf{a}$$

Choose the weight matrix **W** and the bias vector **b** so that *V* takes on the form of a function you want to minimize.

Content-Addressable Memory

Content-Addressable Memory - retrieves stored memories on the basis of part of the contents.

Prototype Patterns:

$\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_Q\}$ (bipolar vectors)

Proposed Performance Index:

$$J(\mathbf{a}) = -\frac{1}{2}\sum_{q=1}^{Q}([\mathbf{p}_q]^T\mathbf{a})^2$$

For orthogonal prototypes, if we evaluate the performance index at a prototype:

$$J(\mathbf{p}_j) = -\frac{1}{2}\sum_{q=1}^{Q}([\mathbf{p}_q]^T\mathbf{p}_j)^2 = -\frac{1}{2}([\mathbf{p}_j]^T\mathbf{p}_j)^2 = -\frac{S}{2}$$

$J(\mathbf{a})$ will be largest when $\mathbf{a}$ is not close to any prototype pattern, and smallest when $\mathbf{a}$ is equal to a prototype pattern.

Hebb Rule

If we use the supervised Hebb rule to compute the weight matrix:

$$\mathbf{W} = \sum_{q=1}^{Q} \mathbf{p}_q (\mathbf{p}_q)^T \qquad \mathbf{b} = \mathbf{0}$$

the Lyapunov function will be:

$$V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} = -\frac{1}{2}\mathbf{a}^T\left[\sum_{q=1}^{Q} \mathbf{p}_q (\mathbf{p}_q)^T\right]\mathbf{a} = -\frac{1}{2}\sum_{q=1}^{Q} \mathbf{a}^T\mathbf{p}_q(\mathbf{p}_q)^T\mathbf{a}$$

This can be rewritten:

$$V(\mathbf{a}) = -\frac{1}{2}\sum_{q=1}^{Q} [(\mathbf{p}_q)^T\mathbf{a}]^2 = J(\mathbf{a})$$

Therefore the Lyapunov function is equal to our performance index for the content addressable memory.

Hebb Rule Analysis

$$\mathbf{W} = \sum_{q=1}^{Q} \mathbf{p}_q (\mathbf{p}_q)^T$$

If we apply prototype $\mathbf{p}_j$ to the network:

$$\mathbf{W}\mathbf{p}_j = \sum_{q=1}^{Q} \mathbf{p}_q (\mathbf{p}_q)^T \mathbf{p}_j = \mathbf{p}_j (\mathbf{p}_j)^T \mathbf{p}_j = S\mathbf{p}_j$$

Therefore each prototype is an eigenvector, and they have a common eigenvalue of S. The eigenspace for the eigenvalue $\lambda = S$ is therefore:

$$X = \text{span}\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_Q\}$$

An S-dimensional space of all vectors which can be written as linear combinations of the prototype vectors.

Weight Matrix Eigenspace

The entire input space can be divided into two disjoint sets:

$$R^S = X \cup X^{\perp}$$

where $X^{\perp}$ is the orthogonal complement of X. For vectors **a** in the orthogonal complement we have:

$$(\mathbf{p}_q)^T \mathbf{a} = 0, \quad q = 1, 2, \dots , Q$$

Therefore,

$$\mathbf{W a} = \sum_{q=1}^{Q} \mathbf{p}_q (\mathbf{p}_q)^T \mathbf{a} = \sum_{q=1}^{Q} (\mathbf{p}_q \cdot 0) = \mathbf{0} = 0 \cdot \mathbf{a}$$

The eigenvalues of **W** are S and 0, with corresponding eigenspaces of X and $X^{\perp}$. For the Hessian matrix

$$\nabla^2 V = -\mathbf{W}$$

the eigenvalues are -S and 0, with the same eigenspaces.

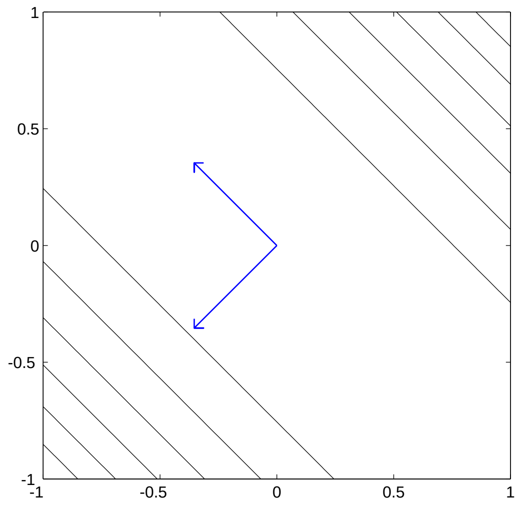
Lyapunov Surface

The high-gain Lyapunov function is a quadratic function. Therefore, the eigenvalues of the Hessian matrix determine its shape. Because the first eigenvalue is negative, V will have negative curvature in X. Because the second eigenvalue is zero, V will have zero curvature in $X^{\perp}$.

Because V has negative curvature in X , the trajectories of the Hopfield network will tend to fall into the corners of the hypercube $\{\mathbf{a}: -1 < a_i < 1\}$ that are contained in X.

Example

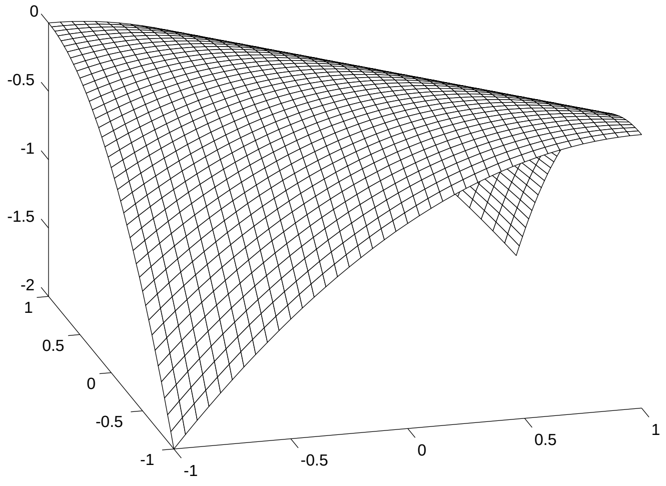
$$\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{W} = \mathbf{p}_1(\mathbf{p}_1)^T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad V(\mathbf{a}) = -\frac{1}{2}\mathbf{a}^T\mathbf{W}\mathbf{a} = -\frac{1}{2}\mathbf{a}^T\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\mathbf{a}$$



$$\nabla^2 V(\mathbf{a}) = -\mathbf{W} = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$$

$$\lambda_1 = -S = -2 \qquad \mathbf{z}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$X = \{\mathbf{a} : a_1 = a_2\}$$



$$\lambda_2 = 0 \qquad \mathbf{z}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$X^\perp = \{\mathbf{a} : a_1 = -a_2\}$$

Zero Diagonal Elements

We can zero the diagonal elements of the weight matrix:

$$\mathbf{W}' = \mathbf{W} - Q\mathbf{I}$$

The prototypes remain eigenvectors of this new matrix, but the corresponding eigenvalue is now (S-Q):

$$\mathbf{W}'\mathbf{p}_q = [\mathbf{W} - Q\mathbf{I}]\mathbf{p}_q = S\mathbf{p}_q - Q\mathbf{p}_q = (S - Q)\mathbf{p}_q$$

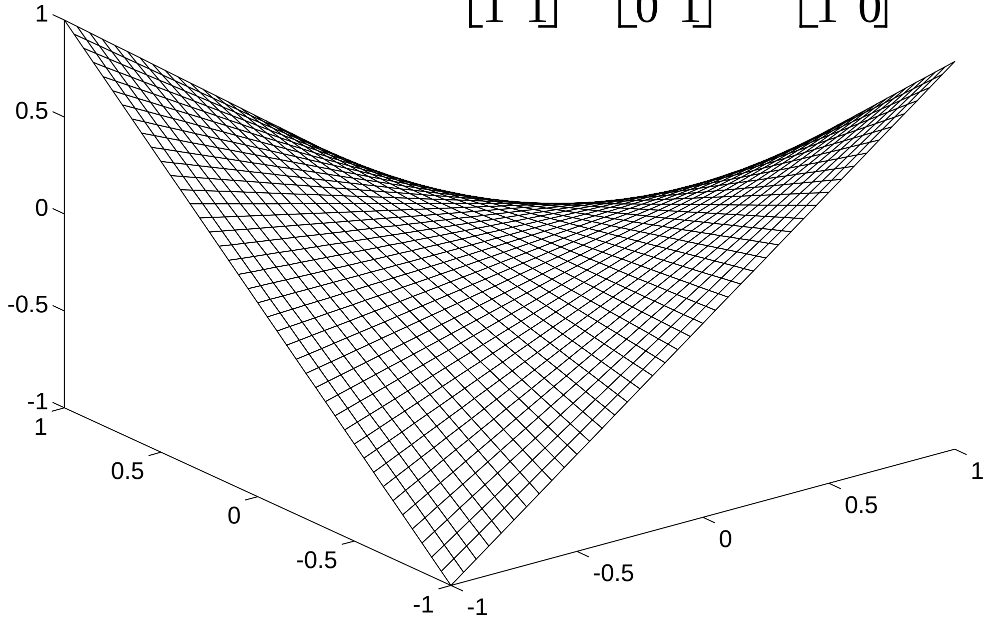
The elements of $X^{\perp}$ also remain eigenvectors of this new matrix, with a corresponding eigenvalue of (-Q):

$$\mathbf{W}'\mathbf{a} = [\mathbf{W} - Q\mathbf{I}]\mathbf{a} = \mathbf{0} - Q\mathbf{a} = -Q\mathbf{a}$$

The Lyapunov surface will have negative curvature in X and positive curvature in $X^{\perp}$, in contrast with the original Lyapunov function, which had negative curvature in X and zero curvature in $X^{\perp}$.

Example

$$\mathbf{W}' = \mathbf{W} - Q\mathbf{I} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



If the initial condition falls exactly on the line a_1=-a_2, and the weight matrix **W** is used, then the network output will remain constant. If the initial condition falls exactly on the line a_1=-a_2, and the weight matrix **W**' is used, then the network output will converge to the saddle point at the origin.