Machine Learning
Baruch College
Lecture 1

Miguel A. Castro

Today We’ll Cover:

- Why Machine Learning?
- Class Specifics (Syllabus etc.)
- Brief Overview: Machine Learning Paradigms
- Neural Networks

Why Machine Learning?

- "*If data had mass, the earth would be a black hole*." (S. Marsland)
  - Banks, hospitals, retailers, laboratories, … store Terabytes of data daily.
  - 7 Billion shares traded in U.S. markets, 60% by HF algo traders.
  - HF data storage needs about double every year.
- Computational Power
  - Fast data storage is cheap and getting cheaper every year (< 10¢/Gb).
  - Moore's Law: Computer Performance doubles every 1.5 years.
- Information Extraction
  - Data is a competitive asset.
  - Extracting information from data is a competitive necessity.
- Machine Learning
  - Has benefited from all this.
  - Has advanced tremendously from increased computational power.
  - Well-suited for tackling large data sets.

Class Specifics

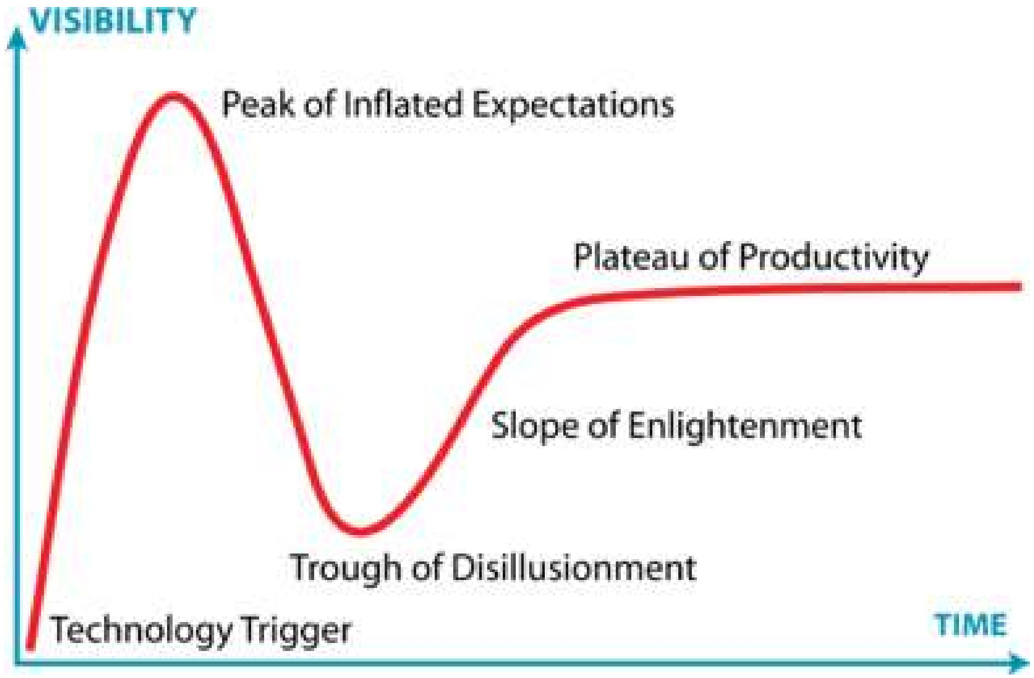
- Texts (on reserve):
  - ***Introduction to Machine Learning*, 2nd Ed. (Alpaydin)**
  - *The Elements of Statistical Learning*, 2nd Ed. (Hastie, Tibshirani & Friedman)
  - *Machine Learning, An Algorithmic Perspective* (Marsland)
  - *Machine Learning* (Mitchell)
- 4 Homework Assignments (30%)
  - Due September 8, 15, 29, October 6
  - In PDF Format by Email
- In-Class Midterm Quiz, September 22 (30%)
- Final Group Project (40%)
  - Groups can be 1-4 people
  - Programming language (if needed) is your choice
  - Project Proposal due September 22 (same day as the Midterm)
  - Report due October 13 (last day)
  - In PDF Format by Email
  - Short (10-15 minute) presentation (one per group)

Machine Learning in Context

- A Branch of Artificial Intelligence (AI). History…
  - “Back in the Day” AI had grandiose expectations (intelligent machines would surpass humans in every way).
  - AI based on logical inference and rules (“crisp” logic).
  - Disillusionment: Turing Test has yet to be passed with flying colors.
  - Realization: The world is too messy (incomplete, faulty, noisy Information).
  - Focus on less lofty but useful applications (robotics, etc.).

Technology Adoption Curve for AI

- Technology Adoption Curve (Gartner Group):



New Paradigm for AI

- Biological brains are not “logical” but heuristic.
- Massive $(\sim 10^{15})$ arrays of simple processors (neurons).
- Better able to handle uncertainty, noise, incomplete information than logical paradigms.
- Enter Artificial Neural Networks …

  - Thought to emulate biological brains (overhyped).

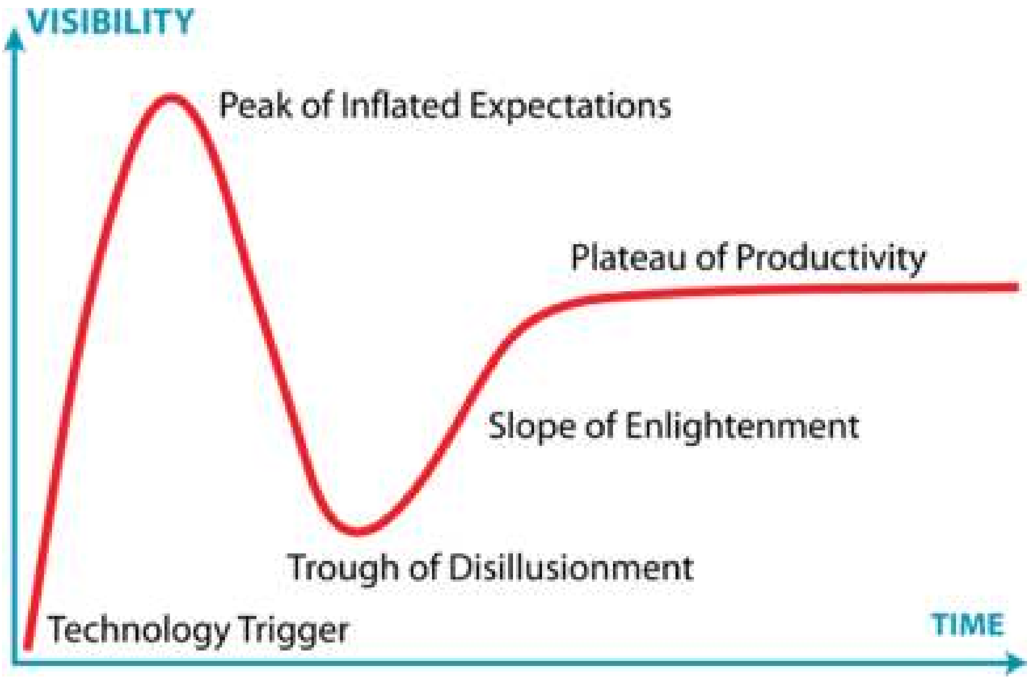
  - Can learn adaptively by example.

  - Can handle uncertainty, noise, incomplete information.

  - Disillusionment: Found to be essentially statistical models.

  - Very useful tools—Focus on applications.

Technology Adoption Curve for Neural Nets



- Used in medical diagnosis
- Fraud detection
- System Control
- Time series forecasting
- Credit scoring
- Pattern recognition, …

Machine Learning

- Is, in some sense, part of AI, but multidisciplinary.
- Draws elements from Statistics;
- Computer Science;
- Mathematics;
- Engineering;
- Neuroscience (both ways).
- Definition: Algorithms to process and extract actionable information from large data sets, often in an automated and adaptive way, to perform useful tasks.

ML Paradigms

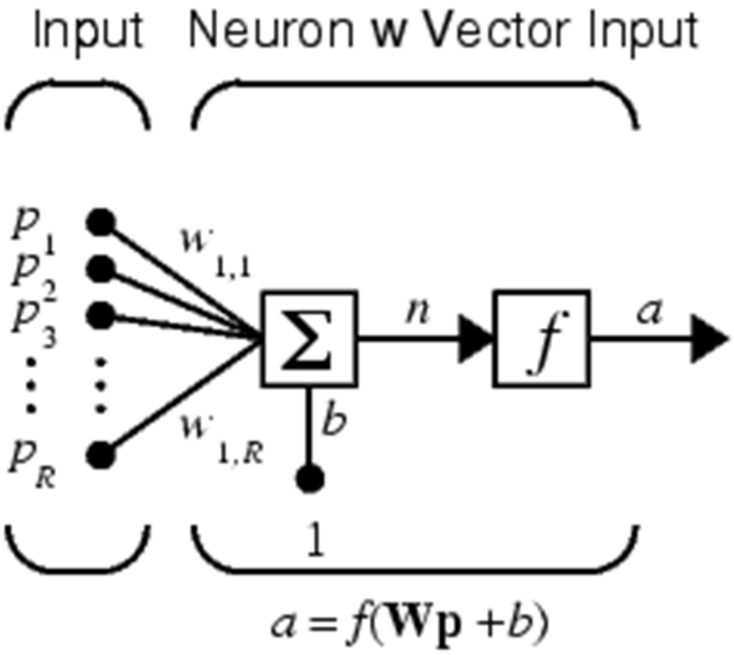
- Supervised vs. Unsupervised
- Supervised learning:
  - “Learning by example” (*i.e.*, Response or Target given)
  - Classification
  - Function approximation
  - Time series prediction
- Unsupervised Learning (Self-Organized Learning):
  - Only Input Patterns, no Response or Target given
  - Clustering
  - Association rules

Artificial Neural Networks

- Archetype of Machine Learning.
- Can perform many ML tasks (classification, function approximation, etc…).
- Illustrate many techniques used in ML.
- They are powerful and have wide applicability.
- A great place to start for us.

The Neural Network Analogy of the Biological Brain

- Basic unit is the neuron.
- Aggregates input patterns (stimuli) and responds via Activation Function.
- Learning is achieved through synaptic strength (weights).
- Massively Parallel.



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

R = Number of input patterns

w = Weight

b = "Bias"

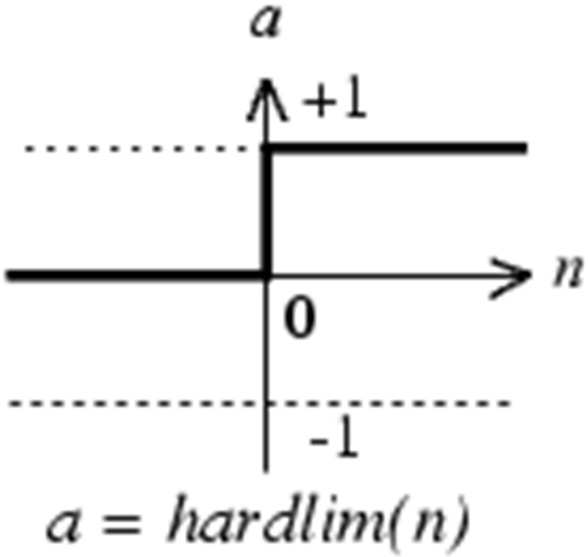
f = Activation function

n = Input to activation

a = Neuron actual output

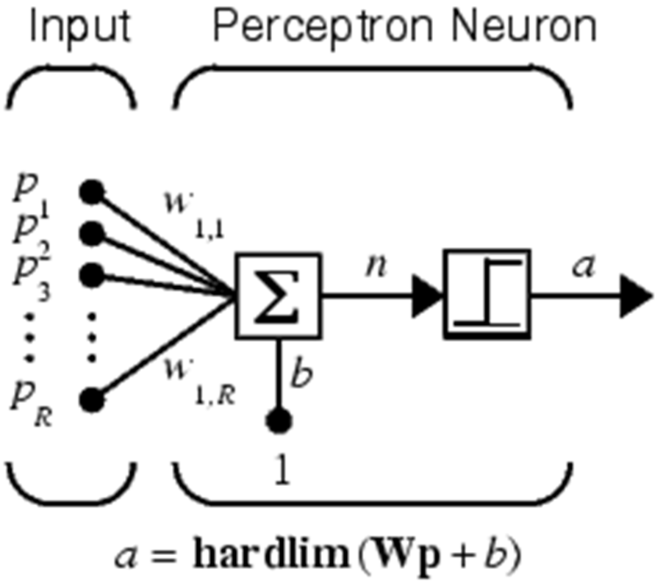
Threshold Activation

- Threshold or “Hard Limit” Activation Function; neuron “fires” when sum of inputs exceeds a threshold.



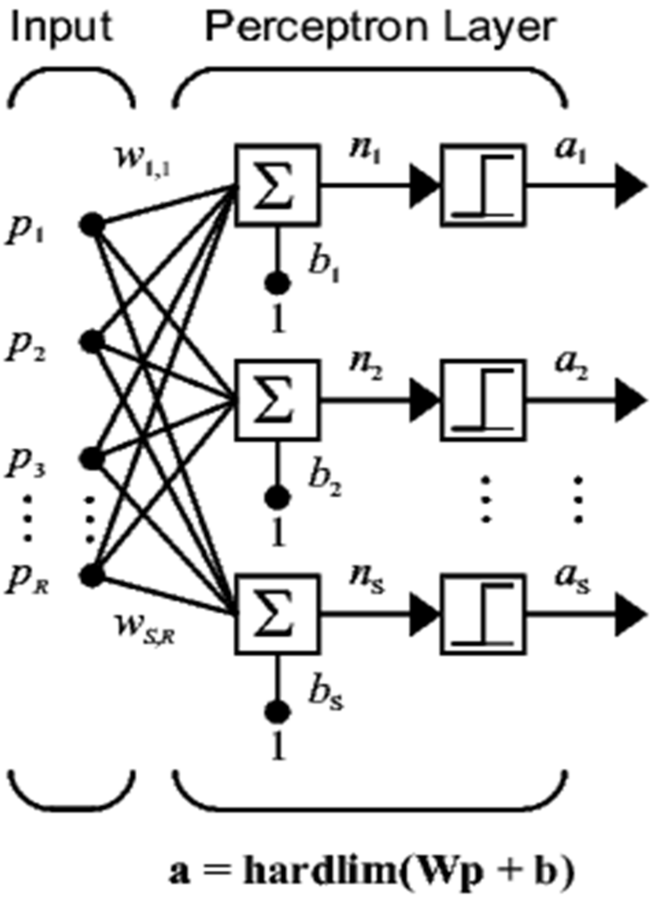
The Perceptron

- Perceptron Neuron (or Node) with Hard Limit activation:



Perceptron Architecture

- High degree of connectivity and parallelism
- (Starting to look like a simplified biological brain.)



R inputs

S outputs

$\mathbf{W}$ is an SxR matrix of weights

$\mathbf{b}$ is an S vector of biases

Now Need Learning Rules

- A ***Learning Rule*** (or training algorithm) is a process for modifying the weights and biases of the network so it can perform a useful task.
- In *Supervised Learning* the Training Set contains targets

$$\{\boldsymbol{p}_1, \boldsymbol{t}_1\}, \{\boldsymbol{p}_2, \boldsymbol{t}_2\}, \{\boldsymbol{p}_3, \boldsymbol{t}_3\}, \dots, \{\boldsymbol{p}_Q, \boldsymbol{t}_Q\}.$$

- In *Unsupervised Learning* the weights and biases are modified in response to input patterns only

$$\{\boldsymbol{p}_1\}, \{\boldsymbol{p}_2\}, \{\boldsymbol{p}_3\}, \dots, \{\boldsymbol{p}_Q\}.$$

- Quantities in brackets are called *Exemplars* (pattern-target or input-output pairs).

Perceptron Learning Rule

- Objective: Minimize the error between actual response and target (over all exemplars):

$$Minimize \quad \boldsymbol{e} = \boldsymbol{t} - \boldsymbol{a}.$$

- Let's look at this closely; there are 3 possibilities for the Perceptron neuron:

1. An input is presented and the actual output is correct so that $\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{a} = \mathbf{0}$ and the weights are unaltered: $\Delta\boldsymbol{w} = \mathbf{0}$.
2. Neuron output is 0 when it should've been 1 ($\boldsymbol{e}$ = 1), and the weights incremented by the input vector, which increases the likelihood that the output will be 1 (**exercise**).
3. Neuron output is 1 when it should've been 0 ($\boldsymbol{e}$ = -1), weights decremented by the input vector.

Perceptron Learning Rule

- Start with an arbitrary set of weights and:
  1. If $\boldsymbol{e} = 0$, $\Delta\mathbf{w} \leftarrow 0$
  2. If $\boldsymbol{e} = 1$, $\Delta\mathbf{w} \leftarrow \mathbf{p}^T$
  3. If $\boldsymbol{e} = -1$, $\Delta\mathbf{w} \leftarrow -\mathbf{p}^T$
- More succintly:

$$\Delta\mathbf{w} \leftarrow \boldsymbol{e}\,\mathbf{p}^T$$

$$\Delta b \leftarrow \boldsymbol{e}$$

- With obvious extensions for multiple neurons.

Perceptron Learning Rule (again)

1. Start with arbitrary weights and biases
2. Update weights and biases according to:

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

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

3. Stop when $\boldsymbol{e}$ is small enough.

- Training is done by incrementing weights for each Exemplar according to the learning rule.
- A Training update using the entire training set is called an _*Epoch*_.

Does it Work?

- Try logic functions like AND, OR:

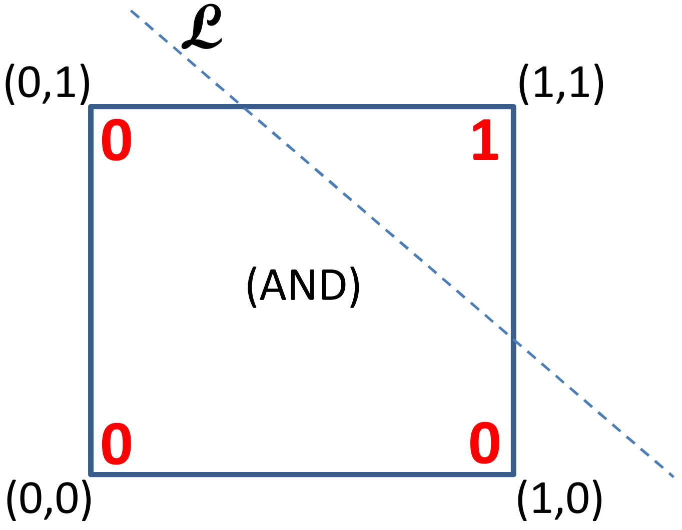
AND		
Input Pattern		**Target**
0	0	0
0	1	0
1	0	0
1	1	1

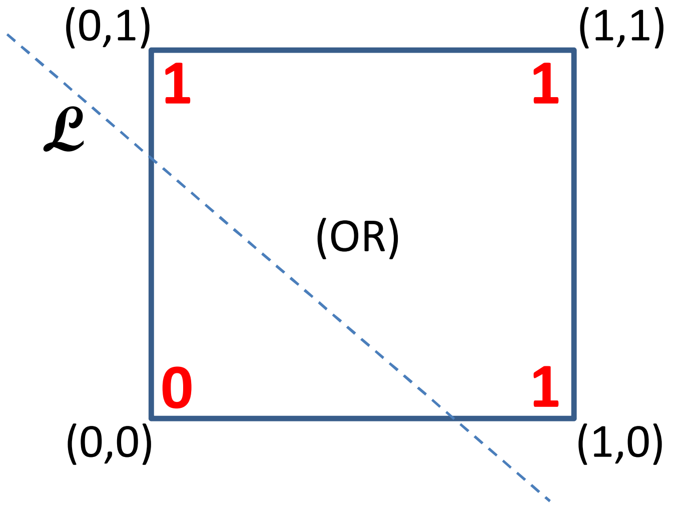
OR		
Input Pattern		**Target**
0	0	0
0	1	1
1	0	1
1	1	1

…

Linear Separability

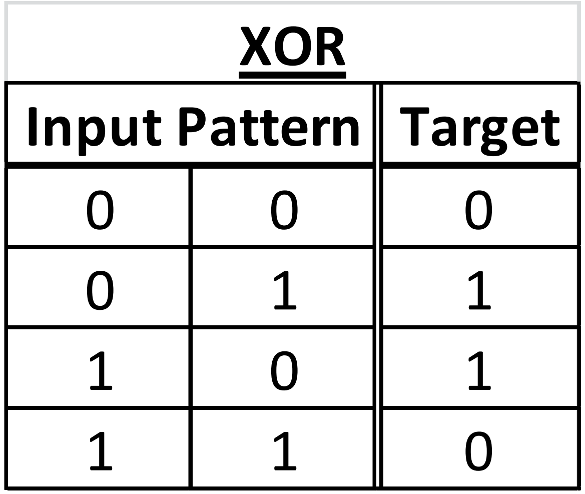
- Perceptrons work perfectly with AND, OR functions
- And with all *Linearly Separable* problems.
- The Hard Limit Activation function allows Perceptron Neuron to "classify" input vectors by dividing the input space into two regions.
- Decision Boundary Line, $\mathcal{L}$, defined by $\mathbf{Wp} + \mathbf{b} = 0$ (Hyperplane in higher dimensions). Output will depend on which side of this line (or hyperplane) the input lies.





Does it Work for Other Classes of Problems?

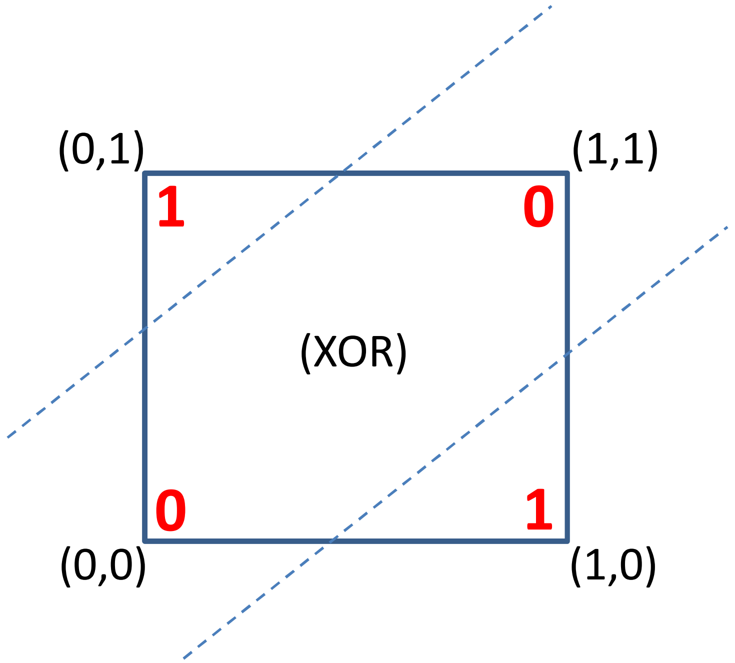
- Exclusive Or (XOR):



XOR		
Input Pattern		Target
0	0	0
0	1	1
1	0	1
1	1	0

…

Linearly Non-Separable



- Perceptron flounders here…
- XOR failure was discovered by Minsky & Papert in 1969, which stagnated Neural Net research for a while.

Can We Fix This?

- Yes…
- Note that:

x XOR y = (x OR y) AND [NOT(x AND y)]

1. Train an OR Perceptron.
2. Train a NOT-AND Perceptron (guaranteed Linearly Separable – why?)
3. Train an AND Perceptron using the outputs of the previous two Perceptrons (guaranteed LS – why?)

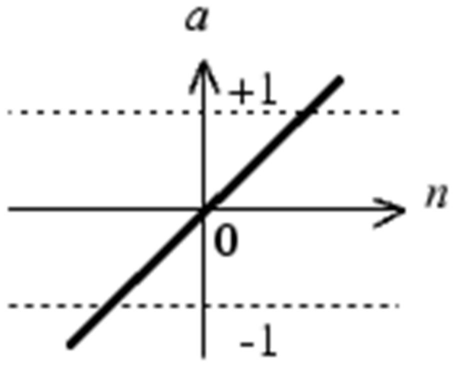
- This ***cascading*** or ***layering*** of Perceptrons is known as a Multi-Layer Perceptron (MLP).
- Requires one extra layer of neurons called the ***Hidden Layer***
- This works, but it’s not very useful as is!

Linearly Non-Separable Problems

- The XOR problem is an example of a broader class of problems: Classify all edges of a hypercube into two classes (arbitrarily located).
- The fix proposed above is not useful because it doesn’t give us an automated (or even concise) way of solving this combinatorially explosive problem by assembling single Perceptrons. We need a new learning rule.
- Things get even more interesting when we have more than two classes.

Need a New Learning Rule

- Perceptron learning rule is “coarse-grained” in that the error (and hence the update rule) is quantized due to the Hard-Limit Activation.
- Idea: Replace Hard-Limit Activation with a smooth activation.
- E.g.: Linear Activation:

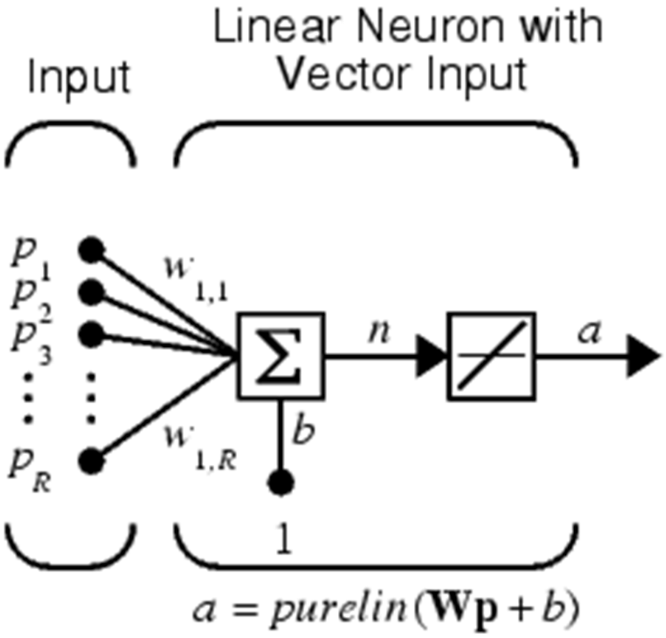


$a = purelin(n)$

ADALINEs

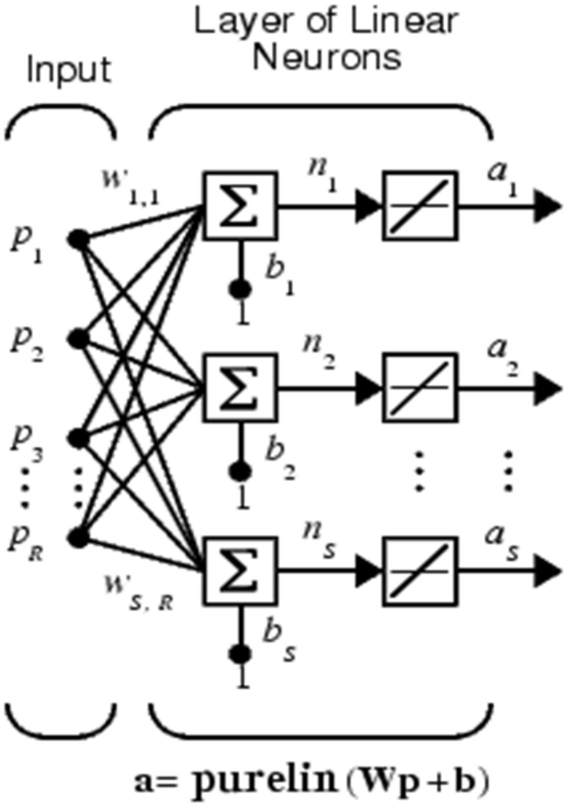
- Adaptive Linear Elements—Perceptrons with Linear Activations:

ADALINE


Input
Linear Neuron with Vector Input
p_1
p_2
p_3
p_R
$w_{1,1}$
$w_{1,R}$
Σ
n
a
b
1
$a = purelin(\mathbf{W}\mathbf{p} + b)$

- Like the Perceptron, also has a Decision Boundary at the line (hyperplane) $\mathbf{Wp} + \mathbf{b} = 0$, but now output is continuous.

ADALINE Architecture



R inputs

S outputs

$\mathbf{W}$ is an SxR matrix of weights

$\mathbf{b}$ is an S vector of biases

Continuous Activations

- Continuous (differentiable) activations allow for error (cost) functions that are continuous (differentiable) in the weights.
- This leads to analytically tractable update (learning) rules; *e.g.*: Gradient Descent.

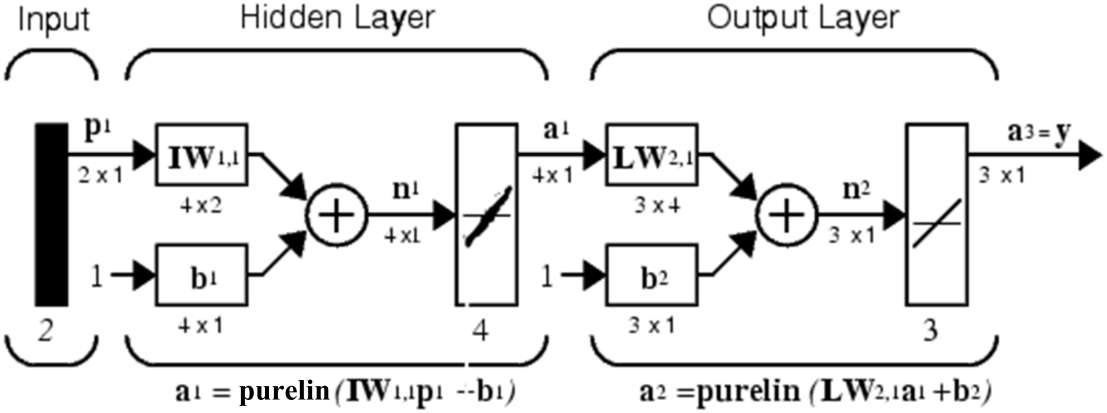
Delta Learning Rule

- Weight update at kth iteration is:
$$\Delta \boldsymbol{W_k} = -\eta \nabla_w e_k$$
(where η is the learning rate).
- For a quadratic error, this has a global minimum and the procedure converges (eventually), and is known as the *Delta Learning Rule*, or the *LMS* Algorithm.
- For linear activations, the update rule is:
$$\Delta \boldsymbol{W_k} = \eta \boldsymbol{e}\boldsymbol{p}^{\boldsymbol{T}},$$
which looks very similar to the Perceptron case.

MADALINEs

- Although easier to train, ADALINEs can only tackle Linearly Separable cases as with the Perceptron case.
- To generalize and be able to tackle Linearly Non-Separable cases, must cascade to form Multiple ADALINEs or MADALINEs with a Hidden Layer:

Layer Diagram



MADALINEs Learning Rule

- One can generalize the single ADALINE learning rule (the Delta Learning Rule) to MADALINEs having one or more hidden layers by repeated applications of the chain rule of differentiation by propagating the output error backwards **(exercise).**
- We have hinted that to solve more general problems one needs “hidden” layers. This was proved in 1986 in a seminal paper…

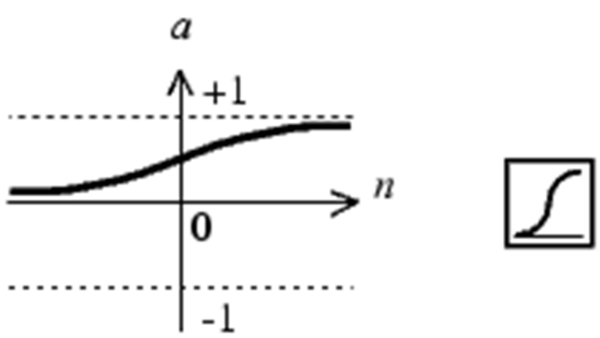
Universal Approximators

- It wasn’t until 1989 that Hornik, Stinchcombe, and White, in their seminal paper (*), proved that a class of Neural Networks was a Universal Function Approximator.
- This means that they can approximate a mapping to an arbitrary degree of accuracy.

(*) Hornik, K., Stinchcombe, M., and White, H. (1989). "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, **2**(5), 359-366.

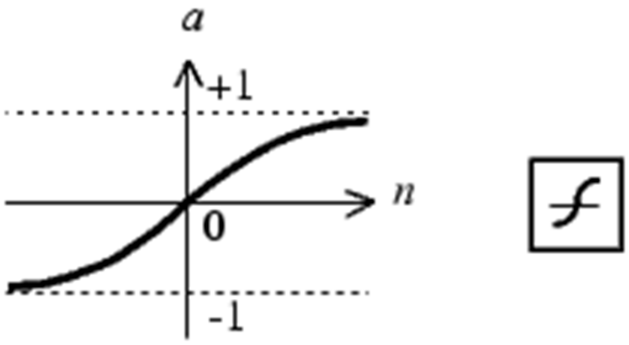
Feedforward Neural Networks

- Multilayer Feedforward Neural Networks are Multilayer Perceptrons with *non-linear* continuous Activations.
- Typically *Sigmoidal* Activation functions; *e.g. Logistic* (LogSig) or *Hyperbolic Tangent* (TanSig):



$a = logsig(n)$

Log-Sigmoid Transfer Function



$a = tansig(n)$

Tan-Sigmoid Transfer Function

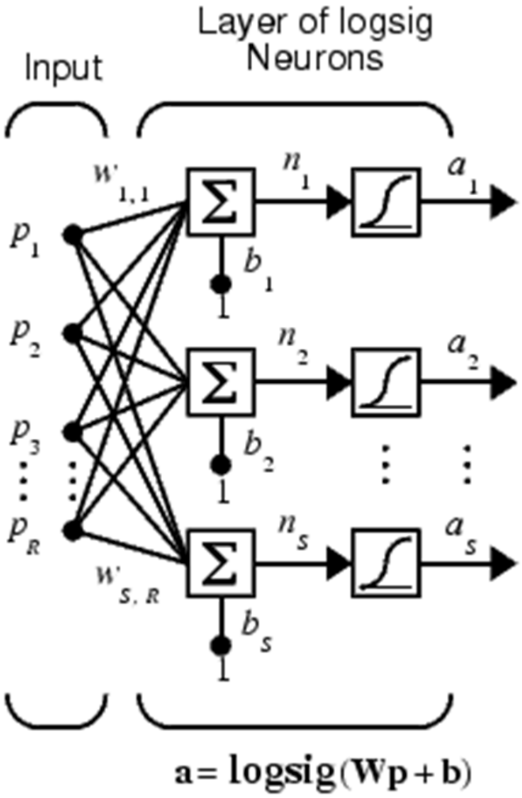
Non-Linear Activation Functions

- Logistic: $a(x) = \frac{1}{1+e^{-x}}$, (0,1)

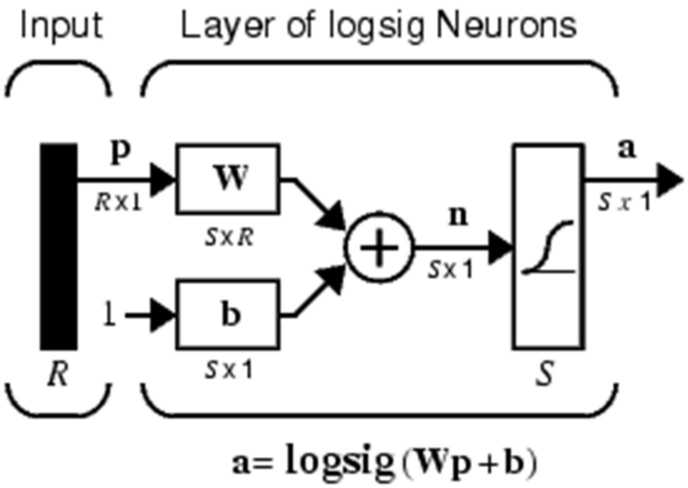
- Tanh: $a(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, (-1,1)

- Modified Logistic: $a(x) = \frac{1-e^{-x}}{1+e^{-x}}$, (-1,1).

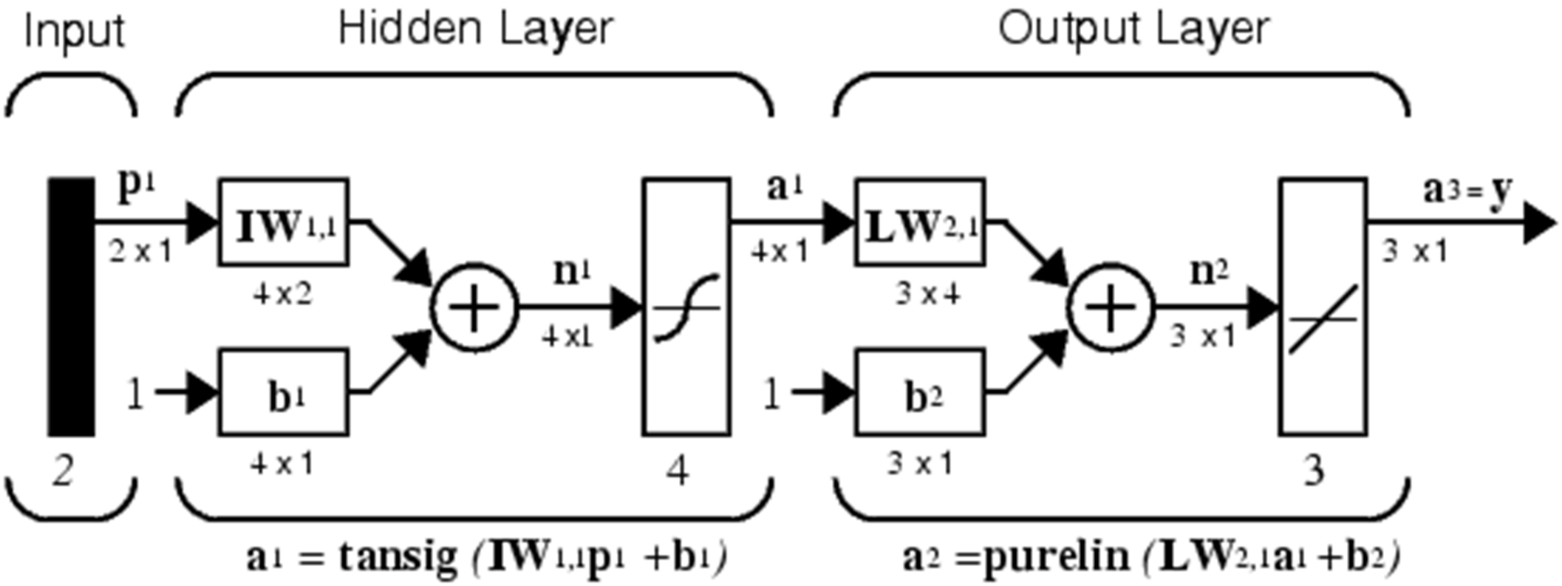
Feedforward NN Layer Architecture



Layer Diagram



Feedforward NN Architecture



- Single Hidden Layer with Tanh Activation.
- Linear Activation Outputs.
- This is the Workhorse Architecture!
- Can solve most problems with it.

Backpropagation of Errors

- In 1986 Rumelhart, Winton, and Williams popularized the general-purpose learning algorithm for Multilayer Feedforward Networks, known as *Backpropagation*.
- This led to a resurgence in the field of Neural Networks (along with increased computational power).
- The idea is to minimize the error by propagating weight updates backwards from the output layer through the input layer applying the chain rule of differentiation repeatedly.

Backpropagation

- Define the error as the squared difference between target and output:

$$E = \frac{1}{2}(\boldsymbol{t} - \boldsymbol{y})^2$$

- Perform a gradient descent by starting at the output and propagating the error back to the input weights using the chain rule:

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial a_h} \frac{\partial a_h}{\partial w_{hj}}.$$

- Update happens at every Exemplar.