Neural Network Lecture

Day 4 and Day 5

Amir Jafari

HPSI
Human Performance Systems, Inc.

What is Machine learning?

- What is Machine Learning (ML)?
- In a technical term, ML is to program computers so that they can learn from input data.
- Learning is the process of acquisition of knowledge or skills through experience.
- The input to this learning process is training data and the output is some expertise that can perform some task.
- Since computers were invented, we wanted to know, computers might be made to learn. How to program them to learn and improve automatically.

Machine Learning in Our Life

- Imagine:
- Computers learn from medical records which treatments are most effective for new diseases
- Computers learn from houses experience to optimize energy costs based on patterns of occupants usage.
- Computers learn from evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper.
- Computer cannot learn nearly as well as people learn yet. However, algorithms have been invented that are effective for certain types of learning tasks, and a theoretical understanding of learning is beginning to emerge.

Machine Learning 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, fuel injector control, automatic braking systems, misfire detection, virtual emission sensors, warranty activity analyzers
- Banking
  - Check and other document readers, credit application evaluators, cash forecasting, firm classification, exchange rate forecasting, predicting loan recovery rates, measuring credit risk

More Machine Learning Applications

- 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
- Defense
  - Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression.
- Speech
  - Speech recognition, speech compression, vowel classification, text to speech synthesis
- Financial
  - Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis.

What is Deep Learning?

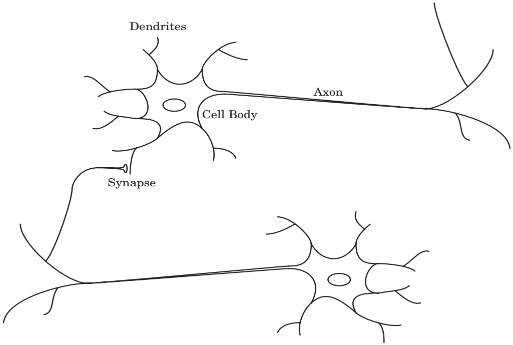
- ”Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, partially supervised or unsupervised.”

Deep Learning Applications

- 8 Inspirational Applications of Deep Learning
- 10 Deep Learning Applications
- 18 DEEP LEARNING STARTUPS
- GPU Application

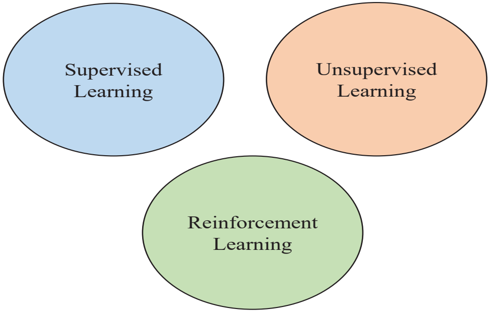
Brain Function

- The brain consists of a large number (approximately 10^{11}) of highly connected elements (approximately 10^4 connections per element) called neurons.



Machine Learning Basics

- The three different types of machine learning.



Supervised Learning

Supervised learning is a method to:

- Approximate a function from a training data set.
- Training data set includes both input and target.
- For each input data there is a target (desired results) associated with.
- Analyzing the training data set to approximate the function, it is called a classifier or pattern recognition (output is discrete) or a regression function (output is continuous).

My note

HPSI
Human Performance Systems, Inc.

Unsupervised Learning

Unsupervised learning is a method to:

- Infer a function from a training data set.
- Training data set includes just the inputs.
- For each input data there is no target.
- Since there is no target there is no error to evaluate the model.
- This distinguishes unsupervised learning from supervised learning.
- Mostly is used in the clustering applications.

My note

Reinforcement Learning

Reinforcement learning is a method to:

- How to map situations to actions.
- It is based on penalty for making mistakes and rewards for any success.
- For each input data there is no target.
- The learning happens based on the trail and error.
- It is inspired by behaviorist psychology.
- Mostly is used game theory, control theory and multi agent systems and so on.

My note

Problem Types

- **Fitting** (nonlinear regression). Map between a set of inputs and a corresponding set of targets. (e.g., estimate home prices from tax rate, estimate emission levels from fuel consumption and speed, predict body fat level from body measurements.)
- **Pattern recognition** (*classification*). Classify inputs into a set of target categories. (e.g., classify a tumor as benign or malignant, from uniformity of cell size.)
- **Clustering** (*segmentation*) Group data by similarity. (e.g., group customers according to buying patterns, group genes with related expression patterns.)
- **Prediction** (time series analysis, system identification, filtering or dynamic modeling). Predict the future value of some time series. (e.g., predict the future value of some stock.)

Softwares

We are going to use:

- Matlab (licensed). This is just for demos and you do not need to install it.
- Python (Open source), widely used in machine learning field.

Introduction

- Machine learning combines statistics and computer science fields.
- Statistics, probability, estimation and confidence intervals are some of main topics in machine learning in the statistical part.
- Linear algebra is another mathematical skill which is highly necessary in machine learning.
- Optimization theory and calculus are widely used in machine learning algorithms.

Why we need math?

- There are so many machine learning codes out there, and they are fairly simple to run.
- You need to download the packages and library to run the machine learning algorithms.
- However, to get some useful results and meaningful performance, you need to have a good mathematical background.
- After this lecture, you will get a glimpse of what types of mathematical skills you need to practice.
- In this lecture, we go over the main topics and further materials will be provided to you in order to strengthen you mathematical skills.

Definition of Probability?

- Relative Frequency.
- Subjective Probability.
- Axiomatic Probability.

Notation

- $a \in A$ a is member of A.
- $\cup$ is union, $\cap$ is intersection.
- $\sum$ is summation.
- $\int$ is integral.
- R is set of real numbers.
- **a,b,c** vector.
- **A,B,C** Matrix.
- $\frac{\sigma}{\sigma x} f(x)$ is a function.
- $y = f(x)$ is a function.
- $\frac{d}{dx} f(x)$ is a function.
- $||A||$ is norm A.
- a,b,c is set.
- $\emptyset$ is empty set.
- $\subset$ is subset.
- $y = f(\mathbf{x})$

My note - Set examples

My note - Venn Diagram

My note - Complement and DeMorgan’s Law

Axiomatic Probability

- $P(A) >= 0$
- $P(s) = 1$
- $P(A \cup B) = P(A) + P(B)$ A and B are disjoint.
- $A \cap B = \emptyset$ is disjoint.
- $A \cap B = P(A) \times P(B)$ if A and B are independent.

My note - Example rolling a die

Conditional Probability

- $P(A|B) = \frac{P(A \cap B)}{P(B)}$
- $P(B|A) = \frac{P(A \cap B)}{P(A)}$
- $P(A \cap B) = P(B|A) \times P(A)$
- $P(A \cap B) = P(A|B) \times P(B)$
- $P(B|A) = \frac{P(A|B) \times P(B)}{P(A)}$ BAYES RULE.

My note - Conditional Probability

Random variable

- A random variable is a mapping from sample space to the real line.
- $F_x(\lambda) = P(s : x(s) <= \lambda)$ Distribution function.
- $f_x(\lambda) = \frac{d}{d\lambda} F_x(\lambda)$ Probability Density function.
- $F_x(\lambda) = \int_{-\infty}^{\lambda} f_x(\mu) d\mu$
- $E[x] = \int_{-\infty}^{\infty} \lambda f_x(\lambda) d\lambda$

My note - Random variable

Random Variable

MATLAB:

```
gaussian_numbers =randn(1000)
hist(gaussian_numbers,20)
```

R:

```
BMI<-rnorm(n=1000, m=24.2, sd=2.2)
hist(BMI)
```

Python:

```
import matplotlib.pyplot as plt
import numpy as np
import plotly.plotly as py
gaussian_numbers = np.random.randn(1000)
plt.hist(gaussian_numbers)
plt.title("Gaussian Histogram")
```

HPSI
Human Performance Systems, Inc.

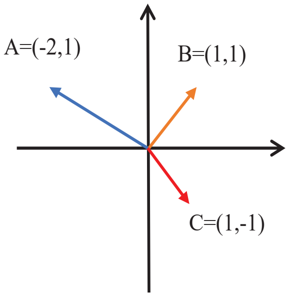
Linear algebra

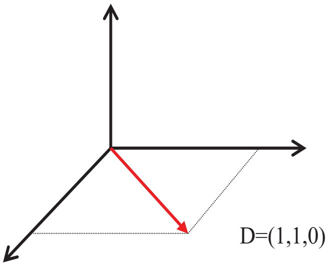
- Operations on or between vectors and matrices
- Linear regression, dimensionality reduction and solution of linear systems of equations are some application of linear algebra.
- Most common form of data in machine learning is in a vector form. The 2D array where rows represent samples represent columns attributes.
- A vector is a n-tuple of values usually real numbers where n is the the dimension of the vector, n can be any positive number.
- Vector is written in a column form.

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

Vectors

- Vector is a point in space or a line with magnitude and direction.





Vector arithmetic

- Addition of the vectors. $\mathbf{x} = \mathbf{a} + \mathbf{b}$
- Scalar multiplication. $\mathbf{x} = a\mathbf{x}$
- Dot product of vector. $a = \mathbf{x} \,.\, \mathbf{y} = \sum_i^n x_i y_i$ or $a = \mathbf{x} \,.\, \mathbf{y} = ||\mathbf{x}||||\mathbf{y}||cos(\theta)$

Matrix

- Matrix is mapping. It is an $m \times n$ two dimensional array.
- A vector is the bases set of a matrix. These bases set creates the matrix.
- A vector and matrix can be transposed. Transpose is the swap of rows and columns.
- Matrix is identified by two subscript. First element in subscript shows row number and the second element shows column number.
- A_{21} shows indicates to second row and first column.

$$\mathbf{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

Matrix arithmetic

- **C = A+B**, two matrix with the same dimension can be added ($c_{ij} = a_{ij} + b_{ij}$), the results has the same size.
- $\mathbf{A} = c.\mathbf{B}$, an scalar can be multiplied by each element of a matrix. ($A_{ij} = c + B_{ij}$), the results has the same size.
- Matrix multiplication.
- **A . (B . C) = (A . B) . C**, associative.
- $\mathbf{A} \,.\, \mathbf{B} \neq \mathbf{B}.\ \mathbf{A}$, commutative.
- $(\mathbf{A} \,.\, \mathbf{B})^T = \mathbf{B}^T.\mathbf{A}^T$, transposition.
- $\mathbf{A}\mathbf{A}^{-1} = I$, the inverse of an n-by-m matrix $\mathbf{A}$ is denoted $\mathbf{A}^{-1}$.

HPSI
Human Performance Systems, Inc.

Matrix arithmetic

- **C = A+B**, two matrix with the same dimension can be added ($c_{ij} = a_{ij} + b_{ij}$), the results has the same size.
- $\mathbf{A} = c.\mathbf{B}$, an scalar can be multiplied by each element of a matrix. ($A_{ij} = c + B_{ij}$), the results has the same size.
- **A . (B . C) = (A . B) . C**, associative, $\mathbf{A} \,.\, \mathbf{B} \neq \mathbf{B}.\ \mathbf{A}$, commutative and $(\mathbf{A} \,.\, \mathbf{B})^T = \mathbf{B}^T.\mathbf{A}^T$, transposition.
- In any chain of matrix multiplications, the column dimension of one matrix in the chain must match the row dimension of the following matrix in the chain.
- $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$, $(k\mathbf{A}^{-1}) = k^{-1}\mathbf{A}^{-1}$, $(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$, $(\mathbf{AB}^{-1}) = \mathbf{B}^{-1}\mathbf{A}^{-1}$

HPSI Human Performance Systems, Inc.

My note - Linear algebra

- Assume: **A** is (3x5), **B** is (5x5) and **C** is (3x1).
- $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{A}$, $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{A}^T$, $\mathbf{C}^T . \mathbf{A} \cdot \mathbf{B}$, $\mathbf{C} . \mathbf{A} \cdot \mathbf{B}$

Eigenvalues and Eigenvectors

- Eigenvalues and eigenvectors play a prominent role in the study of ODE and Linear Algebra and in many applications in the physical sciences.
- Let A be an $n \times n$ matrix. The value λ is an eigenvalue of A if there exists a non-zero vector v such that $Av = \lambda v$
- In this case, vector v is called an eigenvector of A corresponding to.

Eigenvalues

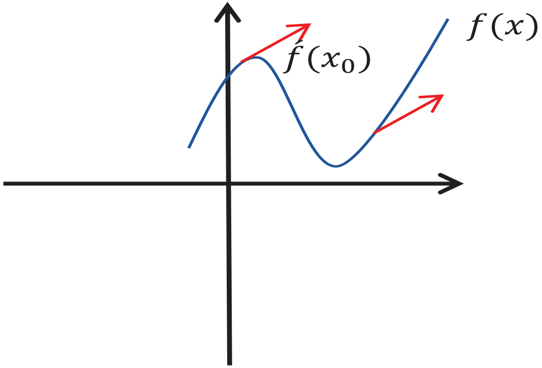
HPSI

Human Performance Systems, Inc.

Eigenvectors

Calculus derivative

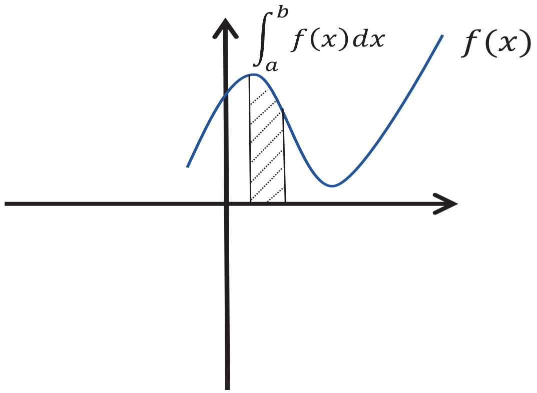
- A derivative, $\frac{d}{dx}f(x)$ is defined as the slope of a function $f(x)$. This is sometimes also written as $f'(x)$.



My note - Calculus derivative

Calculus integral

- Integrals are an inverse operation of the derivative (plus a constant).



My note - Calculus integral

Vector calculus

- Assume vector $\mathbf{x} = (x_0, x_1, ..., x_{n-1})^T$ and a function $f(x)$

$$\frac{\sigma f(x)}{\sigma x} = \begin{pmatrix} \frac{\sigma f(x)}{\sigma x_0} \\ \frac{\sigma f(x)}{\sigma x_1} \\ \vdots \\ \frac{\sigma f(x)}{\sigma x_{n-1}} \end{pmatrix}$$

- This is called the gradient of the function with respect to vector $\mathbf{x}$, written as $\nabla f(x)$ or ∇f

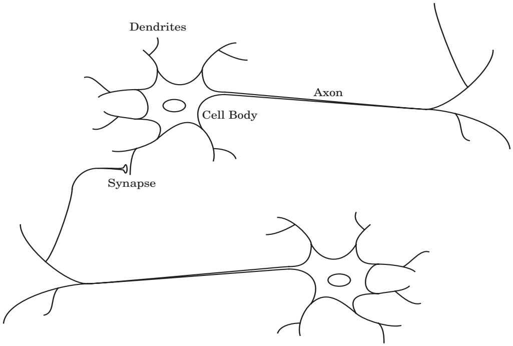
My note - Vector calculus

Summary

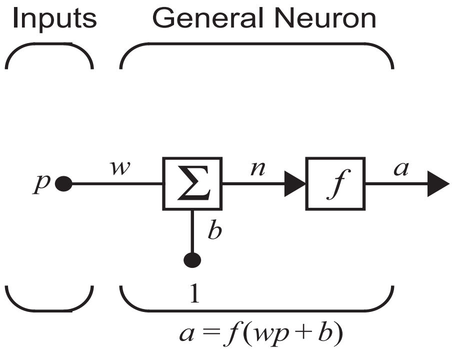
- We were touching the basic concepts of probability in this lecture.
- Linear algebra is very crucial in machine learning algorithms.
- This lecture just briefly introduces the main and basic concepts of math which we needed for basic machine learning algorithms.
- Check khan academy again if you need to review or refresh your mind.
- We are going to use these mathematical concepts through out the course and implement in into the computer programmes.

Brain Function

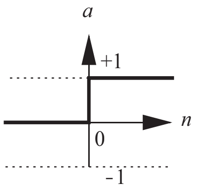
- Neurons Respond Slowly
  - 10^{-3} s compared to to 10^{-9} s for electrical circuits.
- The brain uses massively parallel comptation
  - $\approx 10^{11}$ neurons in the brain.
  - $\approx 10^{4}$ connections per neurons.



Single input model

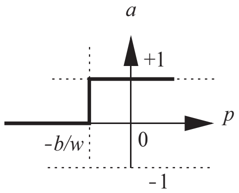


Transfer function



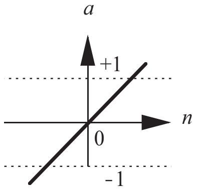
$a = hardlim(n)$

Hard Limit Transfer Function



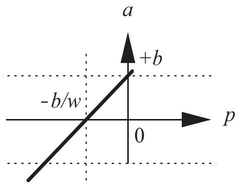
$a = hardlim(wp + b)$

Single-Input *hardlim* Neuron



$a = purelin(n)$

Linear Transfer Function

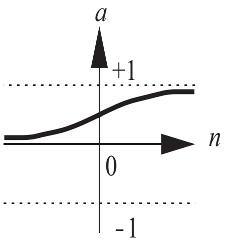


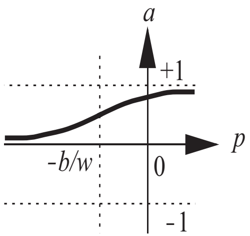
$a = purelin(wp + b)$

Single-Input *purelin* Neuron

HPSI Human Performance Systems, Inc.

Transfer function





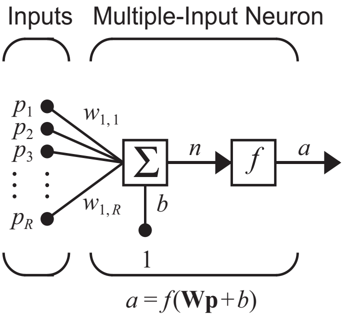
$a = logsig(n)$

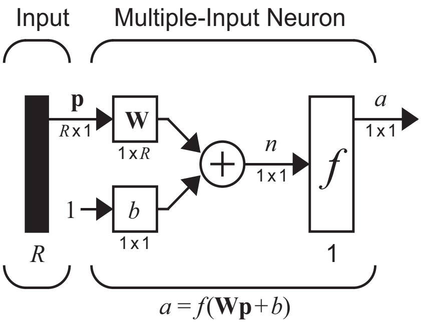
Log-Sigmoid Transfer Function

$a = logsig(wp + b)$

Single-Input *logsig* Neuron

Multiple input neuron

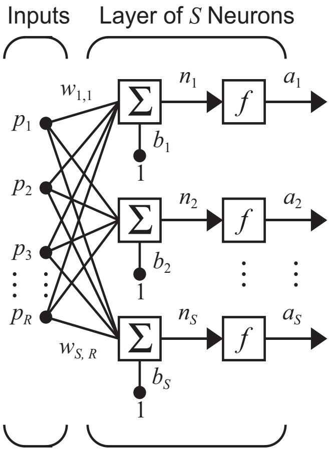




Abreviated Notation

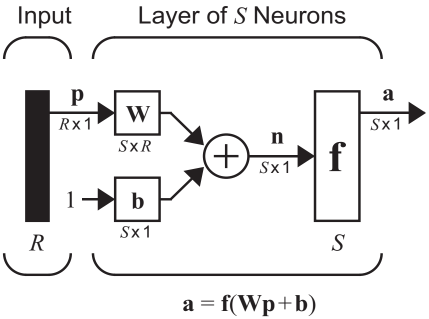
HPSI Human Performance Systems, Inc.

Layer of neurons



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

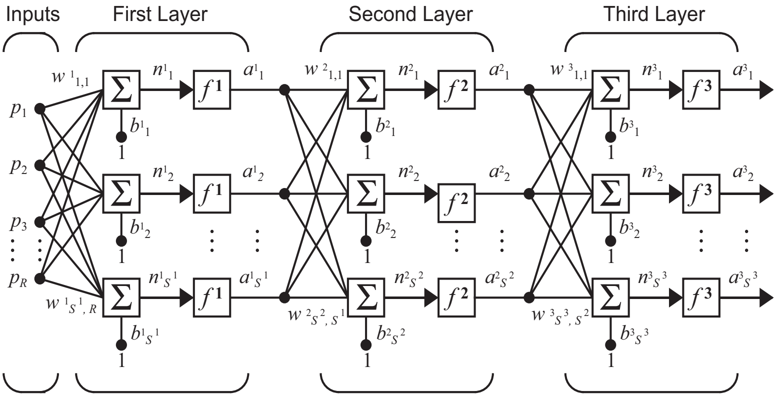
Abbreviated notation



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

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

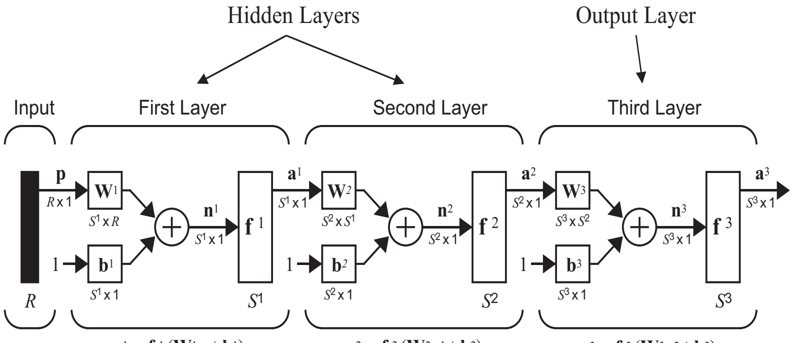
Multilayer network



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

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

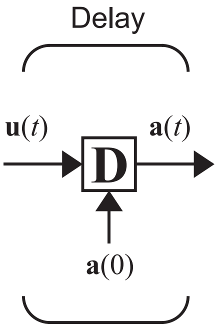
Abbreviated notation



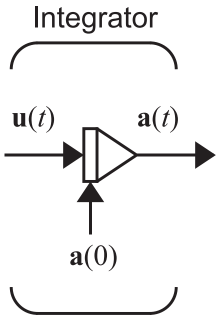
$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$

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

Delays and integrators

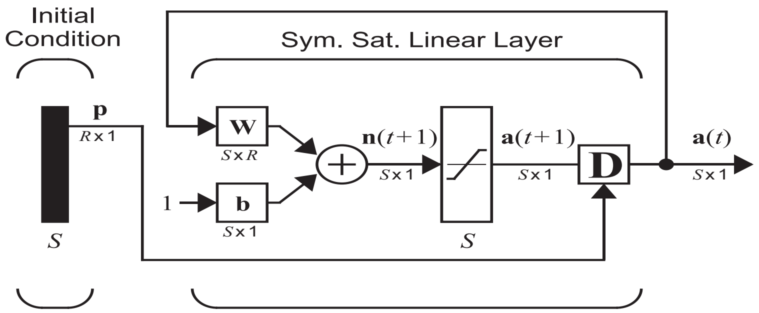


$$\mathbf{a}(t) = \mathbf{u}(t - 1)$$



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

Recurrent network

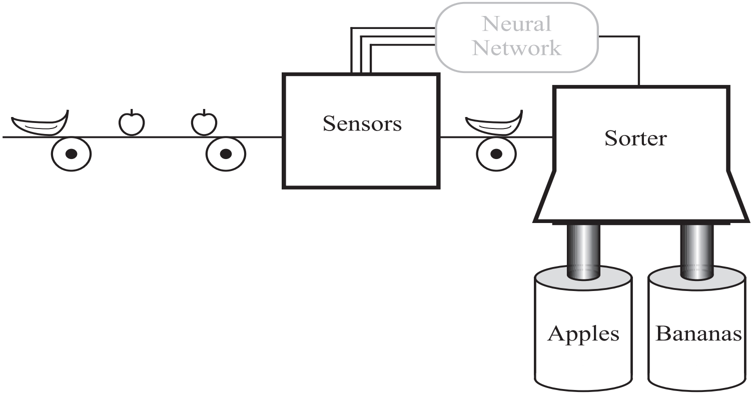


$$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlin}(\mathbf{W}\mathbf{a}(t)+\mathbf{b})$$

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

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

Apple banana sorter



Prototype vectors

Measurement Vector

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

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

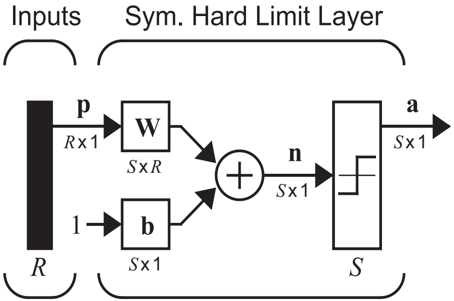
Prototype Banana

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

Prototype Apple

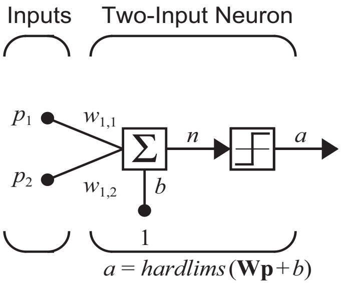
$$\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

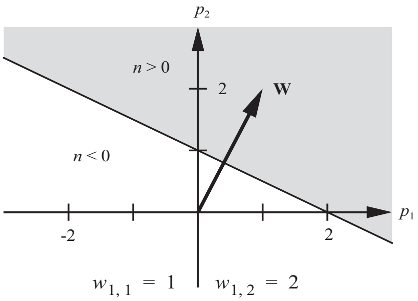
Perceptron



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

Two input case





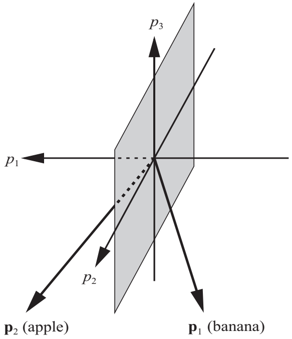
$$a = hardlims(n) = hardlims(\begin{bmatrix} 1 & 2 \end{bmatrix}\mathbf{p} + (-2))$$

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:

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

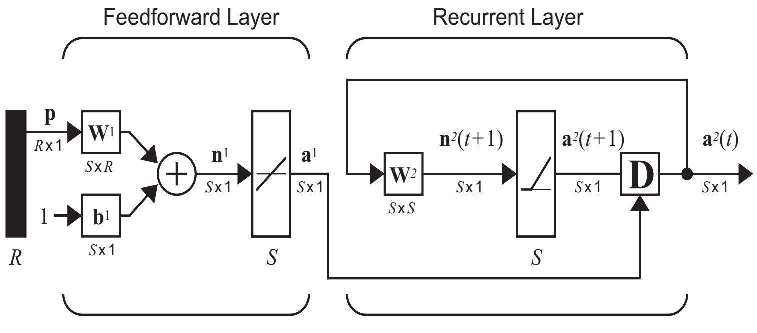
Apple:

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

“Rough” Banana:

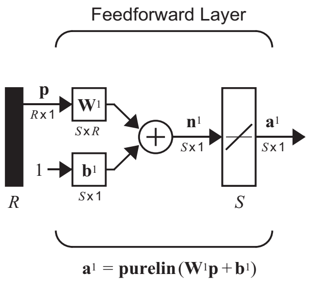
$$a = hardlims\left(\begin{bmatrix}-1 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\end{bmatrix} + 0\right) = 1(\text{banana})$$

Hamming Network



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

Feedforward Layer



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

For Banana/Apple Recognition

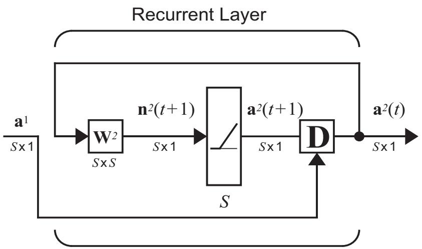
$$S = 2$$

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

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

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

Recurrent Layer

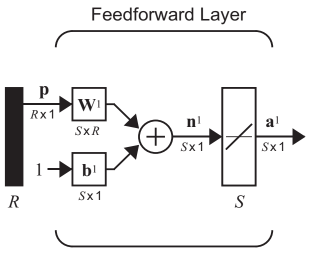


$$\mathbf{a}^2(0) = \mathbf{a}^1 \qquad \mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

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

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

Hamming Operation



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

For Banana/Apple Recognition

$$S = 2$$

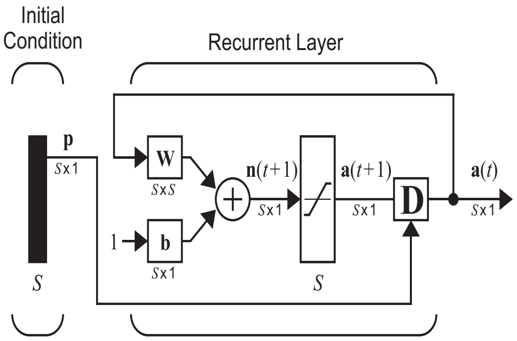
$$\mathbf{W}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

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

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

HPSI Human Performance Systems, Inc.

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

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

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 Prototype Pattern
  - # Neurons = # Elements is each Prototype Pattern

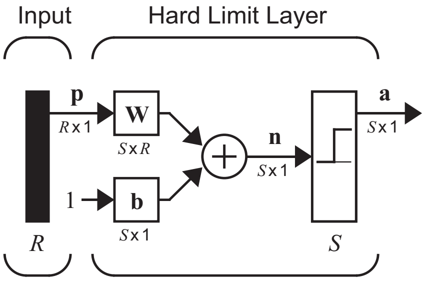
Learning rules

- Supervised learning: Network is provided with a set of examples of proper network behavior (inputs and targets)
  $\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_Q, t_Q\}$

- 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



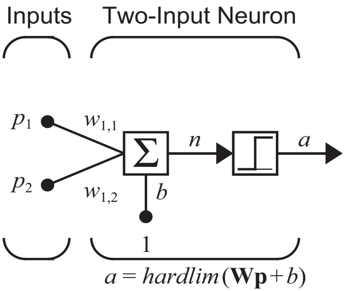
$$\mathbf{a} = \mathbf{hardlim}(\mathbf{Wp} + \mathbf{b})$$

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

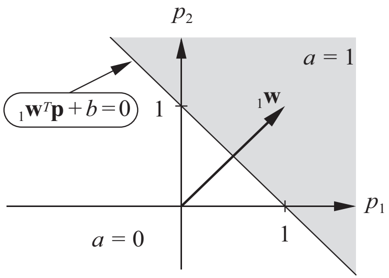
$${}_{i}\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} {}_{1}\mathbf{w}^{T} \\ {}_{2}\mathbf{w}^{T} \\ \vdots \\ {}_{S}\mathbf{w}^{T} \end{bmatrix}$$

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

Single neuron perceptron



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

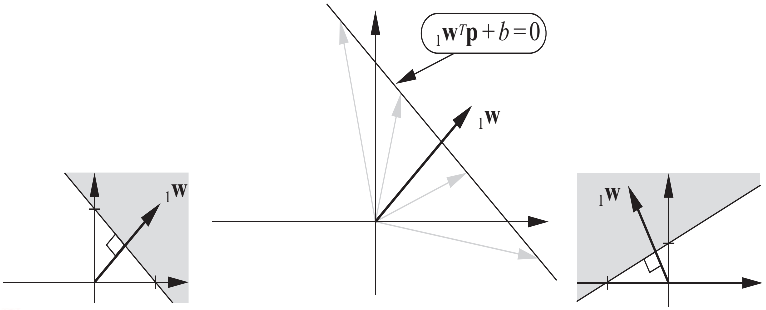


$$a = hardlim({}_1\mathbf{w}^T\mathbf{p} + b) = hardlim(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

HPSI Human Performance Systems, Inc.

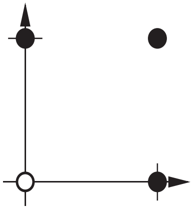
Decision boundary

- ${}_1\mathbf{w}^T\mathbf{p} + b = 0$, ${}_1\mathbf{w}^T\mathbf{p} = b$
- 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 tot he weight vector

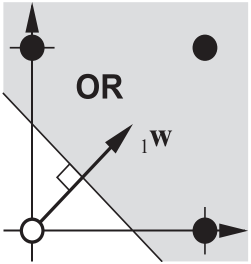


Example OR

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



OR solution



- Weight vector should be orthogonal to the decision boundary.
- $_i\mathbf{w} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$
- Pick a point on the decision boundary to find the bias.
- $_i\mathbf{wp} + b = [0.5 0.5] \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} + b = 0.25 + b = 0 \Rightarrow b = -0.25$

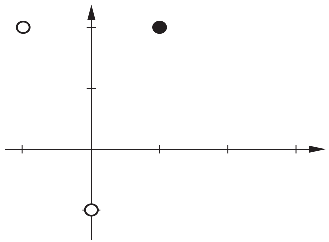
Multiple neuron perceptron

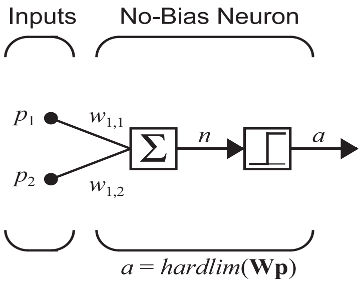
- Each neuron will have its own decision boundary.
- ${}_i\mathbf{w}^T\mathbf{p} + b_i = 0$
- A single neuron can classify input vectors into two categories.
- A multi neuron perceptron can classify input vectors into 2^S categories.

Learning rule test problem

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

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

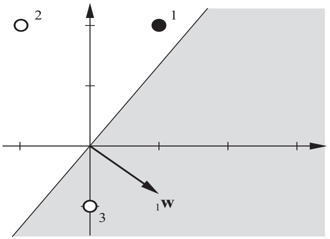




Starting point

Random initial weight:

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



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

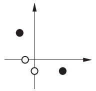
$$ a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) $$

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

Incorrect Classification.

Tentative learning rule

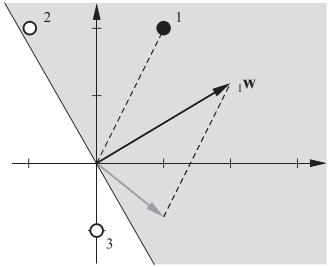
Set ${}_1\mathbf{w}$ to $\mathbf{p}_1$ ×
– Not stable



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

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

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



HPSI Human Performance Systems, Inc.

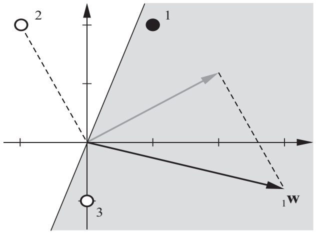
Second input vector

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

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

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

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

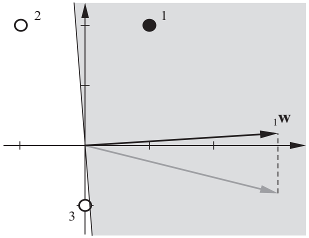


Third input vector

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

$$a = \text{hardlim}(0.8) = 1$$ (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

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

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

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

$$e = t - a$$

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

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

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

$$\boxed{\begin{gathered} {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p} \\ b^{new} = b^{old} + e \end{gathered}}$$

A bias is a weight with an input of 1.

HPSI
Human Performance Systems, Inc.

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

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

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

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

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

HPSI Human Performance Systems, Inc.

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

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

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

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

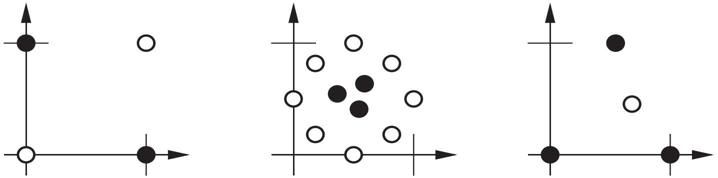
$$a = hardlim(-1.5) = 0 = t_2$$

Perceptron limitations

Linear Decision Boundary

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

Linearly Inseparable Problems



Python implementation

- Some library has to be imported to implement the perceptron learning rule.

```
>>>import numpy as np
>>>import pandas as pd
>>>import matplotlib.pyplot as plt
>>>ppn = Perceptron(eta=0.1, n_iter=10)
>>>ppn.fit(X, y)
```

Summary

- We were touching the main concepts behind neural network in this lecture.
- Neural network is the back bone of machine learning algorithms.
- This lecture just briefly introduces the main and basic concepts of neural network.
- Read chapter 1 through 4 again and look at the solved problems from the following book, Neural Network Design.
- We are going to use these concepts in this lecture for the first mini project.

Taylor series expansion

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

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

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

Example

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

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

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

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

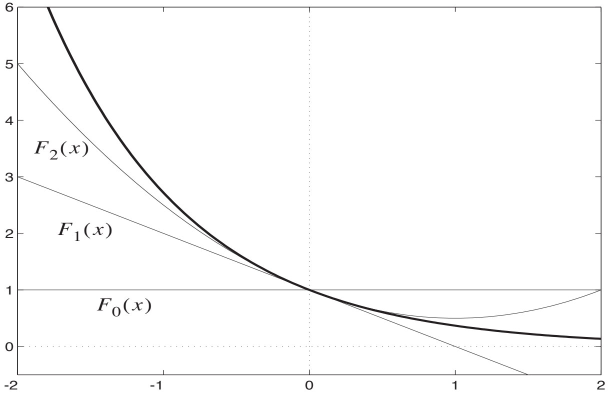
Taylor series approximations:

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

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

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

Plot of approximations



Vector case

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

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

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

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

Matrix form

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

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

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

HPSI
Human Performance Systems, Inc.

Directional derivatives

- First derivative (slope) of $F(\mathbf{x})$ along x_i axis: $\frac{\sigma F(\mathbf{x})}{\sigma x_i}$ **-** (i th element of gradient).

- Second derivative (curvature) of $F(\mathbf{x})$ along x_i axis: $\frac{\sigma^2 F(\mathbf{x})}{\sigma x_i^2}$ **-** (i, i th element of Hesian)

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

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

HPSI Human Performance Systems, Inc.

Example

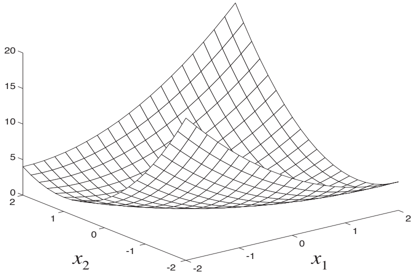
$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2$$

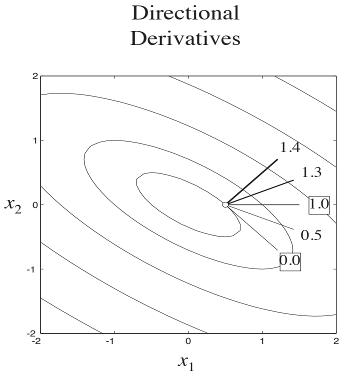
$$\mathbf{x}^* = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

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

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

Plots


20
15
10
5
0
2
1
0
-1
-2
-2
-1
0
1
2
x_2
x_1


Directional Derivatives
1.4
1.3
1.0
0.5
0.0
x_2
x_1
2
1
0
-1
-2
-2
-1
0
1
2

Minima

Strong Minimum:

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

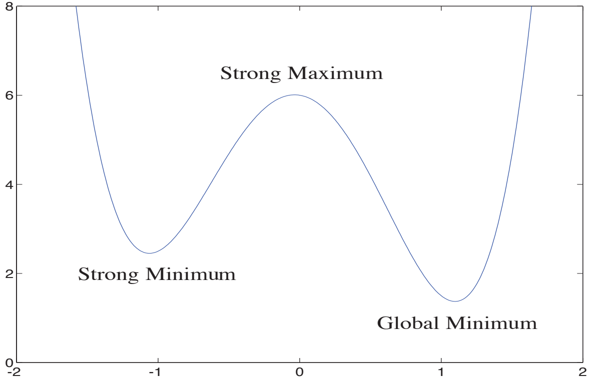
Weak Minimum:

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

HPSI Human Performance Systems, Inc.

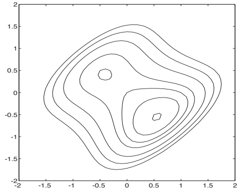
Scalar example

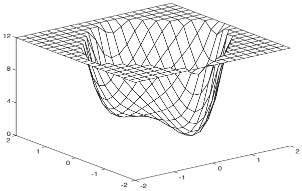
$$F(x) = 3x^4 - 7x^2 - \frac{1}{2}x + 6$$



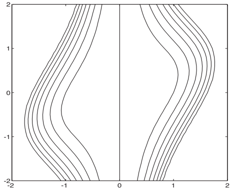
Vector Example

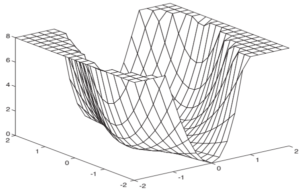
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$





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





Human Performance Systems, Inc.

Example

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

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

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

(Not a function of $\mathbf{x}$ 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 strong minimum.

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:

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

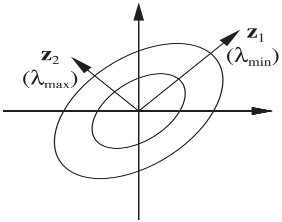
HPSI Human Performance Systems, Inc.

Eigenvector

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

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

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

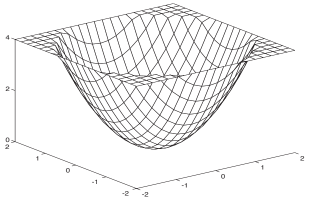


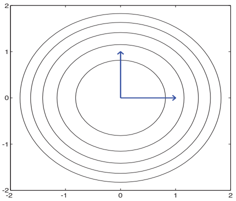
Circular hollow

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

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

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

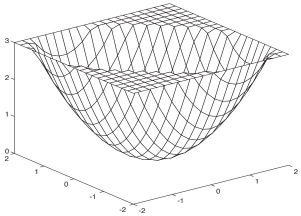


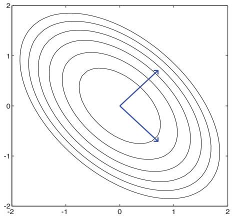


Elliptical hollow

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

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

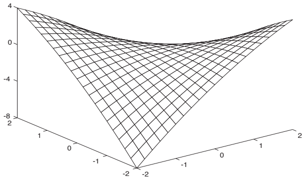


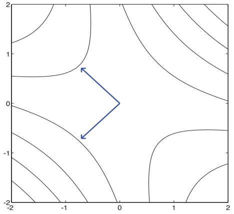


Elongated saddle

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

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

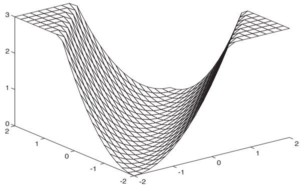


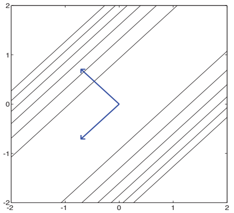


Stationary Valley

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

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





OR solution

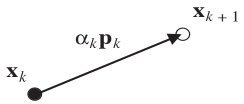
- If the eigenvalues of the Hessian matrix are all positive, the function will have a single strong minimum.
- If the eigenvalues are all negative, the function will have a single strong maximum.
- If some eigenvalues are positive and other eigenvalues are negative, the function will have a single saddle 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: $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{d}$

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

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

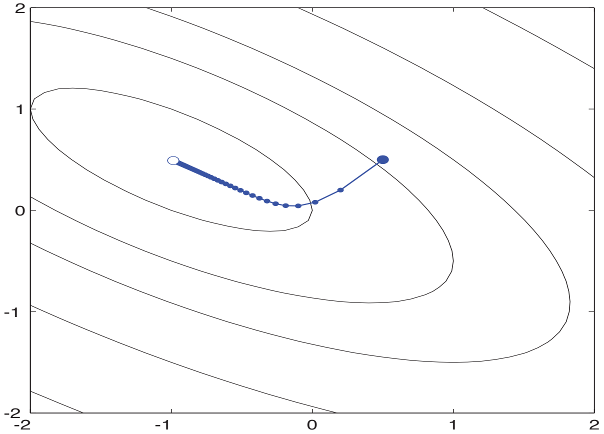
$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \alpha = 0.1$$

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

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

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

Plot



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

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

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

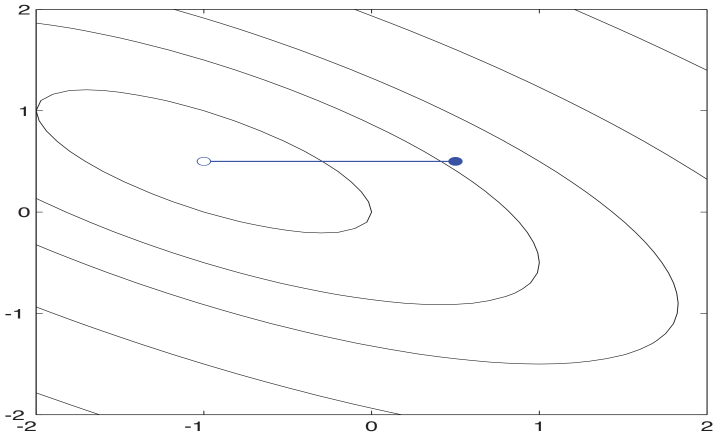
$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix}$$

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

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

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

Plot



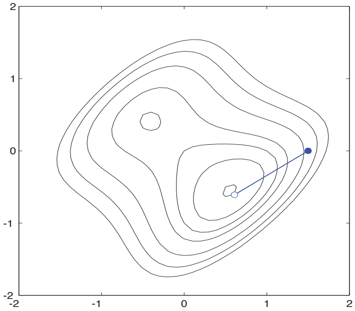
Human Performance Systems, Inc.

Non-Quadratic Example

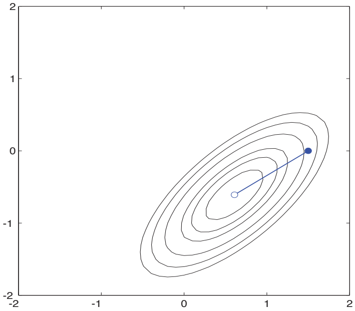
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

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

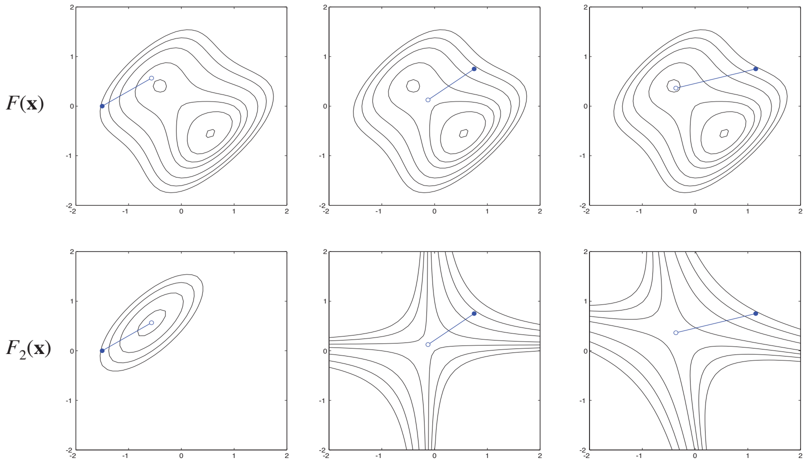
$F(\mathbf{x})$



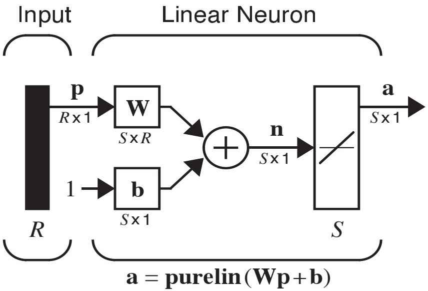
$F_2(\mathbf{x})$



Different Initial Conditions



ADALINE Network

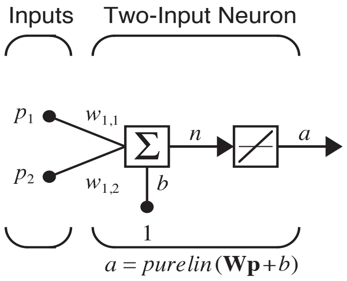


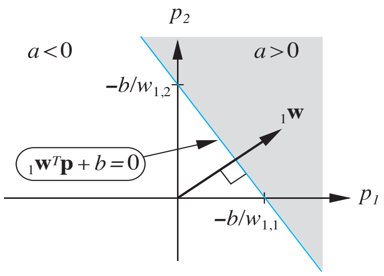
$$\mathbf{a} = \mathbf{purelin}(\mathbf{W}\mathbf{p} + \mathbf{b}) = \mathbf{W}\mathbf{p} + \mathbf{b}$$

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

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

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:

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

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

Notation:

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

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:

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

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

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

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

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

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

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

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:

$$ \boxed{\begin{aligned} \mathbf{W}(k+1) &= \mathbf{W}(k) + 2\alpha\mathbf{e}(k)\mathbf{p}^T(k) \\ \mathbf{b}(k+1) &= \mathbf{b}(k) + 2\alpha\mathbf{e}(k) \end{aligned}} $$

Example

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

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

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

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

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

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

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

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

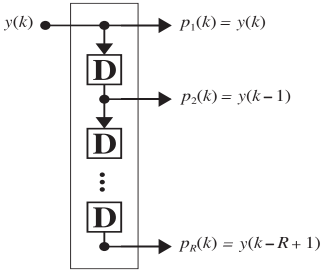
$$\mathbf{W}(3) = \mathbf{W}(2) + 2\alpha e(2)\mathbf{p}^T(2) = \begin{bmatrix} 1.1040 & 0.0160 & -0.0160 \end{bmatrix}$$

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

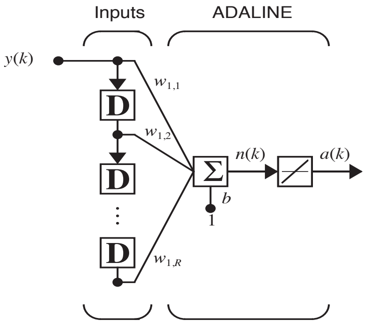
HPSI
Human Performance Systems, Inc.

Adaptive Filtering

Tapped Delay Line



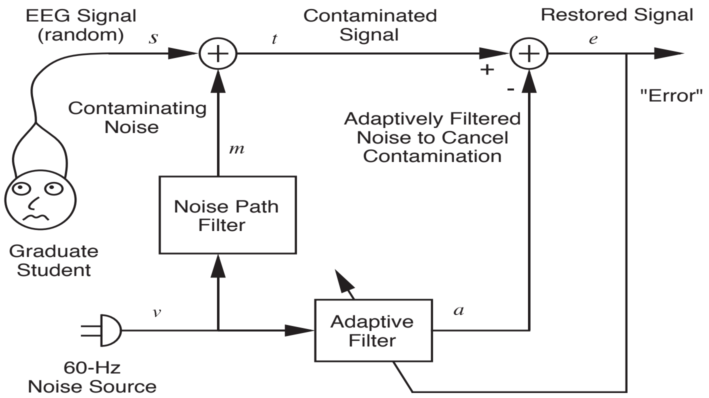
Adaptive Filter



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

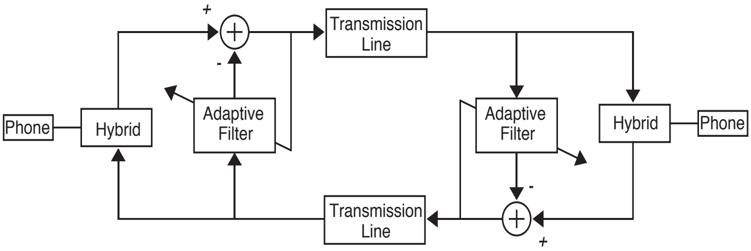
$$a(k) = purelin(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1,i}\, y(k-i+1) + b$$

Example: Noise Cancellation

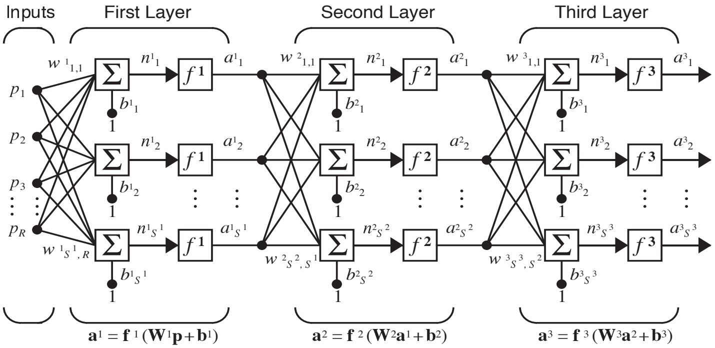


Adaptive Filter Adjusts to Minimize Error (and in doing this removes 60-Hz noise from contaminated signal)

Echo Cancellation



Multilayer Perceptron



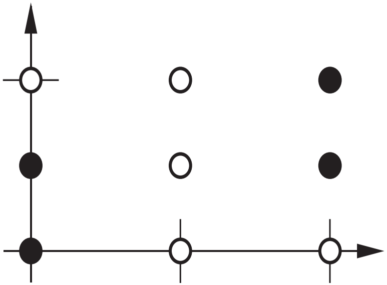
$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$

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

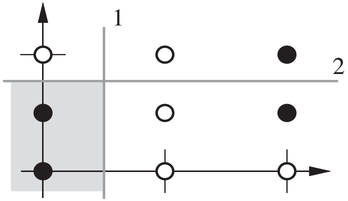
$R - S^1 - S^2 - S^3$ Network

HPSI Human Performance Systems, Inc.

Example



Elementary Decision Boundaries



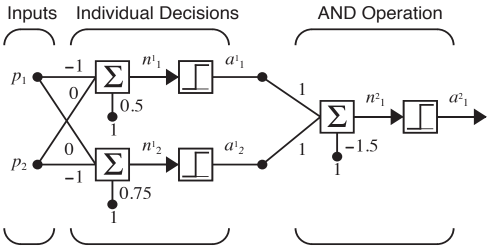
First Boundary:

$$a_1^1 = hardlim\left(\begin{bmatrix}-1 & 0\end{bmatrix}\mathbf{p} + 0.5\right)$$

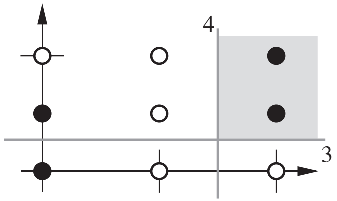
Second Boundary:

$$a_2^1 = hardlim\left(\begin{bmatrix}0 & -1\end{bmatrix}\mathbf{p} + 0.75\right)$$

First Subnetwork



Elementary Decision Boundaries



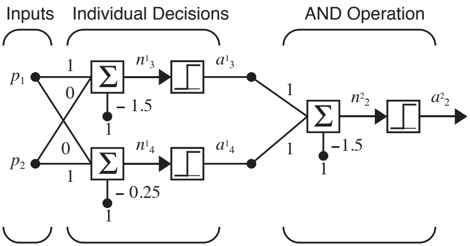
Third Boundary:

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

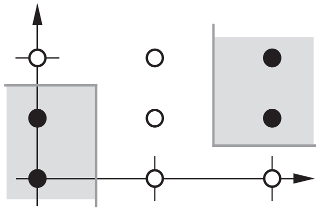
Fourth Boundary:

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

Second Subnetwork



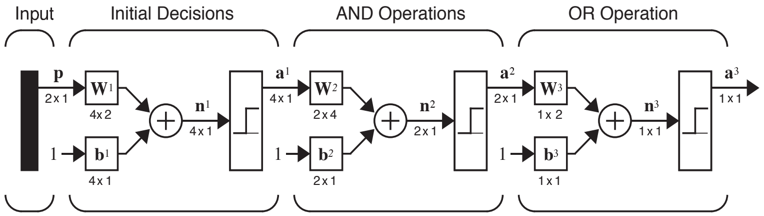
Total Network



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

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

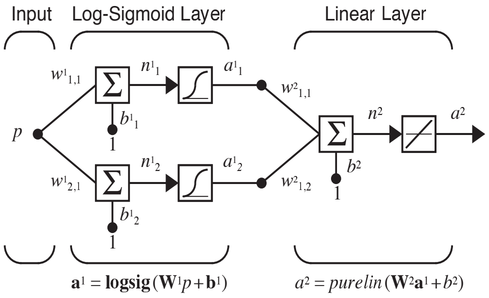
$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathbf{b}^3 = \begin{bmatrix} -0.5 \end{bmatrix}$$



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

HPSI Human Performance Systems, Inc.

Function Approximation Example



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

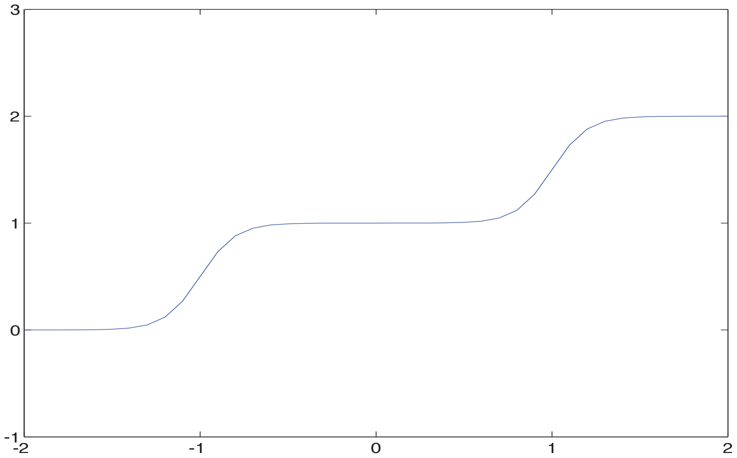
$$f^2(n) = n$$

Nominal Parameter Values

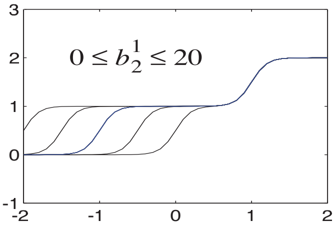
$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -10 \qquad b^1_2 = 10$$

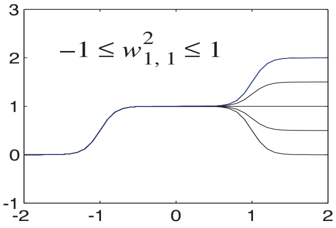
$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = 0$$

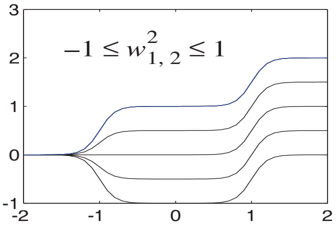
Nominal Response

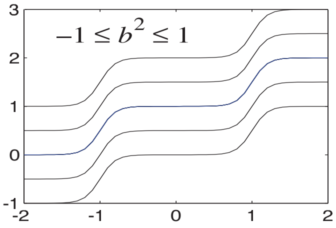


Parameter Variations



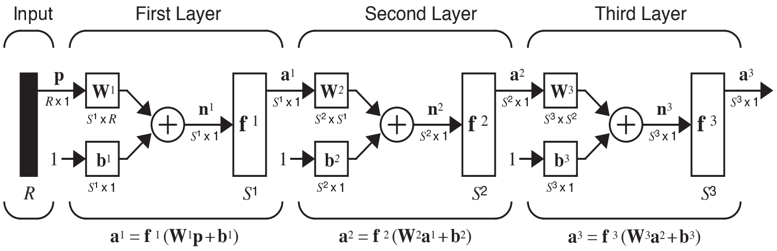






HPSI Human Performance Systems, Inc.

Multilayer Network



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

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

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

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

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

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

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

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

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

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

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

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

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

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

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

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

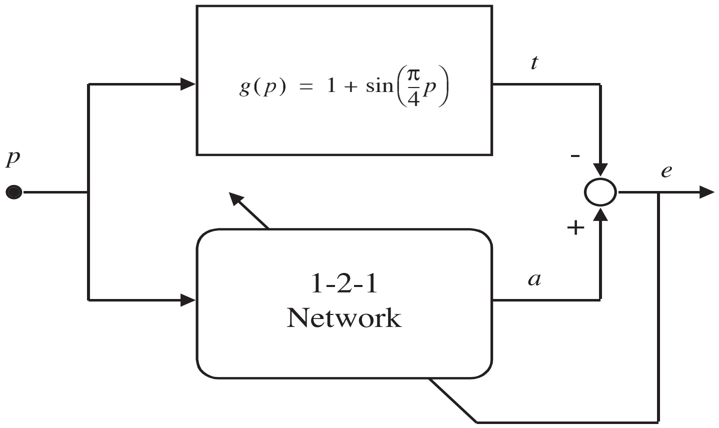
$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$$

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

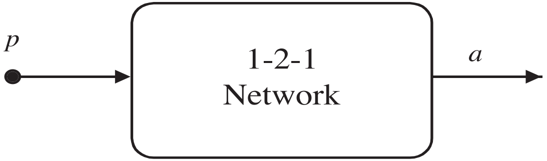
Weight Update

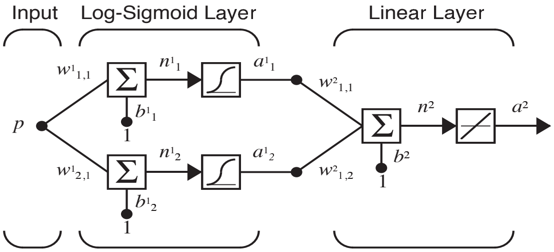
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

Example: Function Approximation



Network



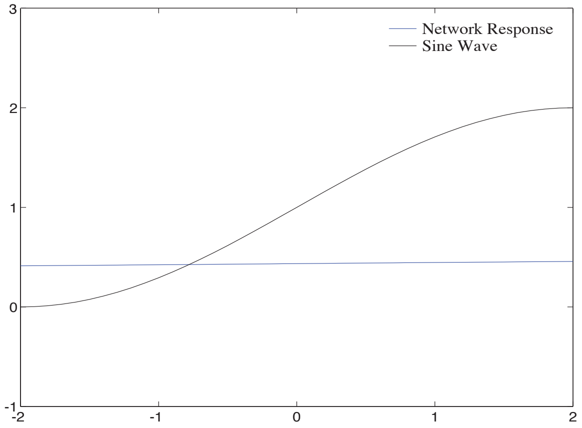


$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$ $a^2 = purelin(\mathbf{W}^2 \mathbf{a}^1 + b^2)$

HPSI
Human Performance Systems, Inc.

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

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

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

HPSI
Human Performance Systems, Inc.

Backpropagation

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

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

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

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

Weight Update

$$\alpha = 0.1$$

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

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

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

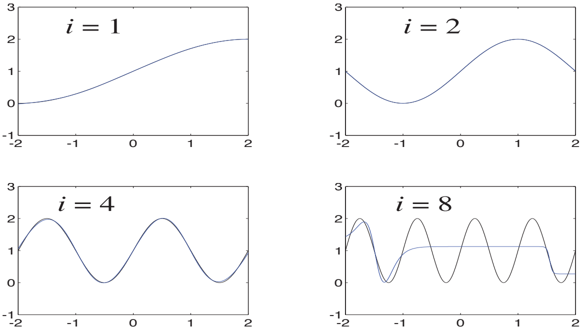
$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1(\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

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

Choice of Architecture

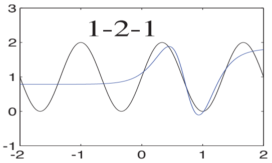
$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right)$$

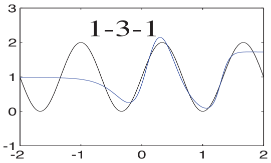
1-3-1 Network

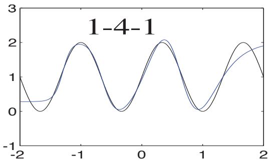


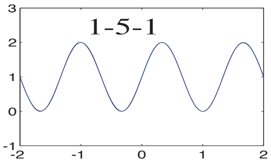
Choice of Network Architecture

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



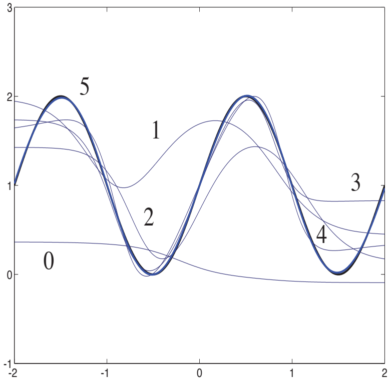


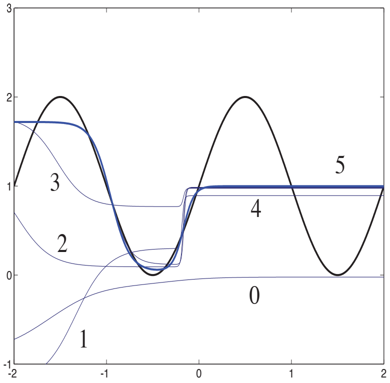




Convergence

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

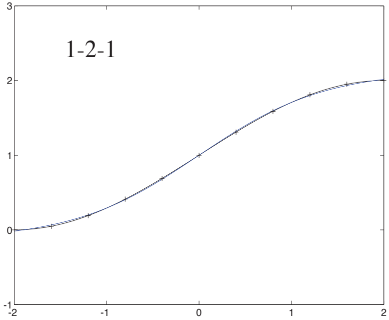


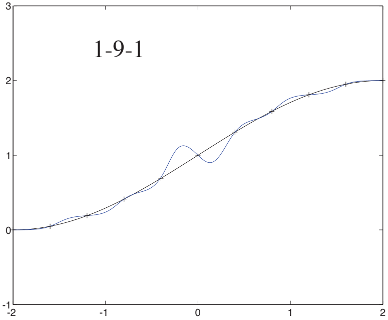


Generalization

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

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





Generalization

- A cat that once sat on a hot stove will never again sit on a hot stove or on a cold one either.

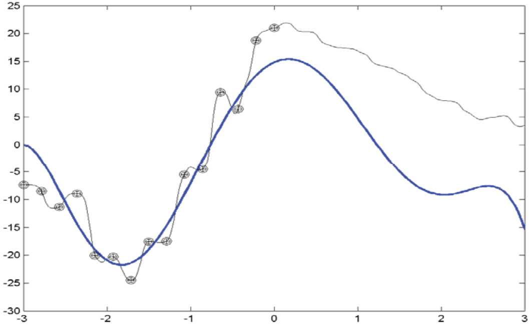
Mark Twain

Cause of Overfitting

- The network input-output mapping is accurate for the training data and for test data never seen before.
- The network interpolates well.
- Poor generalization is caused by using a network that is too complex (too many neurons/parameters). To have the best performance we need to find the least complex network that can represent the data (Ockham’s Razor).
- Find the simplest model that explains the data.

Good Generalization

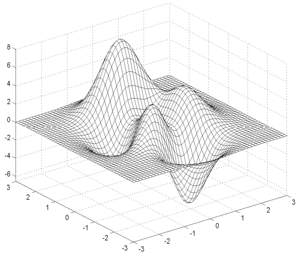
Overfitting Extrapolation

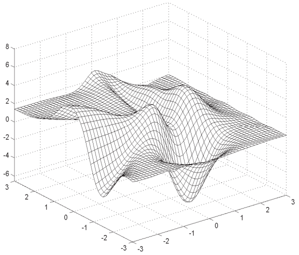


Interpolation

HPSI Human Performance Systems, Inc.

Extrapolation in 3-D





Measuring Generalization

- Part of the available data is set aside during the training process.
- After training, the network error on the test set is used as a measure of generalization ability.
- The test set must never be used in any way to train the network, or even to select one network from a group of candidate networks.
- The test set must be representative of all situations for which the network will be used.

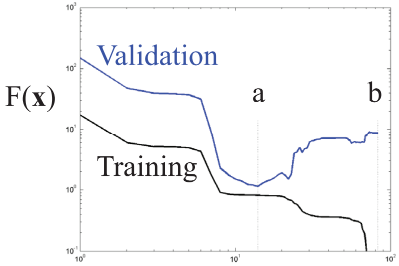
Methods for Improving Generalization

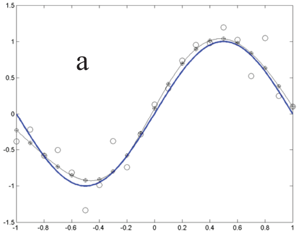
- Pruning (removing neurons) until the performance is degraded.
- Growing (adding neurons) until the performance is adequate.
- Validation Methods
- Regularization

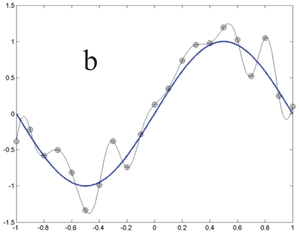
Early Stopping

- Break up data into training, validation, and test sets.
- Use only the training set to compute gradients and determine weight updates.
- Compute the performance on the validation set at each iteration of training.
- Stop training when the performance on the validation set goes up for a specified number of iterations.
- Use the weights which achieved the lowest error on the validation set.

Early Stopping Example







HPSI
Human Performance Systems, Inc.

What is deep learning?

- Deep learning is a branch of machine learning involving algorithms that have many nonlinear processing stages.
- In most cases, deep learning refers to training neural networks that have many layers.
- Before approximately 2006, most neural network applications used one or two hidden layers.
- Since that time, the development of more powerful GPUs, and the associated general purpose programming languages, enabled larger neural networks to be tested.
- Larger networks require large data sets to prevent overfitting. The number of large data sets has increased dramatically in recent years.

Deep Learning Applications

- Google Deepmind – AlphaGO, the first computer GO program to beat a top professional GO player.
- Android operating system speech recognition.
- photosearch for Google+.
- Skype translator – speech recognition.
- Microsoft Cortana digital assistant.
- Facebook – Deep Face, face recognition.
- Apple – Siri

Course objective

- Learn about the most popular deep learning architectures.
- Learn about the most popular open source deep learning software frameworks.
- Learn how to implement deep networks using deep learning frameworks.

Course emphasis

- Key emphasis will be on implementation of deep learning concepts on GPUs using open source software frameworks.
- Will not cover basic machine learning concepts.
- Will assume knowledge of key ideas from linear algebra, optimization, probability, machine learning (as covered, for example, in Neural Network Design – hagan.okstate.edu/nnd.html)

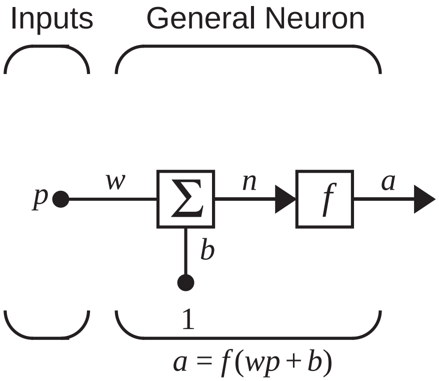
Course schedule

- Introduction
- Multilayer networks
- Training multilayer networks, gradient calculation
- Torch
- Convolution networks
- Training convolution networks, gradient calculation
- Caffe
- Restricted Boltzmann machine
- Deep belief network
- Theano
- Long short term memory
- Training recurrent networks
- TensorFlow

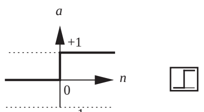
Brief history of deep learning

- Backpropagation for multilayer networks discovered, popularized (1974, 1982, 1985, 1986).
- Convolution networks introduced (1989).
- Long Short Term Memory network developed (1997).
- Deep belief network presented (2006).
- NVIDIA unveiled CUDA, a language for general purpose programming of GPUs (2006)

Basic network building block (neuron)

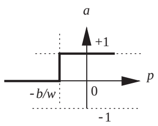


Transfer (activation) functions (1)



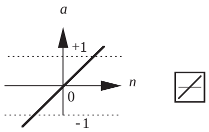
$a = hardlim(n)$

Hard Limit Transfer Function



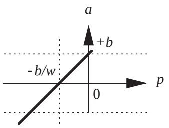
$a = hardlim(wp + b)$

Single-Input *hardlim* Neuron



$a = purelin(n)$

Linear Transfer Function

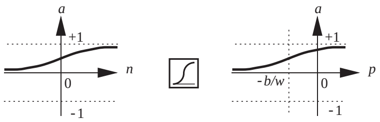


$a = purelin(wp + b)$

Single-Input *purelin* Neuron

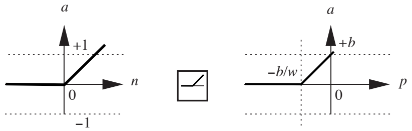
HPSI Human Performance Systems, Inc.

Transfer (activation) functions (2)



$a = logsig(n)$ Log-Sigmoid Transfer Function

$a = logsig(wp + b)$ Single-Input *logsig* Neuron



$a = poslin(n)$ Positive Linear Function

$a = poslin(wp + b)$ Single-Input *poslin* Neuron

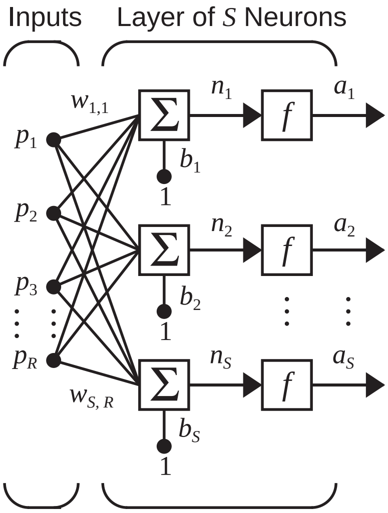
Transfer (activation) functions (3)

Softmax

$$a_i = f_i(\mathbf{n}) = \frac{e^{n_i}}{\sum_{j=1}^{S} e^{n_j}}$$

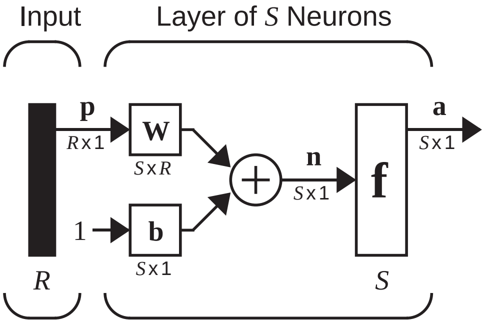
Used at the output layer of a pattern recognition network with multiple output neurons.

Layer of neurons



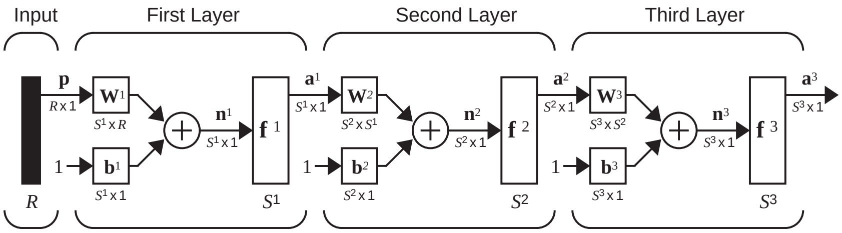
$$\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

Matrix notation



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

Multiple layer network

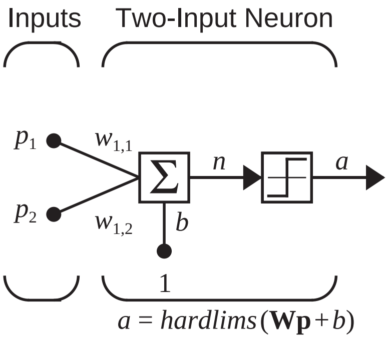


$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$ $\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$

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

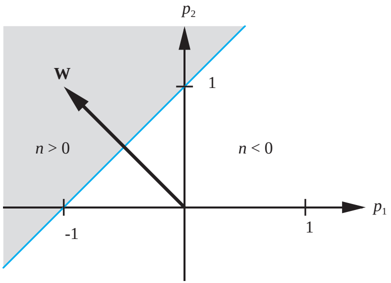
HPSI Human Performance Systems, Inc.

Single layer network decision boundary

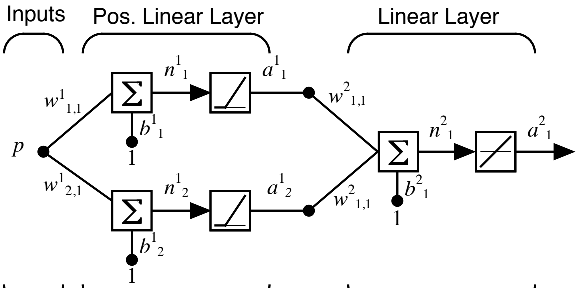


Decision boundary

$$n = \mathbf{Wp} + b == 0$$



Poslin network

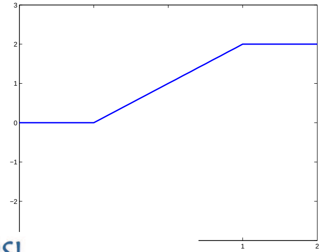


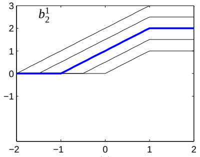
$\mathbf{a}^1 = \mathbf{poslin}(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)$ $\mathbf{a}^2 = \mathbf{purelin}(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2)$

Poslin network function

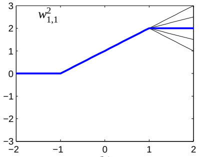
$$\mathbf{W}^1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T, \mathbf{b}^1 = \begin{bmatrix} -1 & 1 \end{bmatrix}^T$$

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

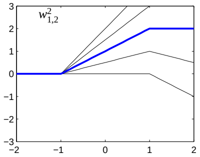




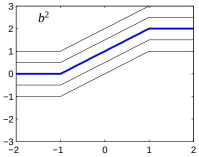
(a)



(b)



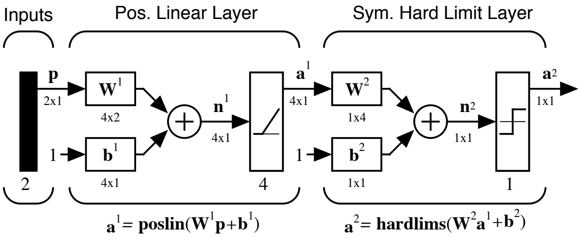
(c)



(d)

HPSI Human Performance Systems, Inc.

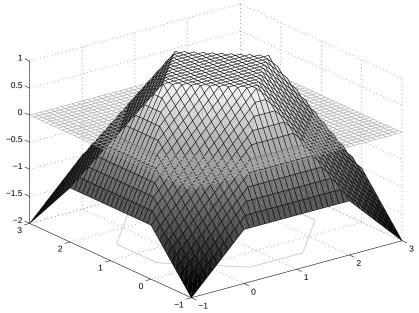
2D poslin network

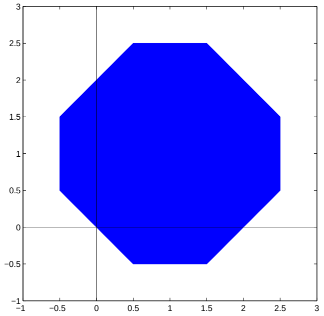


$$\mathbf{W}^1 = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}^T, \mathbf{b}^1 = \begin{bmatrix} -1 & 3 & 1 & 1 \end{bmatrix}^T$$

$$\mathbf{W}^2 = \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix}, \mathbf{b}^2 = [5]$$

2D Poslin network surface and decision boundary





Supervised learning

Training Set

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

Performance Indices

Mean Square Error

$$F(\mathbf{x}) = \frac{1}{QS^M} \sum_{q=1}^{Q} \sum_{i=1}^{S^M} \left(t_{i,q} - a_{i,q}^M\right)^2$$

Cross Entropy

$$F(\mathbf{x}) = -\sum_{q=1}^{Q} \sum_{i=1}^{S^M} t_{i,q} ln \frac{a_{i,q}^M}{t_{i,q}}$$

Approximate performance index

2nd order Taylor series expansion (quadratic)

$$F(\mathbf{x}) \cong F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T\big|_{\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^T\,\nabla^2 F(\mathbf{x})\big|_{\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*)$$

Gradient – Direction of increasing $F(\mathbf{x})$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial F(\mathbf{x})}{\partial x_1} & \frac{\partial F(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial F(\mathbf{x})}{\partial x_n} \end{bmatrix}^T$$

Hessian – Curvature of $F(\mathbf{x})$

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

Optimization of performance

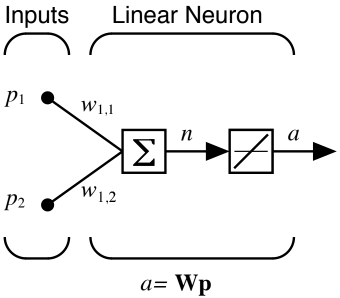
General optimization algorithm

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

- The search direction at iteration k is $\mathbf{p}_k$.
- The learning rate is α_k.
- For small learning rates, the largest reduction in $F(\mathbf{x})$ is obtained by setting $\mathbf{p}_k = -\nabla F(\mathbf{x}_k)$, the negative of the gradient direction. This is called the *steepest descent*, or gradient descent algorithm.

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

Example



$$\left\{\mathbf{p}_1 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix}\right\}, \left\{\mathbf{p}_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} -1 \end{bmatrix}\right\}$$

$$\left\{\mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix} 1 \end{bmatrix}\right\}, \left\{\mathbf{p}_4 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix} 1 \end{bmatrix}\right\}$$

Example performance index

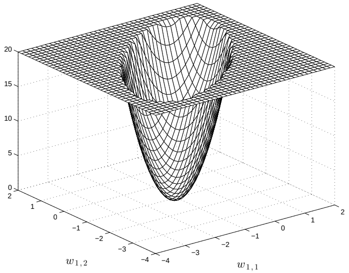
$$\mathbf{U} = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \mathbf{p}_3^T \\ \mathbf{p}_4^T \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} w_{1,1} \\ w_{1,2} \end{bmatrix}$$

$$\begin{aligned} F(\mathbf{x}) &= \sum_{q=1}^{4} (t_q - a_q)^2 = (\mathbf{t} - \mathbf{U}\mathbf{x})^T (\mathbf{t} - \mathbf{U}\mathbf{x}) \\ &= \left(\mathbf{t}^T\mathbf{t} - 2\mathbf{t}^T\mathbf{U}\mathbf{x} + \mathbf{x}^T\mathbf{U}^T\mathbf{U}\mathbf{x}\right) \\ &= c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} \\ \nabla F(\mathbf{x}) &= \mathbf{A}\mathbf{x} + \mathbf{d}, \nabla^2 F(\mathbf{x}) = \mathbf{A} \end{aligned}$$

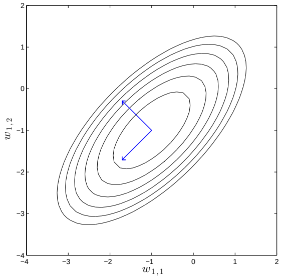
$$c = \mathbf{t}^T\mathbf{t}, \mathbf{d} = -2\mathbf{U}^T\mathbf{t}, \mathbf{A} = 2\mathbf{U}^T\mathbf{U}$$

Example performance surface

$$\mathbf{U} = \begin{bmatrix} -1 & 2 \\ 2 & -1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}, \mathbf{A} = \begin{bmatrix} 12 & -8 \\ -8 & 12 \end{bmatrix}, \mathbf{d} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, c = 4$$



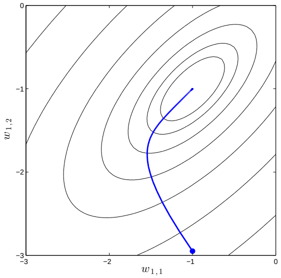
ce surface



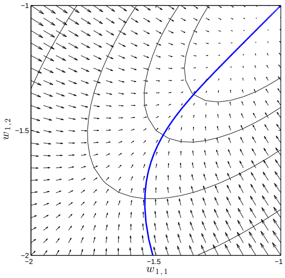
Contour plot

Example steepest descent path

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k) = \mathbf{x}_k - 0.01\,(\mathbf{A}\mathbf{x}_k + \mathbf{d}) = \begin{bmatrix} 0.88 & 0.08 \\ 0.08 & 0.88 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 0.04 \\ 0.04 \end{bmatrix}$$



Steepest descent path



Zoomed path with gradients

Stochastic gradient

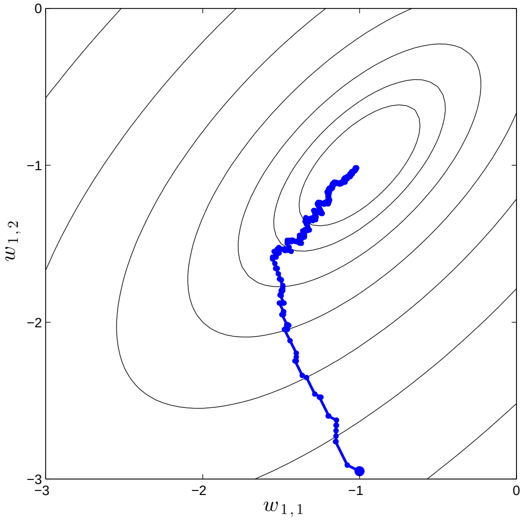
- Standard steepest descent is a batch algorithm, because the entire batch of data is used to compute the gradient.
- If we compute the gradient for one data point, it is an incremental, or stochastic algorithm.
- Mini-batches can also be used, where the algorithm operates on a subset of data – especially useful for large data sets.

$$\hat{F}(\mathbf{x}) = (t_k - a_k)^2$$
$$\nabla\hat{F}(\mathbf{x}) = -2\,(t_k - a_k)\,\nabla a_k = -2e_k\mathbf{p}_k$$

Stochastic gradient algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e_k\mathbf{p}_k$$

Stochastic gradient trajectory



Steepest descent in multilayer networks

- For multilayer networks, the error is not a direct function of weights in the hidden layers.
- To compute the necessary gradients we need to use the chain rule.
- The chain rule is implemented one component at a time (e.g., performance function, transfer function, weight function).

$$\begin{aligned}\frac{\partial \hat{F}(\mathbf{x})}{\partial w_{i,j}^m} &= \frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^m}\frac{\partial n_i^m}{\partial w_{i,j}^m} = \frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^m}\frac{\partial\left(\sum_{l=1}^{S^{m-1}} w_{i,l}^m a_l^{m-1} + b_i^m\right)}{\partial w_{i,j}^m}\\ &= \frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^m} a_j^{m-1}\end{aligned}$$

Module derivatives

Derivative across transfer function

$$\frac{\partial \mathbf{a}^m}{\partial (\mathbf{n}^m)^T} = \dot{\mathbf{F}}^{\mathbf{m}}(\mathbf{n}^m) = \begin{bmatrix} \frac{\partial f_1^m(\mathbf{n}^m)}{\partial n_1^m} & \frac{\partial f_1^m(\mathbf{n}^m)}{\partial n_2^m} & \cdots & \frac{\partial f_1^m(\mathbf{n}^m)}{\partial n_{Sm}^m} \\ \frac{\partial f_2^m(\mathbf{n}^m)}{\partial n_1^m} & \frac{\partial f_2^m(\mathbf{n}^m)}{\partial n_2^m} & \cdots & \frac{\partial f_2^m(\mathbf{n}^m)}{\partial n_{Sm}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{Sm}^m(\mathbf{n}^m)}{\partial n_1^m} & \frac{\partial f_{Sm}^m(\mathbf{n}^m)}{\partial n_2^m} & \cdots & \frac{\partial f_{Sm}^m(\mathbf{n}^m)}{\partial n_{Sm}^m} \end{bmatrix}$$

Derivative across weight function

$$\frac{\partial \mathbf{n}^{m+1}}{\partial (\mathbf{a}^m)^T} = \frac{\partial [\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}]}{\partial (\mathbf{a}^m)^T} = \mathbf{W}^{m+1}$$

Example transfer function derivatives

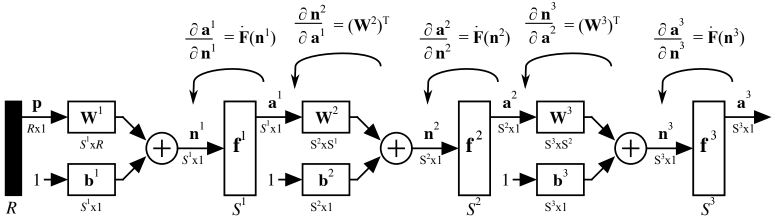
Poslin

$$\dot{\mathbf{F}}^{\mathbf{m}}(\mathbf{n}^m) = \begin{bmatrix} \text{hardlim}(n_1^m) & 0 & \cdots & 0 \\ 0 & \text{hardlim}(n_2^m) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \text{hardlim}(n_{S^m}^m) \end{bmatrix}$$

Softmax

$$\dot{\mathbf{F}}^{\mathbf{m}}(\mathbf{n}^m) = \begin{bmatrix} a_1^m\left(\sum_{i=1}^{S^m} a_i^m - a_1^m\right) & -a_1^m a_2^m & \cdots & -a_1^m a_{S^m}^m \\ -a_2^m a_1^m & a_2^m\left(\sum_{i=1}^{S^m} a_i^m - a_2^m\right) & \cdots & -a_2^m a_{S^m}^m \\ \vdots & \vdots & \ddots & \vdots \\ -a_{S^m}^m a_1^m & -a_{S^m}^m a_2^m & \cdots & a_{S^m}^m\left(\sum_{i=1}^{S^m} a_i^m - a_{S^m}^m\right) \end{bmatrix}$$

Multilayer chain rule (backpropagation)



$\frac{\partial \mathbf{a}^1}{\partial \mathbf{n}^1} = \dot{\mathbf{F}}(\mathbf{n}^1)$
$\frac{\partial \mathbf{n}^2}{\partial \mathbf{a}^1} = (\mathbf{W}^2)^T$
$\frac{\partial \mathbf{a}^2}{\partial \mathbf{n}^2} = \dot{\mathbf{F}}(\mathbf{n}^2)$
$\frac{\partial \mathbf{n}^3}{\partial \mathbf{a}^2} = (\mathbf{W}^3)^T$
$\frac{\partial \mathbf{a}^3}{\partial \mathbf{n}^3} = \dot{\mathbf{F}}(\mathbf{n}^3)$
p
Rx1
R
$\mathbf{W}^1$
S^1xR
1
$\mathbf{b}^1$
S^1x1
$\mathbf{n}^1$
S^1x1
$\mathbf{f}^1$
S^1
$\mathbf{a}^1$
S^1x1
$\mathbf{W}^2$
S^2xS^1
1
$\mathbf{b}^2$
S^2x1
$\mathbf{n}^2$
S^2x1
$\mathbf{f}^2$
S^2
$\mathbf{a}^2$
S^2x1
$\mathbf{W}^3$
S^3xS^2
1
$\mathbf{b}^3$
S^3x1
$\mathbf{n}^3$
S^3x1
$\mathbf{f}^3$
S^3
$\mathbf{a}^3$
S^3x1

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^1} = \frac{\partial \mathbf{a}^1}{\partial \mathbf{n}^1} \times \frac{\partial \mathbf{n}^2}{\partial \mathbf{a}^1} \times \frac{\partial \mathbf{a}^2}{\partial \mathbf{n}^2} \times \frac{\partial \mathbf{n}^3}{\partial \mathbf{a}^2} \times \frac{\partial \mathbf{a}^3}{\partial \mathbf{n}^3} \times \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{a}^3}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^m} = \frac{\partial \mathbf{a}^m}{\partial \mathbf{n}^m} \times \frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{a}^m} \times \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^{m+1}}$$

HPSI Human Performance Systems, Inc.

Initializing backpropagation

For Mean Square Error

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^M} = \sum_{j=1}^{S^M} \frac{\partial \hat{F}(\mathbf{x})}{\partial a_j^M} \frac{\partial a_j^M}{\partial n_i^M} = \frac{1}{S^M} \sum_{j=1}^{S^M} \frac{\partial \left(t_j - a_j^M \right)^2}{\partial a_j^M} \frac{\partial a_j^M}{\partial n_i^M}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial n_i^M} = \frac{-2}{S^M} \sum_{j=1}^{S^M} \left(t_j - a_j^M \right) \frac{\partial a_j^M}{\partial n_i^M}$$

$$= \frac{-2}{S^M} \sum_{j=1}^{S^M} e_j \frac{\partial a_j^M}{\partial n_i^M}$$

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^M} = \frac{\partial (\mathbf{a}^M)^T}{\partial \mathbf{n}^M} \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{a}^M} = \frac{-2}{S^M} \dot{\mathbf{F}}^{\mathbf{M}} \left(\mathbf{n}^m \right) \mathbf{e}$$

Summary of multilayer stochastic gradient

If we define the sensitivity to be

$$\mathbf{s}^m \triangleq \frac{\partial \hat{F}(\mathbf{x})}{\partial \mathbf{n}^m}$$

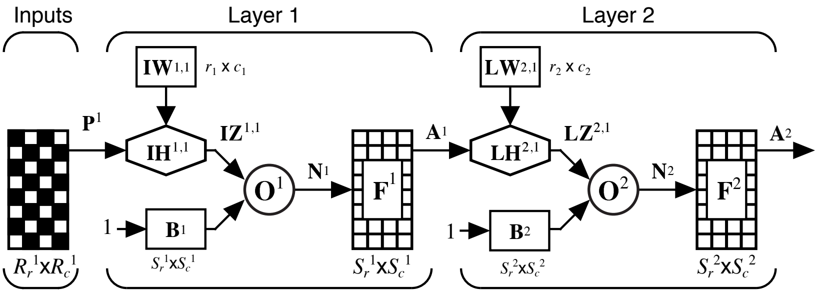
Then the stochastic gradient for mean square error can be computed as

$$\begin{aligned}
\mathbf{s}^M &= \frac{-2}{S^M}\dot{\mathbf{F}}^{\mathbf{M}}\left(\mathbf{n}^m\right)\mathbf{e} \\
\mathbf{s}^m &= \dot{\mathbf{F}}(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \\
\frac{\partial \hat{F}(\mathbf{x})}{\partial w^m_{i,j}} &= s^m_i a^{m-1}_j \\
\frac{\partial \hat{F}(\mathbf{x})}{\partial b^m_i} &= s^m_i
\end{aligned}$$

HPSI
Human Performance Systems, Inc.

Convolution network

- A convolution network is a multilayer feedforward network that has two- or three-dimensional inputs.
- It has weight functions that are not generally viewed as matrix multiplication (or inner product) operations.



2D Convolution

- The principal layer type for convolution networks is the convolution layer.
- Let the input image be represented by the $R_r \times R_c$ matrix $\mathbf{V}$.
- The weight function for this layer performs a convolution operation on the image, using the convolution kernel that is represented by the $r \times c$ matrix $\mathbf{W}$.

$$z_{i,j} = \sum_{k=1}^{r} \sum_{l=1}^{c} w_{k,l} v_{i+k-1,j+l-1}$$

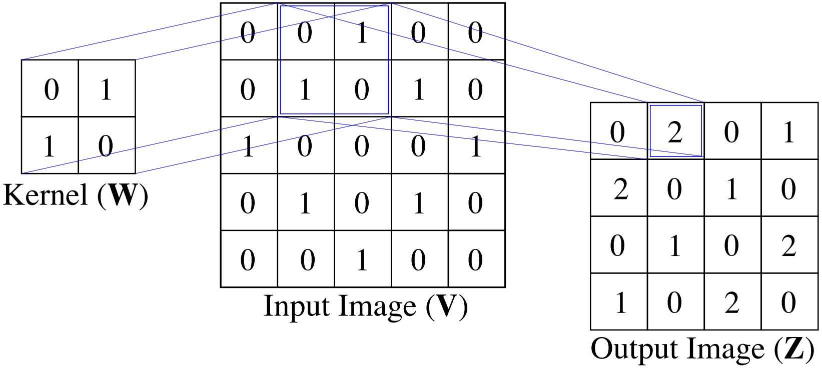
- In matrix form, we will write it as

$$\mathbf{Z} = \mathbf{W} \circledast \mathbf{V}$$

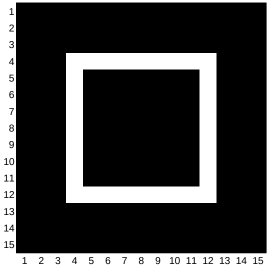
Padding and stride

- Convolution will reduce the number of rows in the image by $r - 1$ and the number of columns by $c - 1$.
- To maintain image size, we can pad the outside of the image with zeros before convolving.
- The width of the zero padding is P^d.
- The output image can be made smaller by taking larger strides, or kernel movements. Normally, the kernel is moved one step at a time when performing the convolution. If the stride is increased to 2, the output image size is reduced by a factor of 2.
- The number of steps for the stride is S^t.

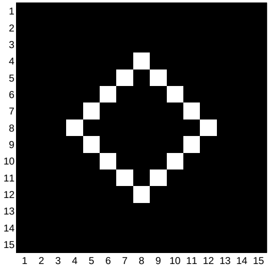
Example convolution



Input Images



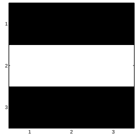
Square



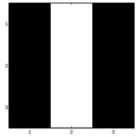
Diamond

HPSI
Human Performance Systems, Inc.

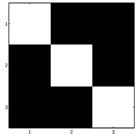
Convolution kernels (filters)



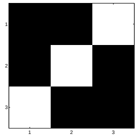
Horizontal kernel



Vertical kernel

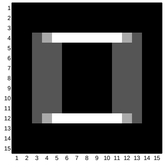


Slash kernel

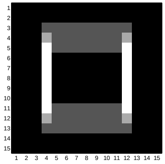


Backslash kernel

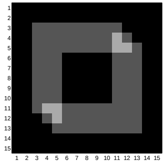
Filtered square



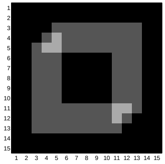
Horizontal filtered square



Vertical filtered square



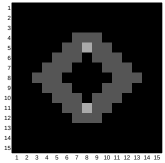
Slash filtered square

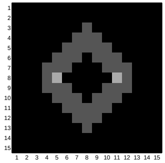


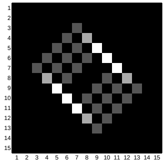
Backslash filtered square

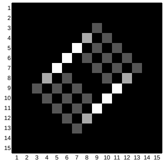
HPSI Human Performance Systems, Inc.

Filtered diamond


Horizontal filtered diamond


Vertical filtered diamond


Slash filtered diamond


Backslash filtered diamond

HPSI Human Performance Systems, Inc.

Pooling

- A pooling (or subsampling) layer often follows a convolution layer and consolidates $r \times c$ elements in the input image to 1 element in the output image.
- The purpose is to reduce the spatial size of the feature map.
- This reduces the number of parameters in the network.

Average pooling

$$z_{i,j} = \left\{ \sum_{k=1}^{r} \sum_{l=1}^{c} v_{r(i-1)+k,c(j-1)+l} \right\} w$$

Matrix format

$$\mathbf{Z} = w \boxplus_{r,c}^{ave} \mathbf{V}$$

Max pooling

- For max pooling, the maximum of the elements in the consolidation window is used, rather than the sum.
- Recent studies have shown max pooling to be a little more effective in some applications than average pooling.

Max pooling

$$z_{i,j} = max\left\{v_{r(i-1)+k,c(j-1)+l} | k = 1, \ldots, r; l = 1, \ldots, c\right\}$$

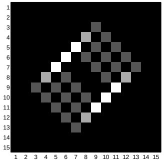
Matrix format

$$\mathbf{Z} = \boxplus_{r,c}^{max}\mathbf{V}$$

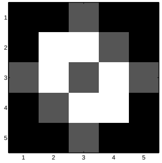
Pooling: choice of stride and size

- As in convolution operations, various strides can be used in pooling operations.
- Normally, the consolidation window is square, and the stride is equal to the window size, so there is no overlap in the consolidations.
- The most common choice is $r = 2$, $c = 2$ and $S^t = 2$.

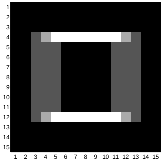
Max pooling (3x3) examples



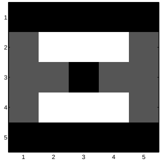
Backslash filtered diamond



Max pooled filtered diamond



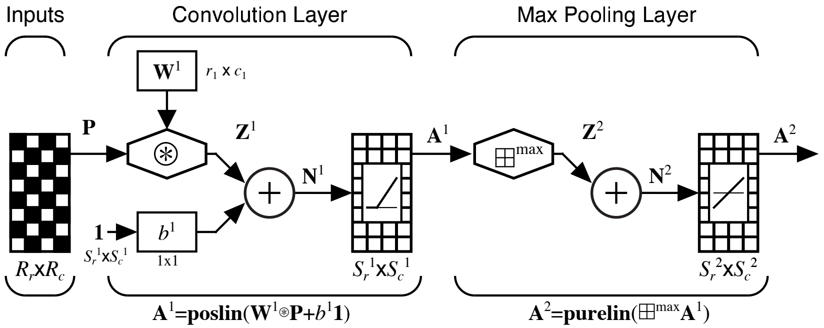
Horizontal filtered square



Max pooled filtered square

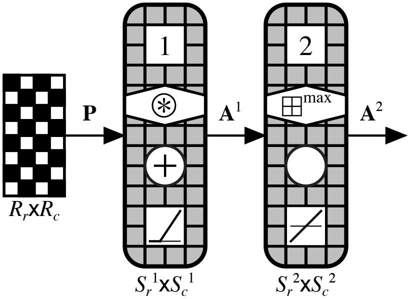
Network diagram notation

The following figure represents the combination of a convolution layer and a pooling layer. Notice that the bias in the convolution layer is a scalar, which is added to each element of $\mathbf{IZ}^{1,1}$.



Simplified notation

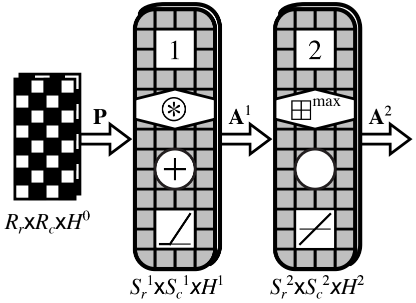
When many layers are involved, it is helpful to have a simplified notation to represent each layer.



Feature maps

- Each convolution kernel (weight) can identify one elemental feature in the input.
- Complex patterns will consist of combinations of many elemental features.
- Multiple kernels can be included in a single convolution layer to extract multiple features.
- LeCun, in his original developement, called the outputs of one kernel operation a feature map (FM).
- This idea can be extended to the input image. For example, a color image consists of red, green and blue planes.
- A convolution layer then takes a set of feature maps as an input, and produces another set of feature maps as an output.

Network diagram with feature maps



1
2
max
P
$\mathbf{A}^1$
$\mathbf{A}^2$
$R_r \mathsf{x} R_c \mathsf{x} H^0$
$S_r^1 \mathsf{x} S_c^1 \mathsf{x} H^1$
$S_r^2 \mathsf{x} S_c^2 \mathsf{x} H^2$

Connection matrix

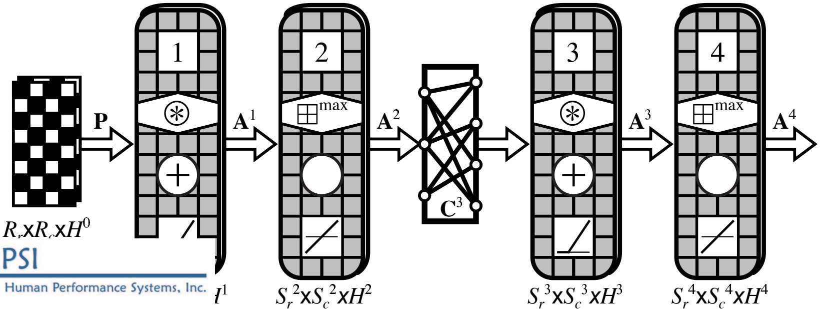
- Assume that there are H^m FMs in Layer m, and H^{m+1} FMs in layer $m+1$.
- If each FM in Layer m is connected to every FM in Layer $m+1$, then there would be $H^m \times H^{m+1}$ convolution kernels in Layer $m+1$.
- We can reduce the number of kernels by using only a subset of the possible connections.
- The connection matrix between Layer m and Layer $m+1$ will be denoted $\mathbf{C}^{m+1}$.
- Element $c_{i,j}^{m+1}$ will equal 1 when FM j in Layer m is connected to FM i in Layer $m+1$. Otherwise, it will equal 0.

Notation with connection matrix

$$z_{i,j}^{m,h} = \sum_{l \in C^{m,h}} \sum_{u=1}^{r^m} \sum_{v=1}^{c^m} w_{u,v}^{m,(h,l)} a_{i+u-1,j+v-1}^{m-1,l}$$

$C^{m,h}$ is the set of indices of FMs in layer $m-1$ that connect to FM h in Layer m (from row h of $\mathbf{C}^m$).

$$\mathbf{Z}^{m,h} = \sum_{l \in C^{m,h}} \mathbf{W}^{m,(h,l)} \circledast \mathbf{A}^{m-1,l}$$



HPSI Human Performance Systems, Inc.

Uses of connection matrices

- Connection matrices are generally located between a pooling layer and the following convolution layer.
- If no connection block appears between a pooling layer and the following convolution layer, the FMs are fully connected.
- The number of FMs in a convolution layer and its following pooling layer are equal, and feature map *h* in the convolution layer is connected only to feature map *h* in the following pooling layer. No connection block is shown.

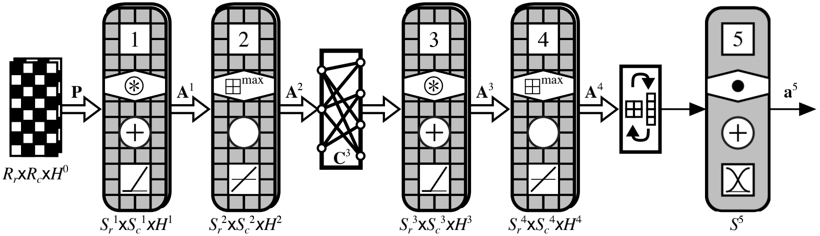
Conversion from matrix to vector

- In addition to the connection block, there is one other block that can appear between layers of a convolution network.
- It converts the output of a layer from a set of FMs to a single vector.
- It stacks the columns of the FMs on top of each other, starting from the first FM to the last. ($\mathbf{a}_i$ indicates the i^{th} column of matrix $\mathbf{A}$.)

$$vec(\mathbf{A}^m) = \left[\left(\mathbf{a}_1^{m,1}\right)^T \left(\mathbf{a}_2^{m,1}\right)^T \cdots \left(\mathbf{a}_{S_c^m}^{m,1}\right)^T \left(\mathbf{a}_1^{m,2}\right)^T \cdots \left(\mathbf{a}_{S_c^m}^{m,H^m}\right)^T \right]^T$$

Use of matrix to vector conversion

The matrix to vector conversion is normally used before the final layer of a convolution network, which is a standard dot product (matrix multiplication) layer. (The same conversion block can indicate vector to matrix conversion, if needed.)



Derivative definitions

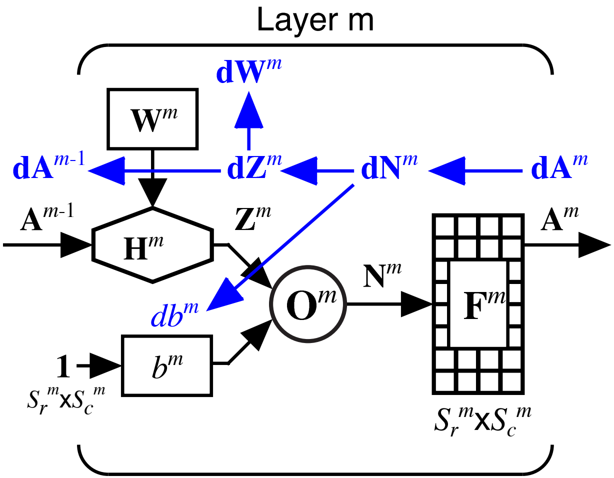
In order to compute the gradient of the performance with respect to the weights and biases, we need to perform a backpropagation operation. Because the net input is a matrix, we will use a different notation for sensitivity – **dN**. Other derivatives will be defined as:

$$\mathbf{dN} \equiv \frac{\partial \hat{F}}{\partial \mathbf{N}}, \mathbf{dA} \equiv \frac{\partial \hat{F}}{\partial \mathbf{A}}, \mathbf{dZ} \equiv \frac{\partial \hat{F}}{\partial \mathbf{Z}}, \mathbf{dW} \equiv \frac{\partial \hat{F}}{\partial \mathbf{W}}, \mathbf{db} \equiv \frac{\partial \hat{F}}{\partial \mathbf{b}}$$

As we backpropagate across Layer *m*, we will be performing the following operations.

$$\mathbf{dA}^m \rightarrow \mathbf{dN}^m \rightarrow \mathbf{dZ}^m \rightarrow \mathbf{dA}^{m-1}$$

Backpropagating across a layer



Backpropagating across the transfer function

$$dn_{i,j}^{m,h} \equiv \frac{\partial \hat{F}}{\partial n_{i,j}^{m,h}} = \frac{\partial a_{i,j}^{m,h}}{\partial n_{i,j}^{m,h}} \times \frac{\partial \hat{F}}{\partial a_{i,j}^{m,h}}$$

$$\frac{\partial \hat{F}}{\partial a_{i,j}^{m,h}} = da_{i,j}^{m,h}$$

$$dn_{i,j}^{m,h} = \dot{f}^{m,h}(n_{i,j}^{m,h}) \times da_{i,j}^{m,h}$$

Matrix form (**dN**)

$$\mathbf{dN}^{m,h} = \dot{\mathbf{F}}^{m,h} \circ \mathbf{dA}^{m,h}$$

Here $\circ$ is the Hadamard product, which is an element by element matrix multiplication.

Backpropagating across the summation net input function

Scalar form (dz)

$$dz_{i,j}^{m,h} = dn_{i,j}^{m,h}$$

Matrix form (**dZ**)

$$\mathbf{dZ}^{m,h} = \mathbf{dN}^{m,h}$$

Scalar form (*db*)

$$db^{m,h} = \frac{\partial \hat{F}}{\partial b^{m,h}} = \frac{\partial \hat{F}}{\partial n_{i,j}^{m,h}} \times \frac{\partial n_{i,j}^{m,h}}{\partial b^{m,h}}$$

$$db^{m,h} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} dn_{i,j}^{m,h}$$

Matrix form (*db*)

$$db^{m,h} = (\boxplus_{S_r^m, S_c^m} \mathbf{dN}^{m,h}))$$

Backpropagating across the convolution weight function

Scalar form (da)

$$da_{i,j}^{m-1,l} = \frac{\partial \hat{F}}{\partial a_{i,j}^{m,l}} = \frac{\partial \hat{F}}{\partial z_{u,v}^{m,h}} \times \frac{\partial z_{u,v}^{m,h}}{\partial a_{i,j}^{m-1,l}}$$

$$da_{i,j}^{m-1,l} = \sum_{h \epsilon C_b^{m,l}} \sum_{u=1}^{S_r^m} \sum_{v=1}^{S_c^m} dz_{i,j}^{m,h} w_{i-u+1,j-v+1}^{m,(h,l)}$$

$C_b^{m,l}$ – FMs in layer m where lth FM in layer $m-1$ connects.

Matrix form (**dA**)

$$\mathbf{dA}^{m-1,l} = \sum_{h \epsilon C_b^{m,l}} (rot180(\mathbf{W}^{m,(h,l)})) \star (\mathbf{dZ}^{m,h})$$

Here rot180 means matrix is rotated by 180 degrees.

Gradient for convolution weight

Scalar form (dw)

$$dw_{u,v}^{m,h,l} = \frac{\partial \hat{F}}{\partial w_{u,v}^{m,(h,l)}} = \frac{\partial \hat{F}}{\partial z_{i,j}^{m,h}} \times \frac{\partial z_{i,j}^{m,h}}{\partial w_{u,v}^{m,(h,l)}}$$

$$dW_{u,v}^{m,h,l} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} dZ_{i,j}^{m,h} v_{u+i-1,v+j-1}^{m,l}$$

here l is feature map in layer $m-1$.

Matrix form (**dW**)

$$\mathbf{dW}^{m,h,l} = \mathbf{dZ}^{m,h} \star \mathbf{V}^{m,l}$$

Backpropagating across average pooling

Scalar form (da)

$$da^{m-1,h}_{r^{m-1}(i-1)+k,c^{m-1}(j-1)+l} = \frac{\partial \hat{F}}{\partial a^{m-1,h}_{i,j}} = \frac{\partial \hat{F}}{\partial z^{m,h}_{i,j}} \times \frac{\partial z^{m,h}_{i,j}}{\partial a^{m-1,h}_{i,j}}$$

$$= dZ^{m,h}_{i,j} \times w^{m,h}$$

For k=1 to r^m and l=1 to c^m.

Matrix form (**dA**)

$$\mathbf{dA}^{m-1,h} = \left(w^{m,h}\right) \boxtimes^{ave}_{r^m,c^m} \left(\mathbf{dZ}^{m,h}\right)$$

$\boxtimes^{ave}_{j,k}\mathbf{A}$ takes each element of the matrix **A** and expands to j rows and k columns (reverse of $\boxplus^{ave}_{j,k}\mathbf{A}$).

HPSI Human Performance Systems, Inc.

Gradient for average pooling weight

Scalar form (dw)

$$dw^{m,h} = \frac{\partial \hat{F}}{\partial w^{m,h}} = \frac{\partial \hat{F}}{\partial z_{i,j}^{m,h}} \times \frac{\partial z_{i,j}^{m,h}}{\partial w^{m,h}}$$

$$dw^{m,h} = \sum_{i=1}^{S_r^m} \sum_{j=1}^{S_c^m} dz_{i,j}^{m,h} \left\{ \sum_{k=1}^{r^m} \sum_{l=1}^{c^m} a_{r^m(i-1)+k,c^m(j-1)+l}^{m-1,h} \right\}$$

Matrix form (dw)

$$dw^{m,h} = \boxplus_{S_r^m,S_c^m}^{ave}(\mathbf{dZ}^{m,h} \circ (\boxplus_{r^m,c^m}^{ave}\mathbf{A}^{m-1,h}))$$

First it performs a reduction operation that produces a matrix of size S_r^m by S_c^m, and then, after a Hadamard matrix multiplication, it performs another reduction to produce the scalar gradient.