Data Science: Machine Learning (MTH 9899) Baruch College Lecture 1

Miguel A. Castro

Lecture 1:

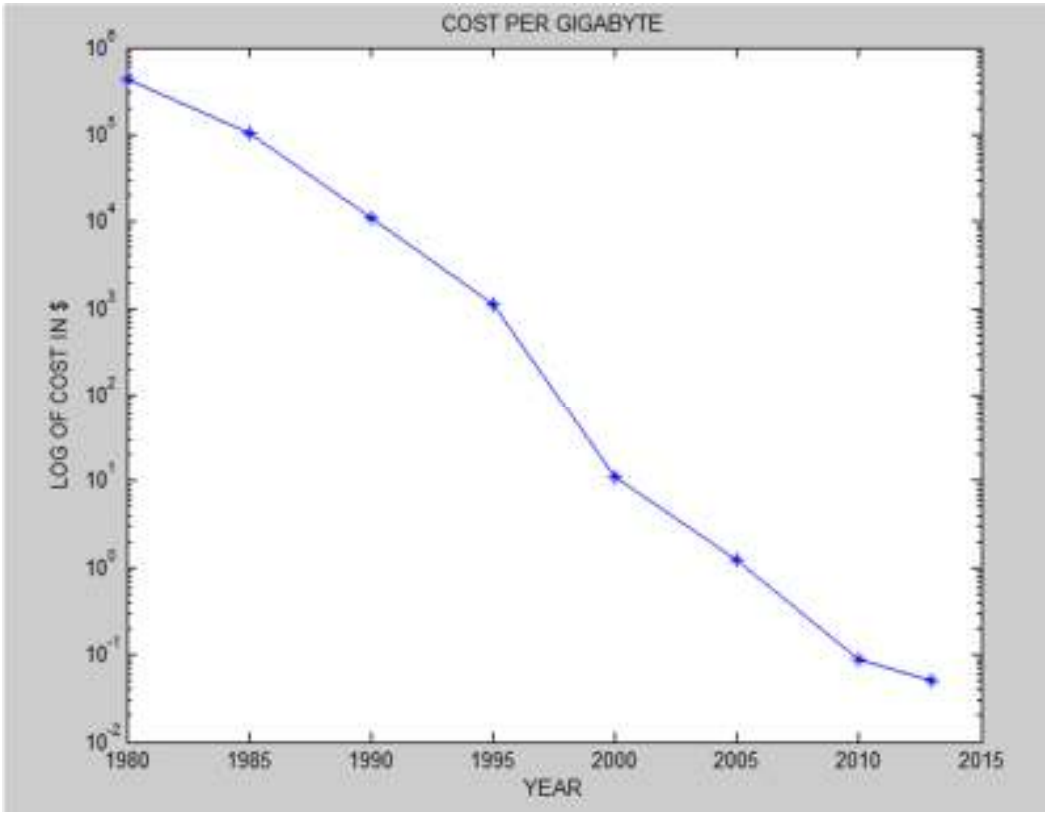
- Why Study Machine Learning?
- Class Specifics (Deliverables etc.)
- Brief Overview: Machine Learning Paradigms
- Start right away with Neural Networks

Why Machine Learning?

- In a nutshell: Because of the coalescence of Big Data and Big Computing.
- Big Data = Today's "Big Buzz" and it's no mystery. Here are some quotes:
  - "From the dawn of civilization until 2003, humankind generated five exabytes of data. Now we produce five exabytes ($1\ exabyte = \ 2^{60} \approx 10^{18}\ bytes$) ***every two days***… and the pace is ***accelerating***." – Eric Schmidt, Executive Chairman, Google. (This is old info, so now it's even more.)
  - "Big Data could know us better than we know ourselves." – Dan Gardner
  - "Data is the New Oil." – Clive Humby
  - The average person today processes more data in a single day than a person in the 1500s did in an entire lifetime.
  - Each of us now leaves a trail of digital "exhaust:" a very large stream of phone records, texts, browser histories, GPS data, and other information, that will live on forever.
  - "If data had mass, the earth would be a black hole." (S. Marsland)
- Banks, hospitals, retailers, laboratories, … store Terabytes of data daily.
- In U.S. markets 7 Billion shares are traded daily, 60% by HFT algo traders.
- HFT data storage needs about double every year.

Why Machine Learning?

- Computational Power
  - Fast data storage is cheap and getting cheaper every year (< 4¢/Gb compared to $437,500/Gb in 1980).



  - Moore's Law: "Computer Performance doubles every 1.5 to 2 years." (This comes from transistor size reduction. Moore's Law has held for several decades now.)

Why Machine Learning?

- Computational Power (cont.)
  - However, Moore's Law is slowing down due to physical limits:
    - *Mesoscopic Limit*: The crossover from bulk matter to the atomic/quantum regime (Transistor size today: ~14 nm; Bulk matter properties break down at ~5 nm? No longer bulk matter properties, Quantum Effects start to kick in).
    - *Dennard Scaling*: FLOPS/Watt should scale with Moore's Law. But this has already broken down: Clock speeds haven't increased for the last several years now.
  - To keep up the computational power increase, we now need new Approaches/Paradigms:
    - Multi-core and novel chip architectures
    - 3D Chips (Flash drives will have more capacity than HDs soon!)
    - Quantum Computing (immense leap in computational capacity and speed). Not quite a technological reality yet. Until then…
    - Parallel Processing—the new reality in high-power computing

Why Machine Learning?

- But the key to all this massive amount of data and computational power is *Information Extraction.*
  - Nowadays data is a competitive asset.
  - But… data is useless without information extraction.
  - Extracting information from data is a competitive *necessity*, no longer a choice or a nice-to-have.
- Enter… Machine Learning
  - Useful for extracting information from large data sets.
  - Has benefited from both Big Data and increases in Compute Power.
  - Has advanced tremendously in the last couple of decades.
  - Goes hand-in-hand with “Big Data” and “Big Computing.”
  - ML is now and will become even more important and highly prized (read: well-paid) in the foreseeable future. As “hot” now as ever.
  - Piqued your interest?
- On to Class Specifics…

Class Specifics

- T.A.: Liqun Zhu
- 7 Class Meetings: 3/25, 4/01, [ ]*, 4/15, 4/22, 4/29, 5/06, 5/13. (*No class on 4/08: Spring Recess.)
- Class meets 6:05 – 8:30 PM. Be on time!
- I won't be adhering to a textbook, but the following are useful…
- 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)

Class Specifics (cont.)

- 5 Homework Assignments (25%)
  - Due ***Before the Start of Each Lecture*** (no exceptions).
  - ***Late Homework Assignments will receive a grade of "0".***
  - Must be in PDF Format and sent by Email to *both* me and Liqun.
- In-Class Comprehensive Final Exam, May 06 (40%)
- Final Group Project, May 13 (25%)
  - Either an original research project *or* a reading from the literature.
  - If you choose to do original research, programming should be done in Matlab.
  - Groups can be 1-4 people.
  - Project Proposal due on or before April 29 by email to me for my approval.
  - Report and in-class Presentation due May 13 (last day).
  - Report and slides must be emailed in PDF Format (along with any code) to me.
  - Short (10 - 15 minute) presentation (one per group).
  - Teammates mutual assessments emailed to me on or before May 18.
- Instructor Discretion (10%)
  - Class participation, questions, contribution to discussion, independent work…
- See Class Syllabus for further clarification

Machine Learning in Context

- ML is a Branch of Artificial Intelligence (AI). History…
  - "Back in the Day" AI was based on logical inference and rules ("crisp" logic).
  - AI had grandiose expectations: Intelligent machines would surpass humans in every way, do our grunt work, ...
  - Then came disillusionment: The "Turing Test" of machine intelligence has yet to be passed with flying colors.
  - Realization: Conventional "crisp" logic by itself doesn't work because the real world is too messy (too much incomplete, faulty, noisy information).
  - Case in Point: See the documentary on IBM's Watson, the Jeopardy! quiz show champion, where ML came to the rescue.
  - The focus today is on less lofty but more pragmatic applications (robotics, pattern recognition, text mining, etc.).

Machine Learning

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

Technology Adoption Curve for AI

- Technology Adoption Curve (Gartner Group):

- ML is in the Productivity Plateau.

ML Paradigms

Supervised vs. Unsupervised Learning

- **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 (automated extraction)

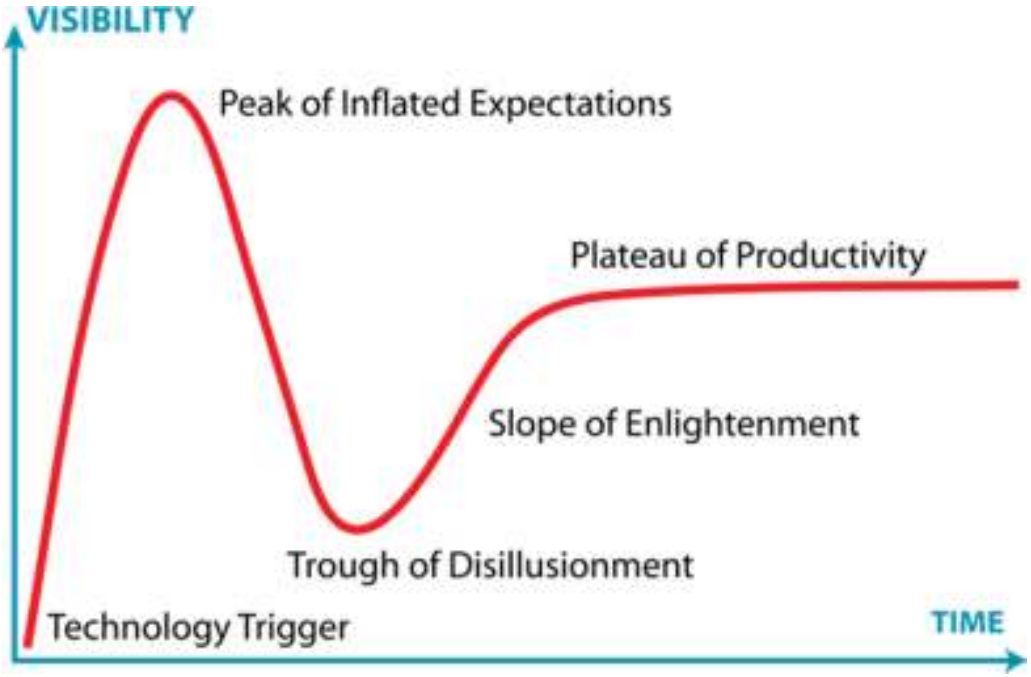
New Paradigm for AI

- The biological brain is not “logical” but heuristic-based.
- Has massive $(\sim 10^{11})$ arrays of simple processors (neurons)
- Has massively parallel connectivity (synaptic connections to up to 10^4 other neurons)
- Has the equivalent of 1 trillion bit/s processing power, and up to 1,000 TeraBytes of storage capacity (by comparison, Library of Congress has about 10 TeraBytes).
- Could have even more capacity: Glial cells used to be thought of as structural, but are now thought to also play a role in communication and neuroplasticity; they are about 10 times more numerous than neurons.
- Biological brains are better able to handle uncertainty, noise, incomplete information than old “crisp logic” AI paradigms.

Neural Networks

- Enter… Artificial Neural Networks (ANNs or NNs)
- One of the first developments in ML
  - Originally, were meant to emulate biological brains, although this notion was a bit overhyped, in hindsight.
  - Consist of many simple units working in parallel.
  - Can learn adaptively by example.
  - Can handle uncertainty, noise, incomplete information.
  - Disillusionment: Black boxes, difficult to interpret/elucidate, computationally intensive, cannot fully model/explain biological brains.
  - Nevertheless, they are very useful tools: Their focus within ML is on applications, although there are still insights that NS gains from ANNs.

Technology Adoption Curve for Neural Nets



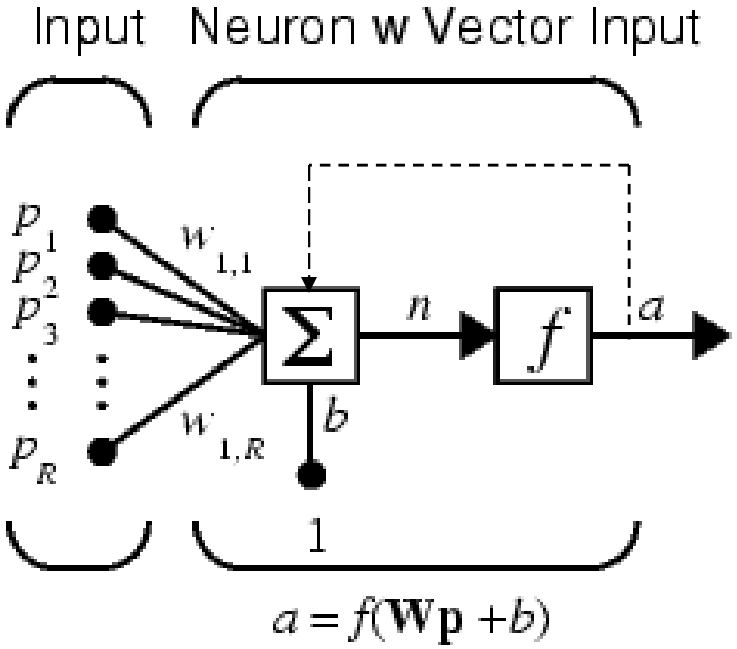
- Used in medical diagnosis
- Fraud detection
- System Control
- Time series forecasting
- Credit scoring
- Bond rating
- Pattern recognition, …

Artificial Neural Networks

- ANNs are the archetypes of Machine Learning.
- Can perform many ML tasks (classification, clustering, forecasting, universal approximation, etc…).
- They illustrate many techniques used in ML.
- They are powerful and have wide applicability.
- A great place for us to start...

The Neural Network Analogy of the Biological Brain

- Like the brain's, ANN's basic unit is the ***Neuron*** (a.k.a.: ***Node***).
- The Neuron aggregates input patterns (stimuli) and responds via an Activation Function (a.k.a.: Transfer Function).
- Learning is achieved through synaptic strengths (weights).
- Massively Parallel, (possibly with feedback where output goes back to input).



R = Number of input patterns

w = Weight

b = "Bias" ("Reference Stimulus")

n = Input to activation

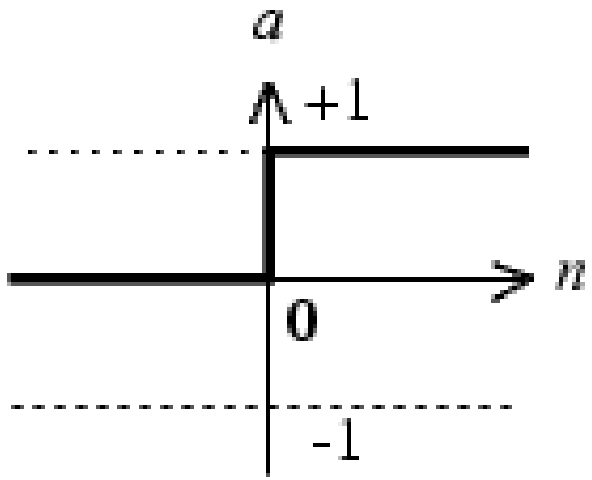
f = Activation Function

a = Neuron's Actual Output

Not shown: Multiple connections to other Neurons.

Threshold Activation

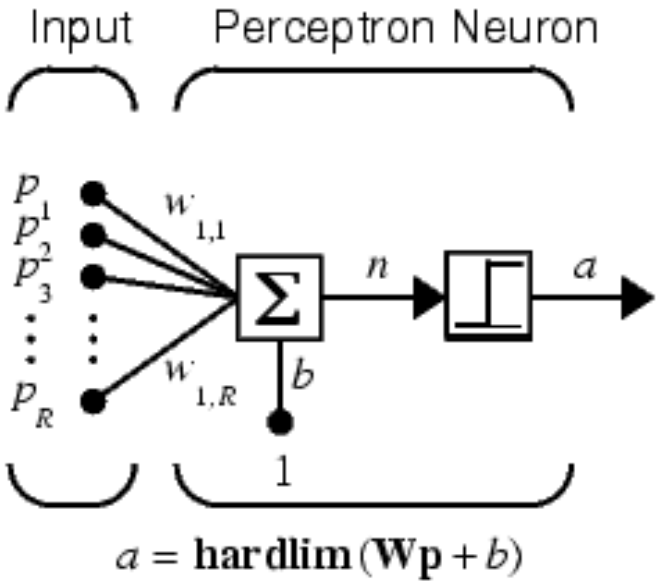
- A type of Activation (or Transfer) Function is the Threshold or “Hard Limit” Function, where the Neuron “fires” when the sum of inputs exceeds a threshold:



- This is similar to how biological neurons work.

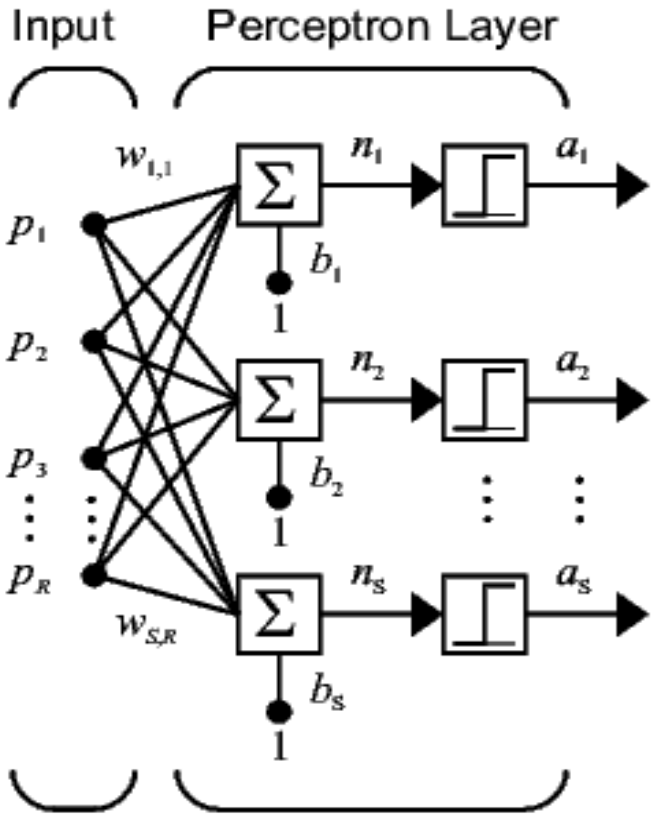
The Perceptron

- A *Perceptron* is a type of ANN that has a Hard Limit Activation Function.
- Here's an illustration of a Perceptron Neuron (or Node) with Hard Limit Activation:



Perceptron Architecture

- Perceptrons have a high degree of connectivity and parallelism.
- This is starting to look like a simplified biological brain structure: high parallelism and threshold activations.



Many Hard-Limit Activated Nodes

R inputs

S outputs

W is an SxR matrix of weights

b is an S vector of biases

Now We 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 the Biological Brain analog, the weights and biases are the synaptic connection strengths.
- In *Supervised Learning* the Training Set contains targets:

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

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

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

- Quantities in brackets are called *Exemplars* (Pattern-Target or Input-Output pairs in the case of Supervised Learning; Input Patterns only in Unsupervised case).

Perceptron Learning Rule

- Learning Objective: Minimize the error $\boldsymbol{e}$ between actual response $\boldsymbol{a}$ and desired target $\boldsymbol{t}$ (over all exemplars):

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

- Let's look at this closely (scalar example); there are 3 possibilities for the Perceptron neuron:
  1. An input is presented and the actual output is correct so that $\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{a} = \boldsymbol{0}$ and the weights are unaltered: $\Delta \boldsymbol{w} = \boldsymbol{0}$.
  2. Neuron output is 0 when it should've been 1 ($\boldsymbol{e}$ = 1), and the weights are 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 are decremented by the input vector.

Perceptron Learning Rule

- To recap, we start with an arbitrary set of weights and:
  1. If $\boldsymbol{e} = 0$, $\Delta\mathbf{w} \leftarrow 0$; Done.
  2. If $\boldsymbol{e} = 1$, $\Delta\mathbf{w} \leftarrow \mathbf{p}^T$; Continue…
  3. If $\boldsymbol{e} = -1$, $\Delta\mathbf{w} \leftarrow -\mathbf{p}^T$; Continue…
- More succinctly (left as an exercise):

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

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

- With obvious extensions for multiple neurons.

Perceptron Learning Rule (again)

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

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

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

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

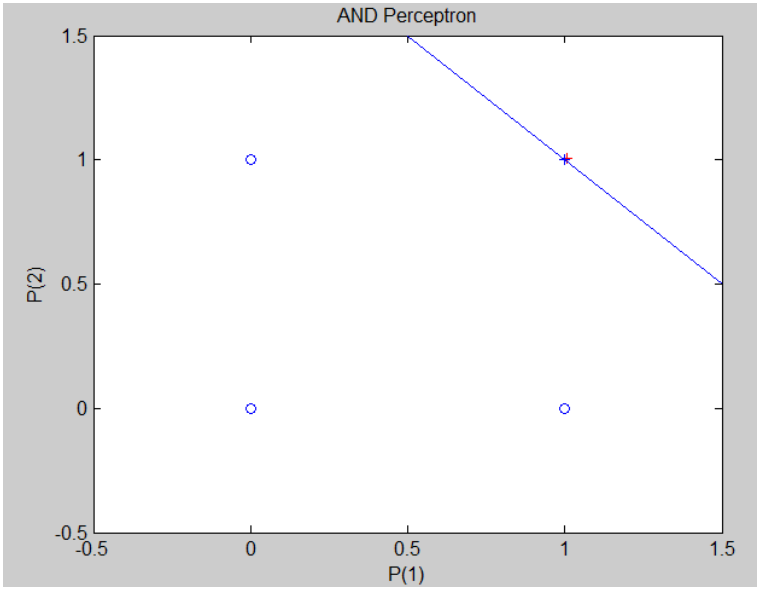
OR

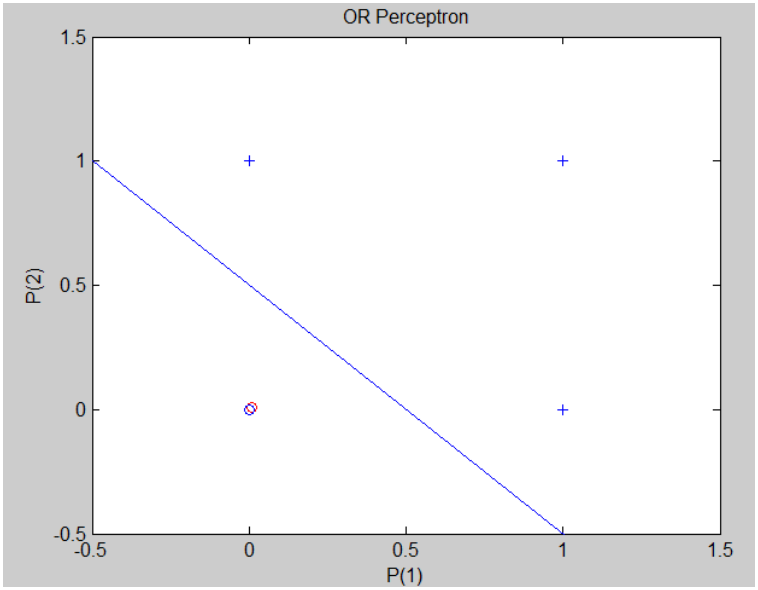
Input Pattern		Target
0	0	0
0	1	1
1	0	1
1	1	1

…

- Works: Not by design, but by training on data!

Perceptron for AND and OR

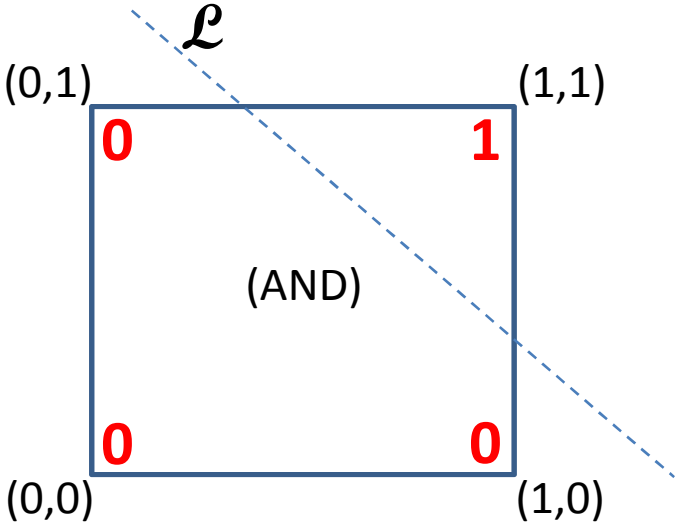


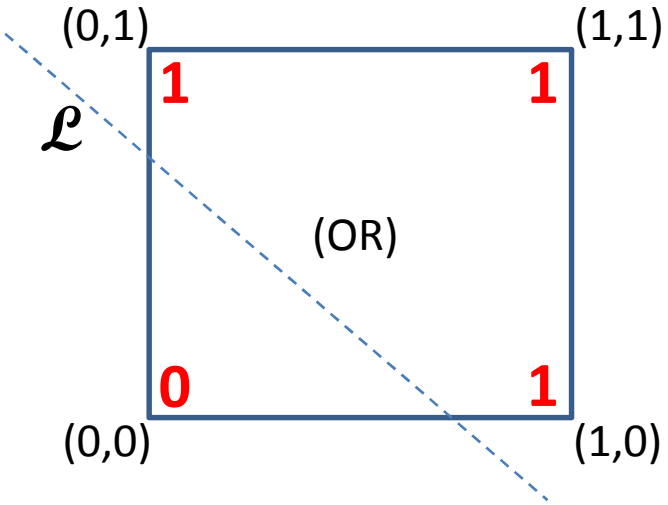


- Single Perceptron with two inputs works for AND and OR functions.
- Straight Line separates the two output classes “o” and “+”.

Linear Separability

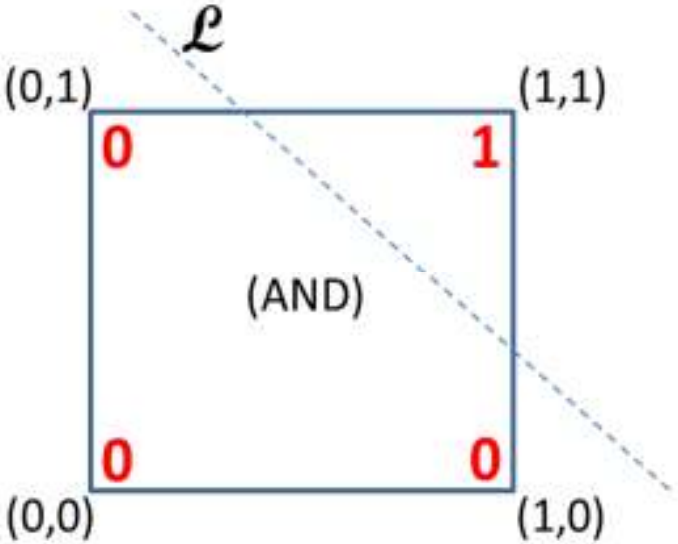
- Perceptrons work well with AND & OR functions.
- In fact, they work with all *Linearly Separable* problems.
- The Hard Limit Activation function allows Perceptron Neurons to "classify" input vectors by dividing the input space into two regions via a Decision Boundary Line.
- The Decision Boundary Line, $\mathcal{L}$, is defined by $\mathbf{Wp} + \mathbf{b} = 0$ and is actually a Hyperplane in higher dimensions. The output (in red) will depend on which side of this line (or hyperplane) the input lies:

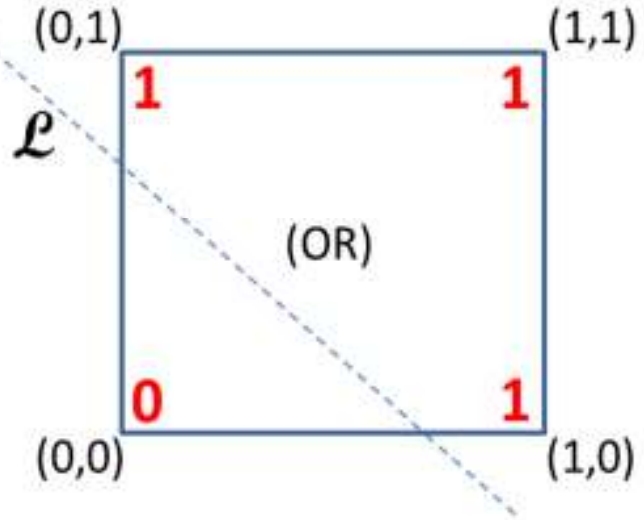


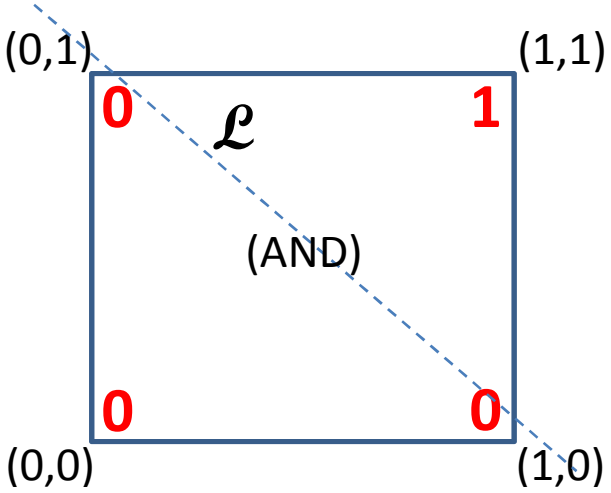


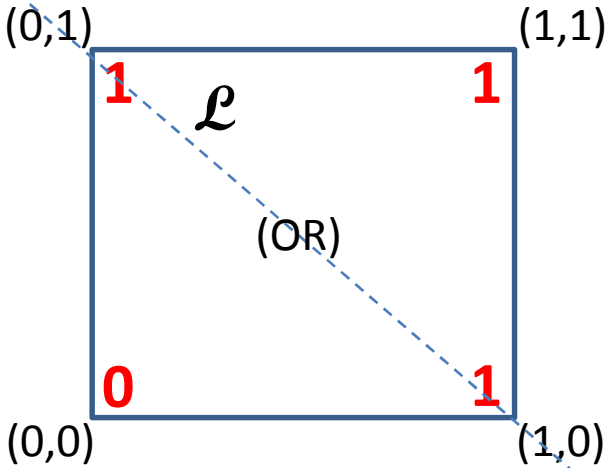
Linear Separability (Aside):

- There is freedom to choose decision boundaries, but…
- Are all decision boundaries equally "good" (*e.g.* robust to roundoff)?



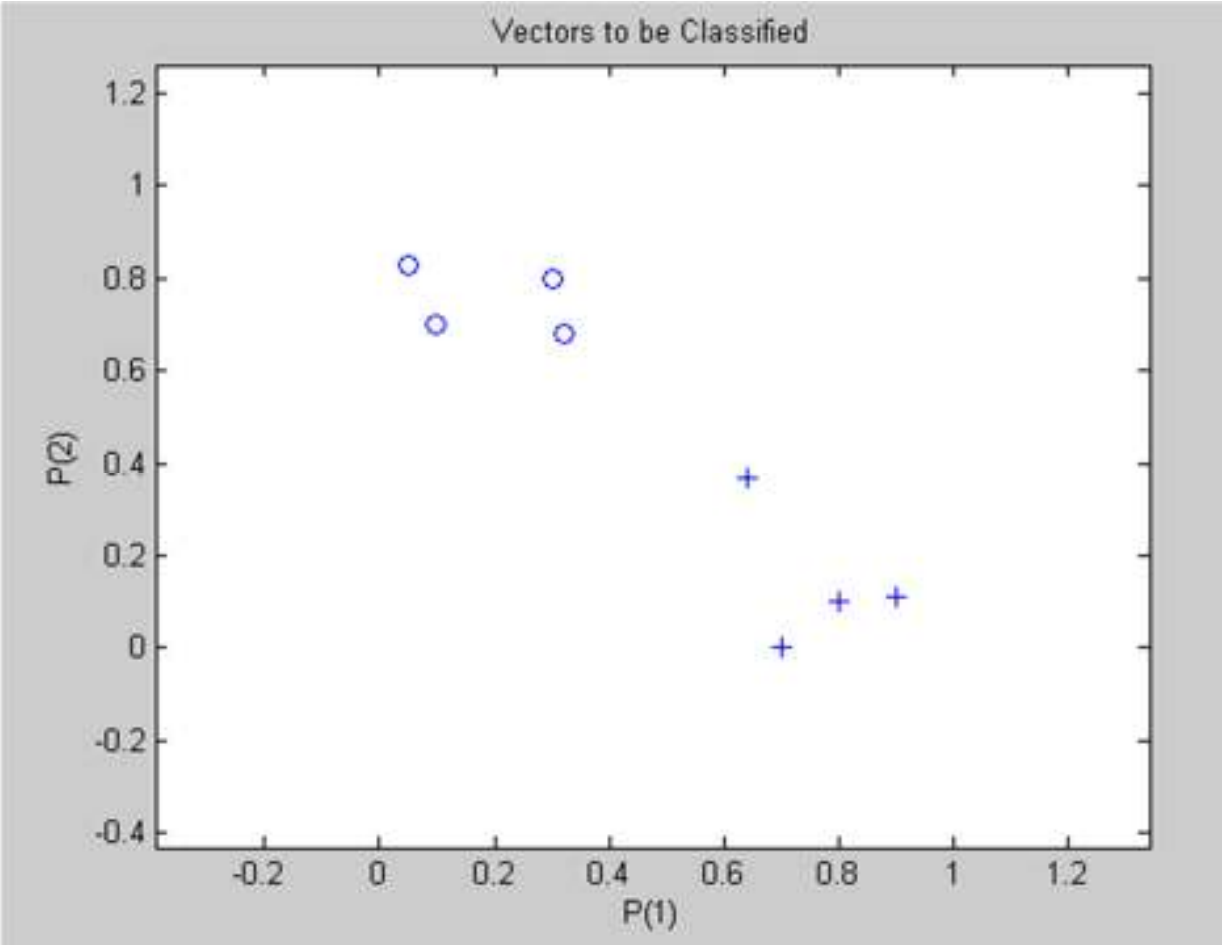






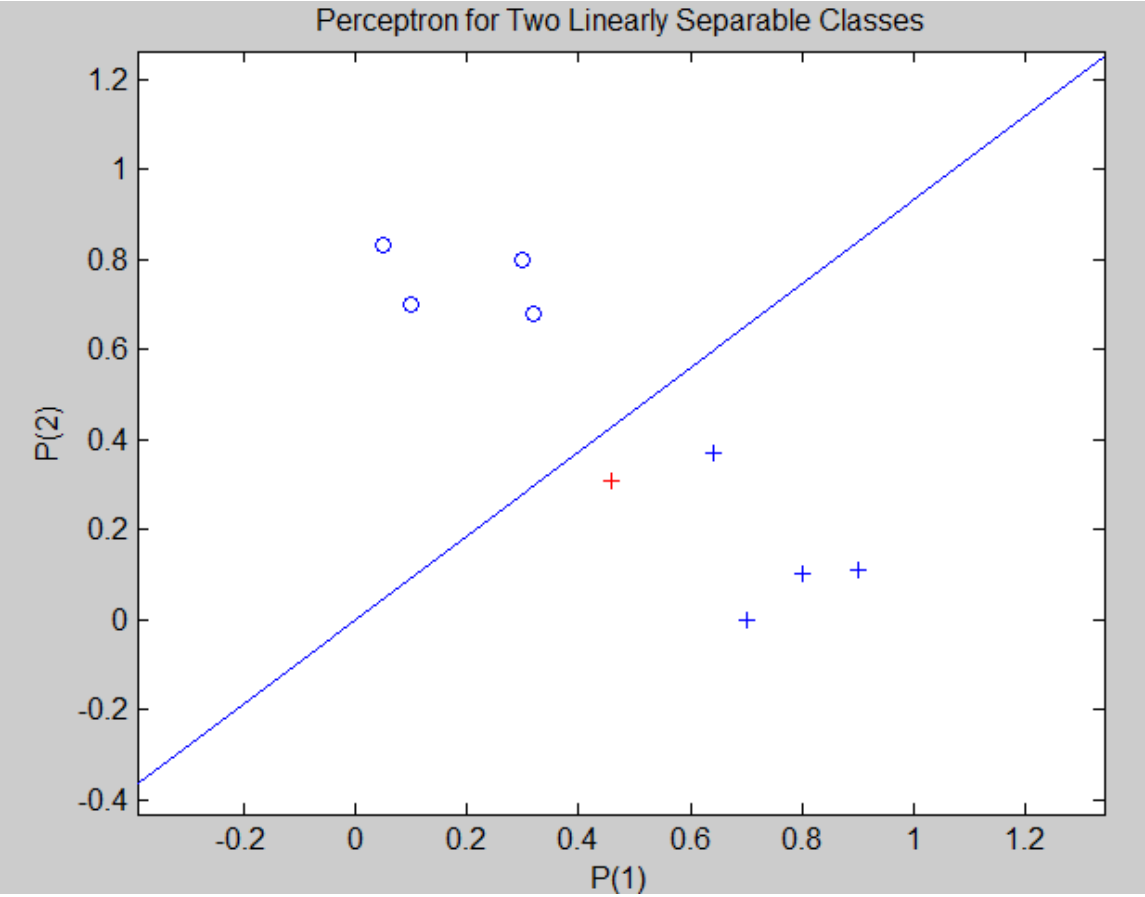
Linear Separability

- Perceptrons can also classify non-binary input vectors as long as the two output classes are Linearly Separable.
- Are these vectors Linearly Separable?



...

Linear Separability



- Single Perceptron with two inputs works for non-binary Linearly Separable Classes.
- Straight line separates the two non-binary classes.

Does it Work for Other Classes of Problems?

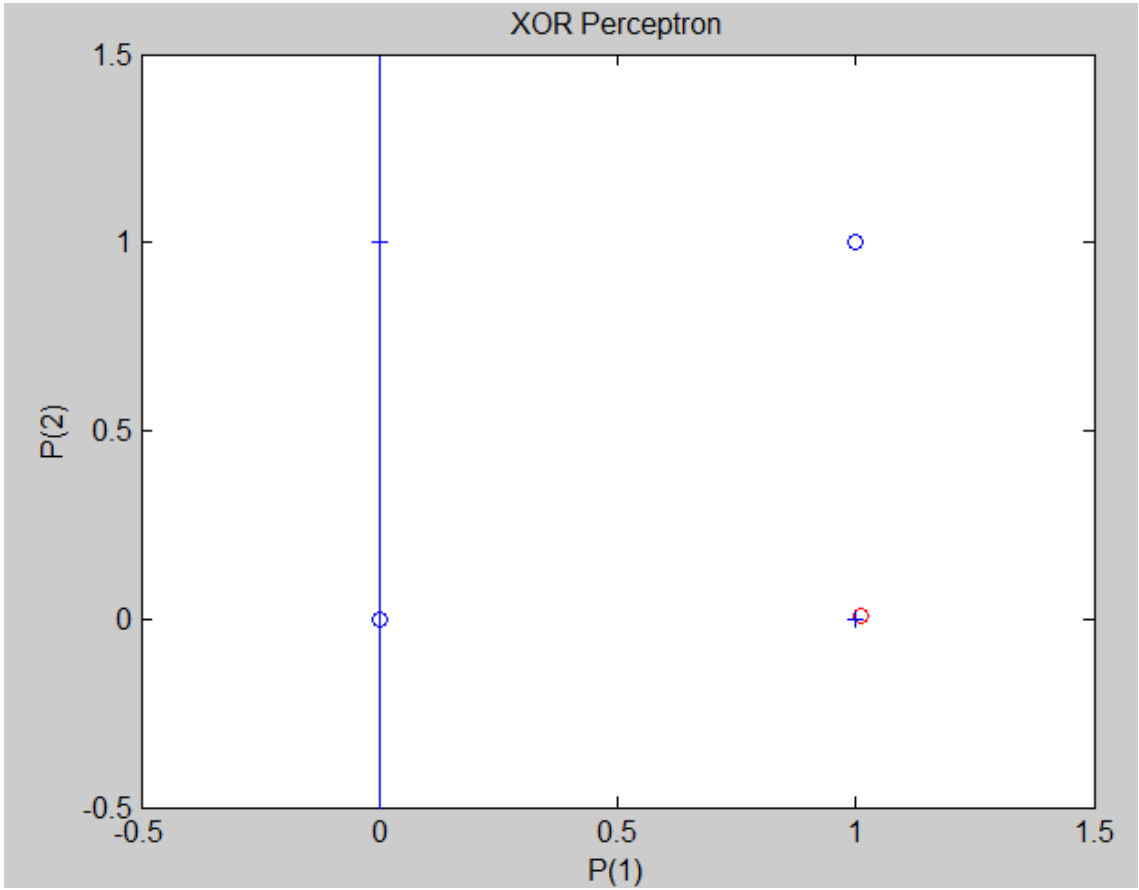
- Consider the Exclusive Or Function (XOR):

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

…

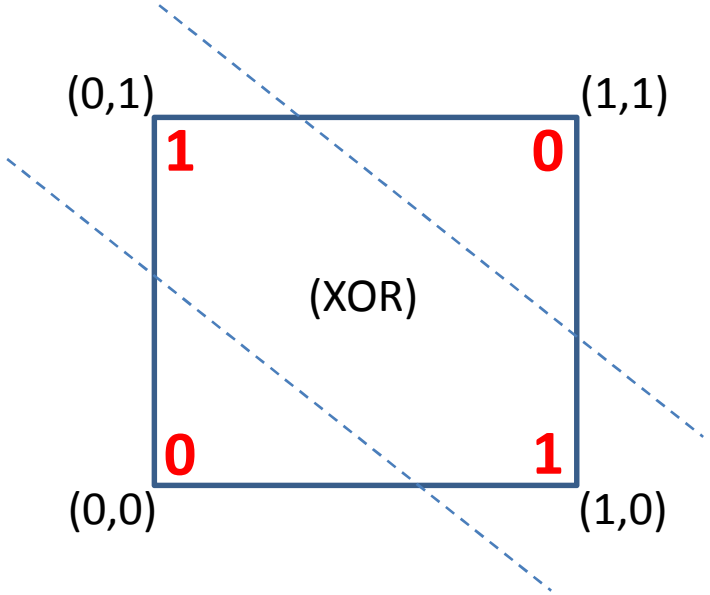
- The Perceptron does ***not*** work, no matter how long it trains for!

Perceptron for XOR



- Single Perceptron with two inputs does ***not*** work for XOR.
- A single line cannot separate the two output classes.

XOR is Linearly Non-Separable



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

Can We Fix This?

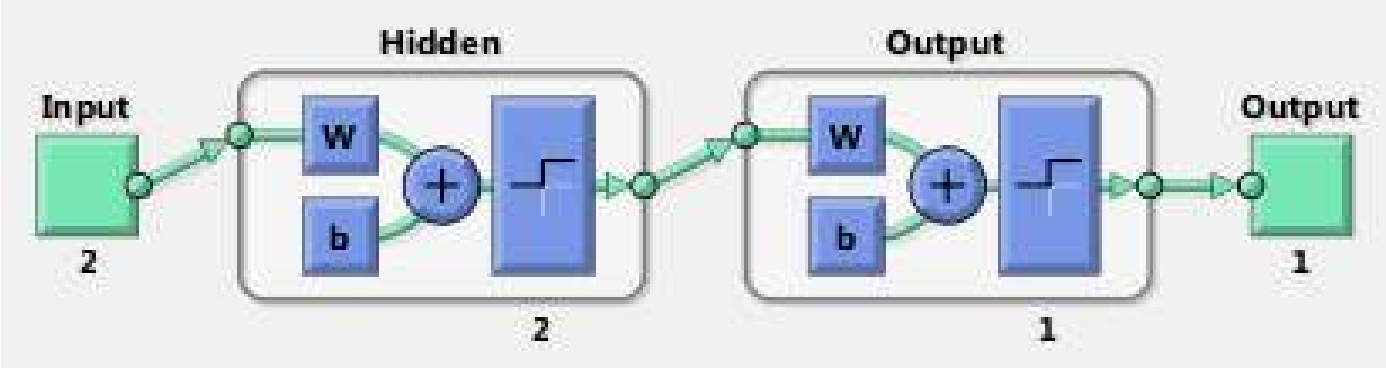
- Yes…
- Note that we can express XOR as a combination of ANDs and ORs:

$$x \textbf{ XOR } y = (x \textbf{ OR } y) \textbf{ AND } [\textbf{NOT}(x \textbf{ AND } y)].$$

1. Train an OR Perceptron.
2. Train a NOT-AND (NAND) Perceptron (guaranteed Linearly Separable – **why?**)
3. Train an AND Perceptron and use the outputs of the previous two Perceptrons as inputs .
4. Alternatively, you only need to train an OR and an AND Perceptrons (**why?**).

- This *cascading* or *layering* of Perceptrons is known as a Multi-Layer Perceptron (MLP).
- This fix requires one extra layer of neurons called the *Hidden Layer.*

Multi-Layer Perceptron for XOR

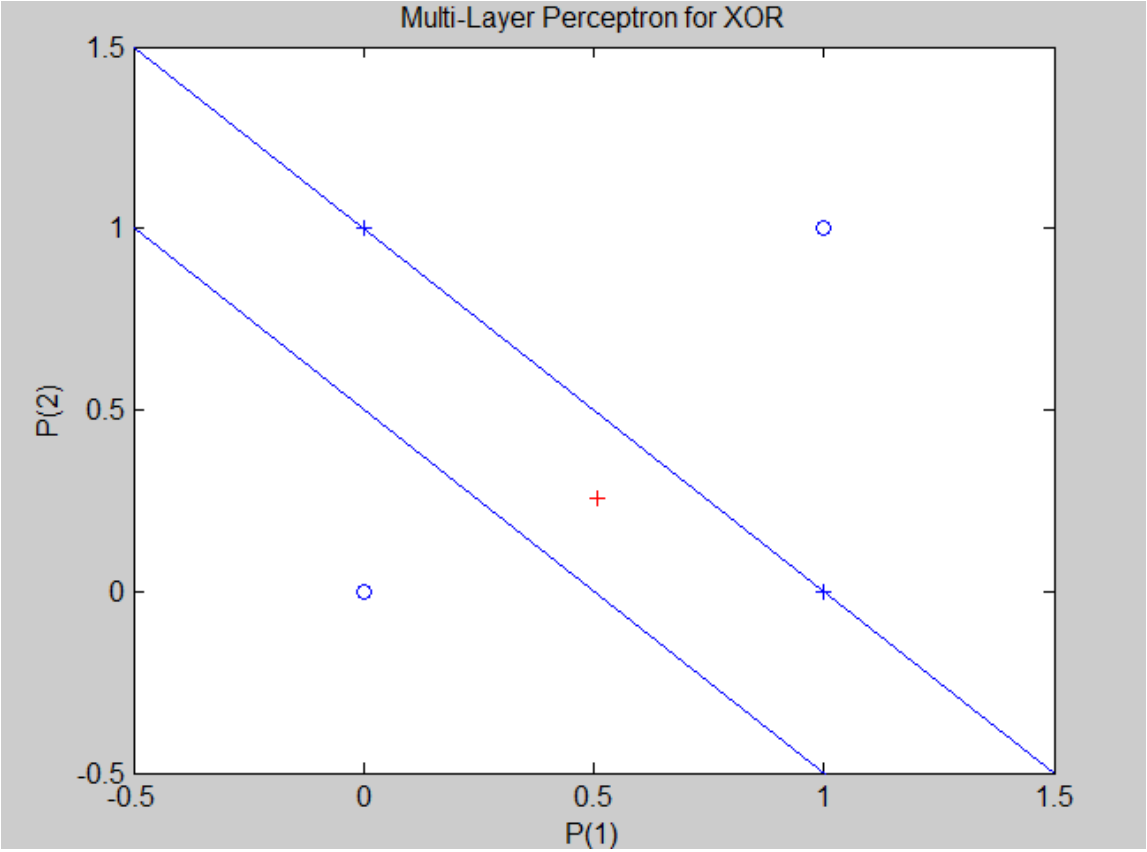


- This diagram illustrates a Multi-layer Perceptron with two inputs, two hidden-layer nodes, and one output node.
- The two hidden nodes are made up of a previously-trained OR single Perceptron and a previously-trained NAND single Perceptron.
- The output node is a previously-trained single AND Perceptron.
- This combination should allow the MLP to implement the XOR function as follows:

$$x \textbf{ XOR } y = (x \textbf{ OR } y) \textbf{ AND } [\textbf{NOT}(x \textbf{ AND } y)].$$

...

Can We Fix This?



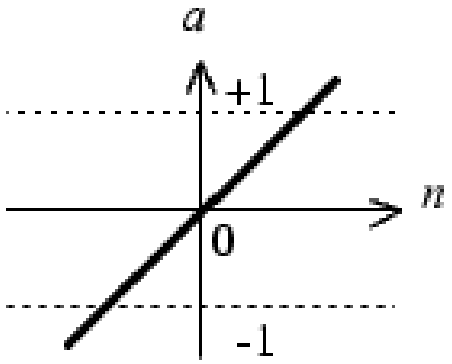
- This works, ***but*** it's not very useful as is! We have to do much of the work for the MLP manually—this defeats the automated purpose of ML.
- More complicated functions require even more manual work…

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)*."
- This Linearly Non-Separable Problem cannot be solved with a single line (hyperplane in higher dimensions).
- The fix proposed above (assembling individually trained Perceptrons) is not useful because it doesn't give us an automated (or even concise) way of solving this combinatorially explosive problem.
- Things get even more interesting when we have more than two classes.
- We need a new learning rule where learning is not manual!
- On the bright side, we've learned that introducing an extra (hidden) layer of neurons allows us to solve Linearly Non-Separable Problems.

Searching for a New Learning Rule

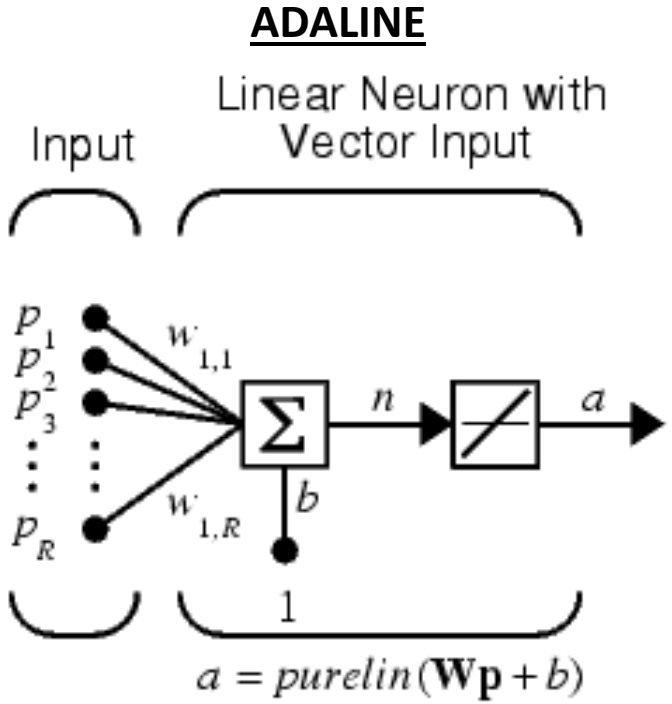
- We want an automated way of solving Linearly Non-Separable Problems.
- Note that the Perceptron learning rule is “coarse-grained” in that the error surface (and hence the update rule) is quantized due to the Hard-Limit Activation. (Hint: Smooth Activations?)
- Idea: Replace the Hard-Limit Activation with a Smooth (differentiable) Activation, which would make the error surface amenable to gradient-descent optimization.
- E.g.: A Purely Linear Activation:



$a = purelin(n)$

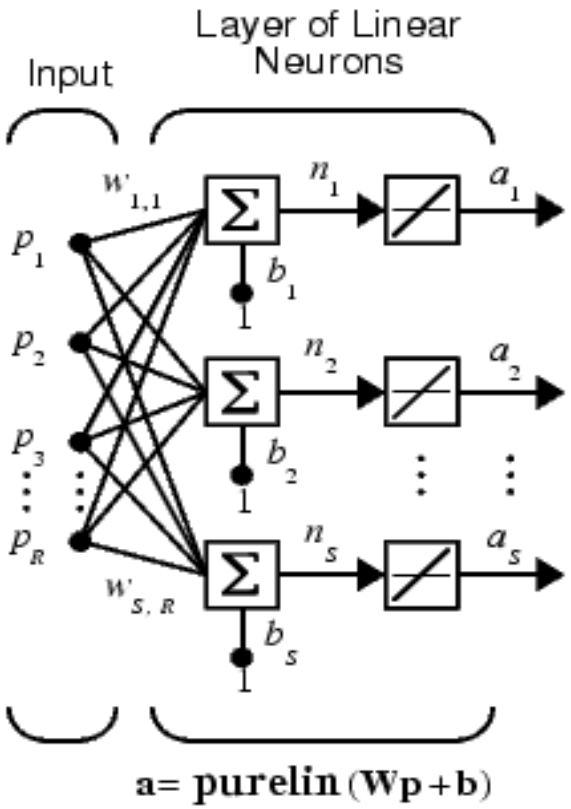
ADALINEs

- ADALINES (Adaptive Linear Elements) are Perceptrons with Purely Linear Activations:



- Like the Perceptron, the ADALINE also has a Decision Boundary at the line (hyperplane) $\mathbf{W}\mathbf{p} + \mathbf{b} = 0$, but now the output is *continuous* and the error surface is *differentiable*.

ADALINE Architecture



Linear Activations

R inputs

S outputs

W is an SxR matrix of weights

b is an S vector of biases

Continuous Activations

- Smooth (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.
- One such rule is known as the *Delta Learning Rule*.

Delta Learning Rule

- For smooth activations, the weight update at the k^{th} iteration is:

$$\Delta \boldsymbol{W_k} = -\eta \nabla_w e_k$$

(where η is the *Learning Rate*).

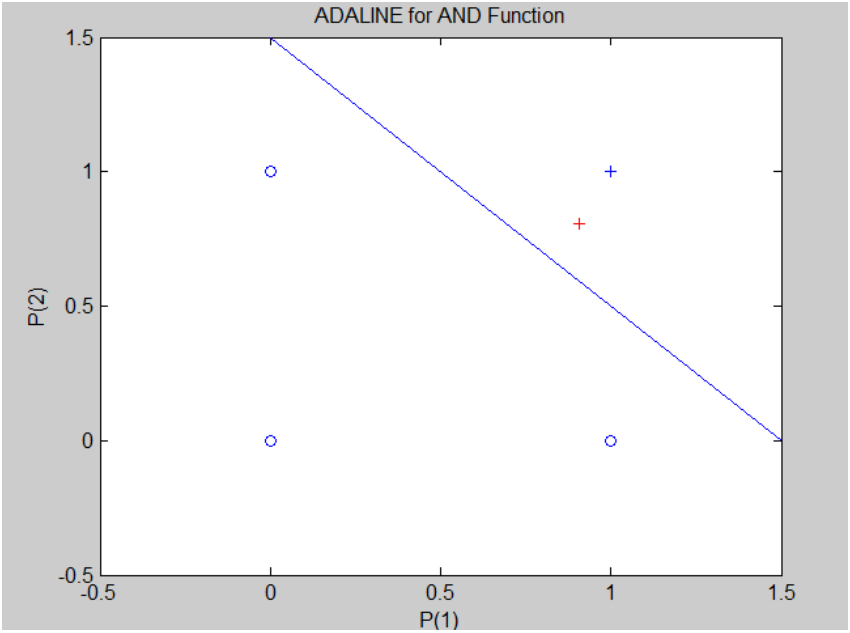
- For a quadratic error, *e.g.:* $\boldsymbol{e} = \frac{1}{2}(\boldsymbol{t} - \boldsymbol{a})^2$, the error surface should have a global minimum and the procedure converges (eventually), and is known as the *Delta Learning Rule*, the *Widrow-Hoff Rule*, or the *LMS Algorithm*.
- For linear activations, the update rule (for ADALINEs) is:

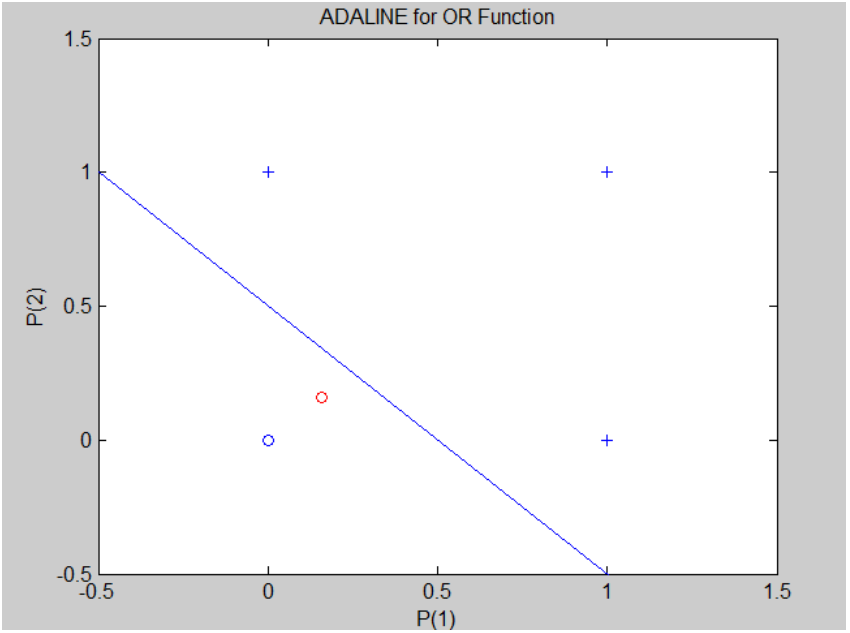
$$\Delta \boldsymbol{W_k} = \eta \boldsymbol{e}\boldsymbol{p}^{\boldsymbol{T}} \text{ (why?)},$$

which looks very similar to the Perceptron case.

…

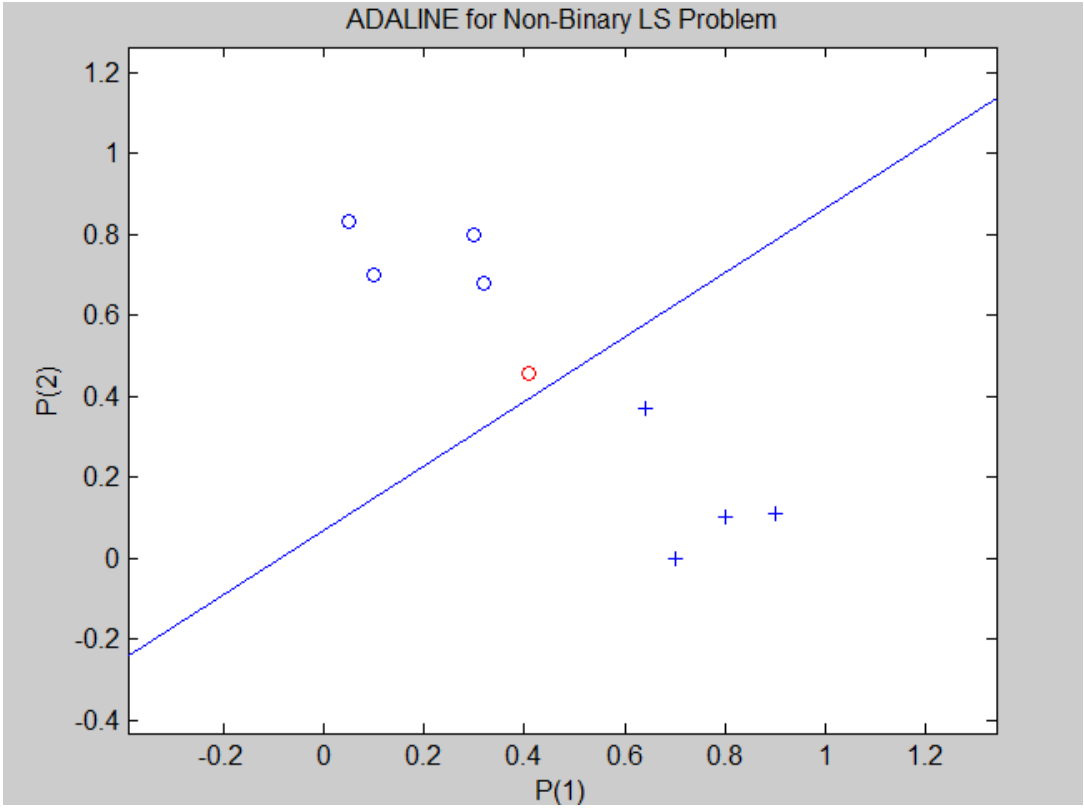
ADALINEs for AND and OR Functions





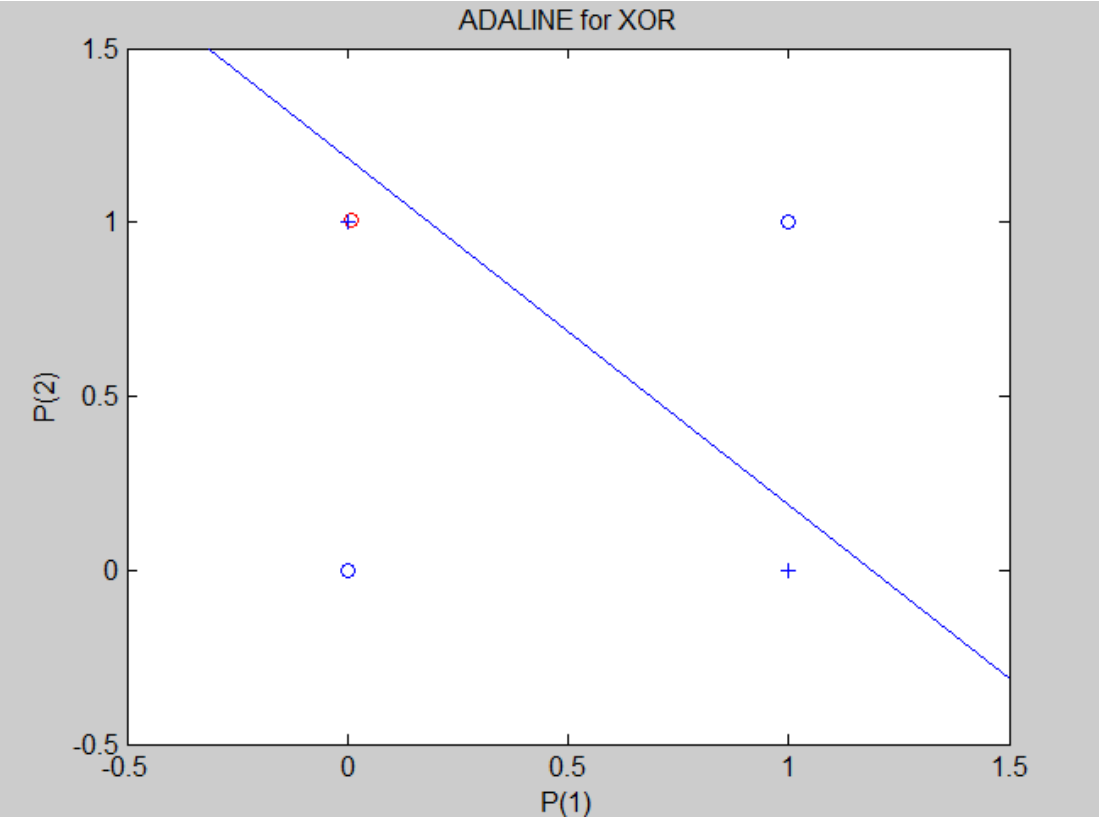
- ADALINES work well for AND and OR functions.
- Notice the Decision Boundary Lines are more “robust” than the ones generated by the AND and OR Perceptrons; i.e., they better classify inputs in the neighborhood of the binary inputs.
- More general linearly separable problems …

ADALINEs for More General LS Problems



- ADALINES work well for more general Linearly-Separable Problems.
- XOR? …

ADALINEs for XOR

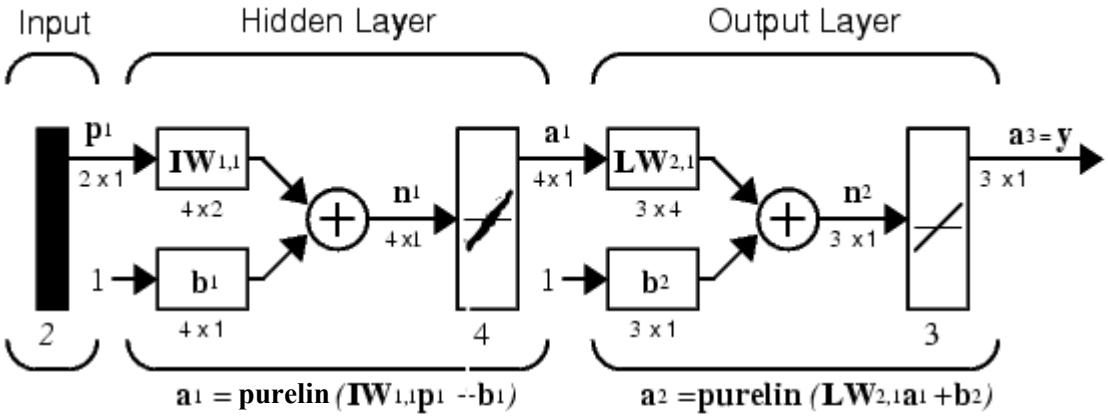


- ADALINEs do ***not*** work for the XOR problem.
- ADALINEs can only tackle Linearly Separable cases as with the Perceptron case. Hint from the Perceptron case: Introduce more layers.
- To generalize and try to tackle Linearly Non-Separable cases, try cascading to form Multiple ADALINEs or MADALINEs with a Hidden Layer.

MADALINEs

- This diagram illustrates a Multiple ADALINE (MADALINE) with one hidden layer.

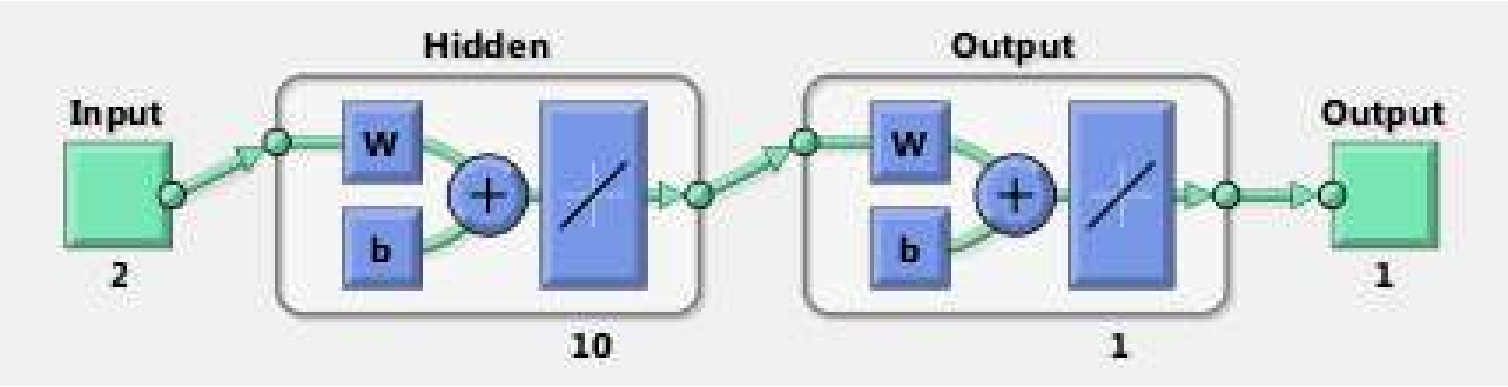
Layer Diagram



MADALINEs Learning Rule

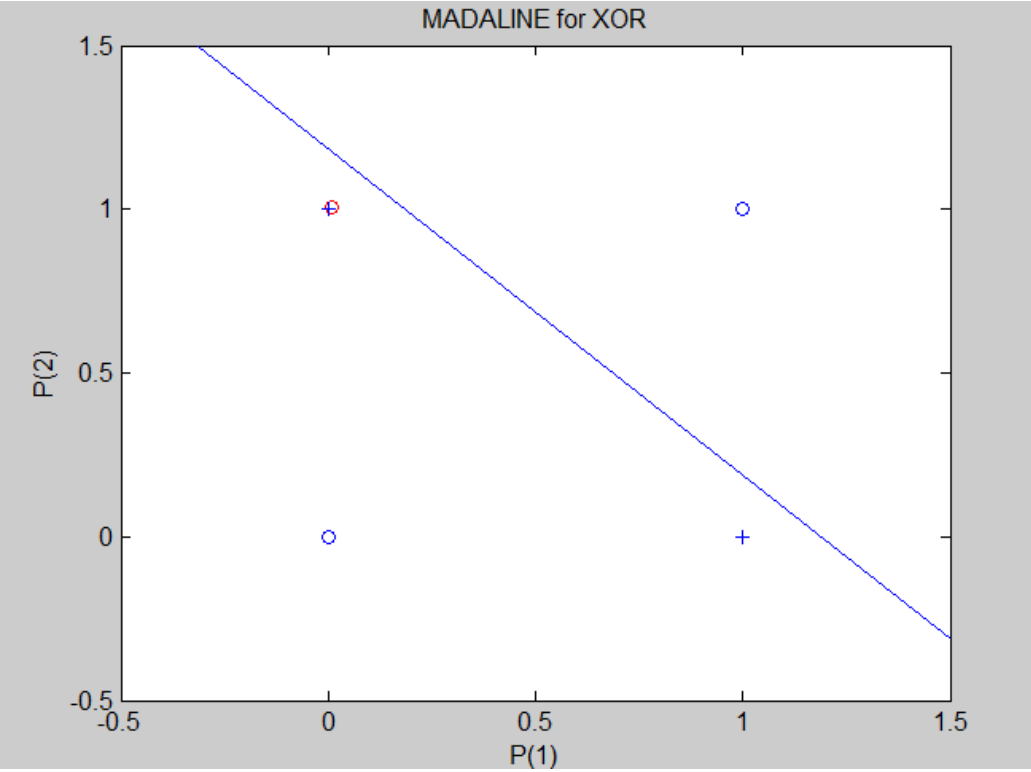
- One can generalize the single ADALINE learning rule (the Delta Learning Rule) to MADALINEs having one or more hidden layers by propagating the output error backwards through repeated applications of the chain rule of differentiation **(left as an exercise).**

- Can the MADALINE below, with 10 hidden nodes, solve the XOR Problem?



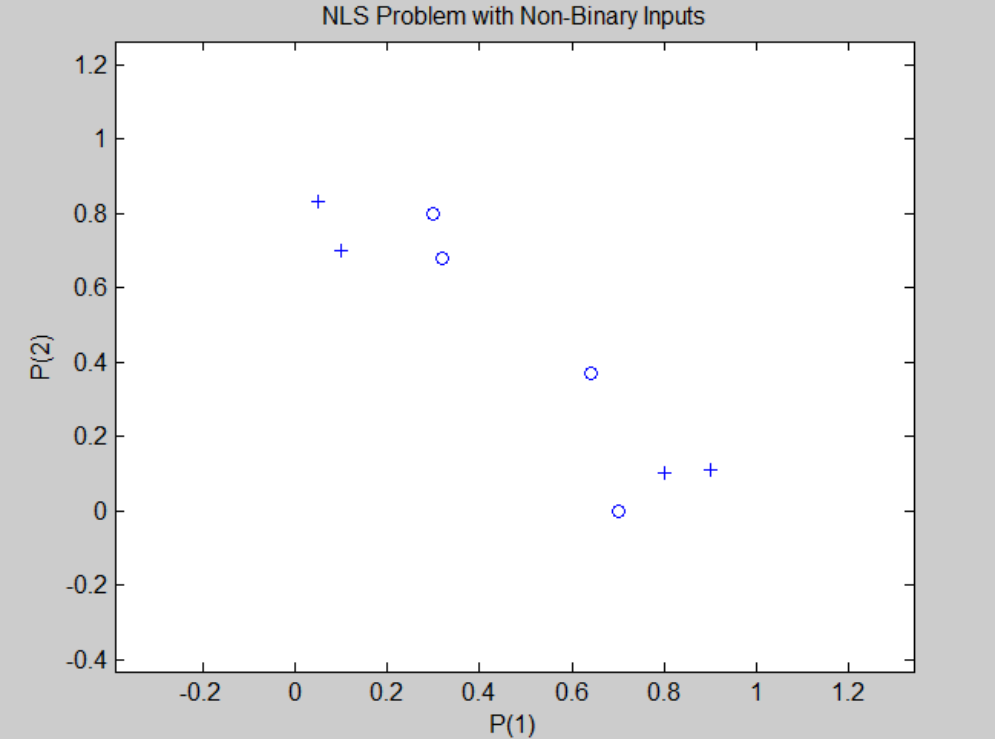
...

MADALINEs for the XOR Problem



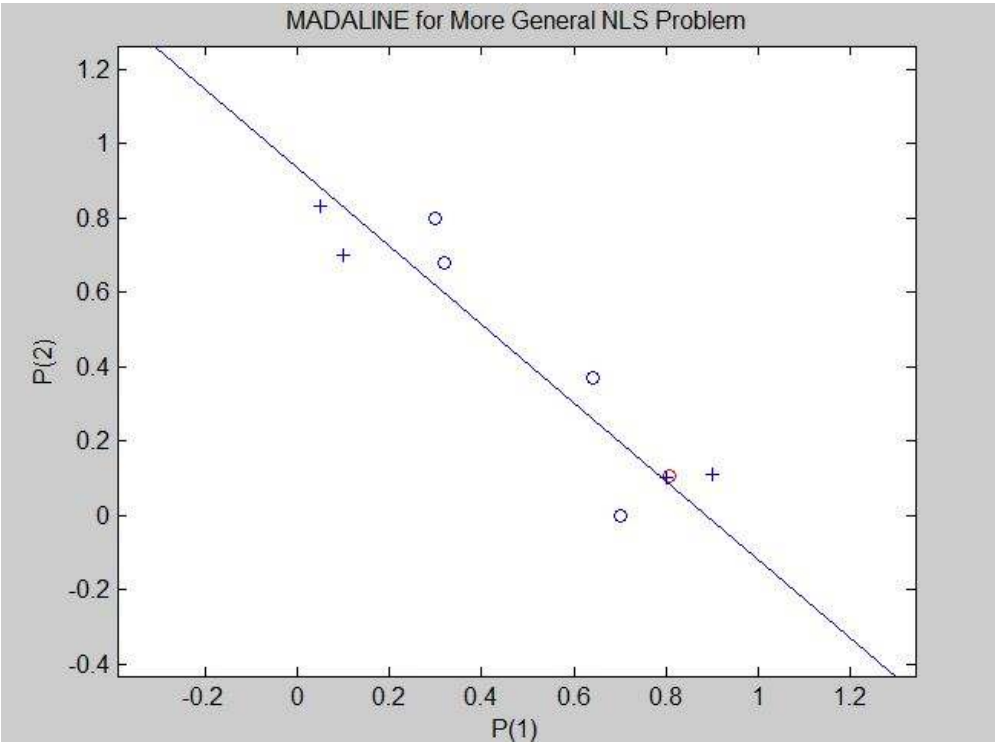
- Even with 10 Hidden Nodes, the MADALINE fails at the XOR Problem.
- It appears that MADALINEs cannot handle Linearly Non-Separable Problems.

A More General NLS Problem



- This is a more general Non-Linearly Separable Problem with non-binary inputs.
- Clearly, a single Line (Hyperplane) could not separate the two output classes.
- We can try a MADALINE on this problem …

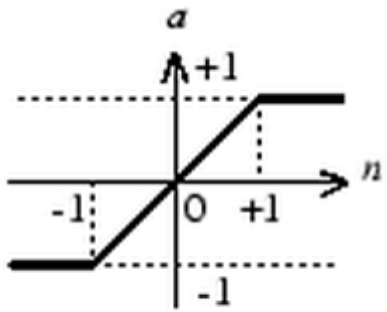
MADALINEs for More General NLS Problems



- Again, even with 10 Hidden Nodes, MADALINEs cannot solve Linearly Non-Separable Problems.
- Is this surprising?
- No, since we can show that MADALINEs implement a single classification hyperplane of the form $\widetilde{\mathbf{W}}\mathbf{p} + \tilde{\mathbf{b}} = \mathbf{0}$, with $\widetilde{\mathbf{W}}$ and $\tilde{\mathbf{b}}$ being functions of the hidden and output weight matrices and biases. **Left as an exercise.**

In Search for an NLS Problem Solver

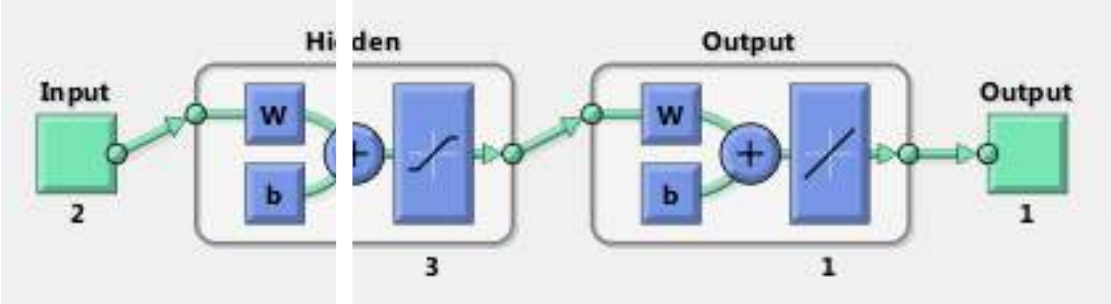
- Idea: Combine features of both Perceptron and ADALINE.
- Instead of a linear Transfer Function, try truncated or Saturated Linear Transfer Functions (satlins):



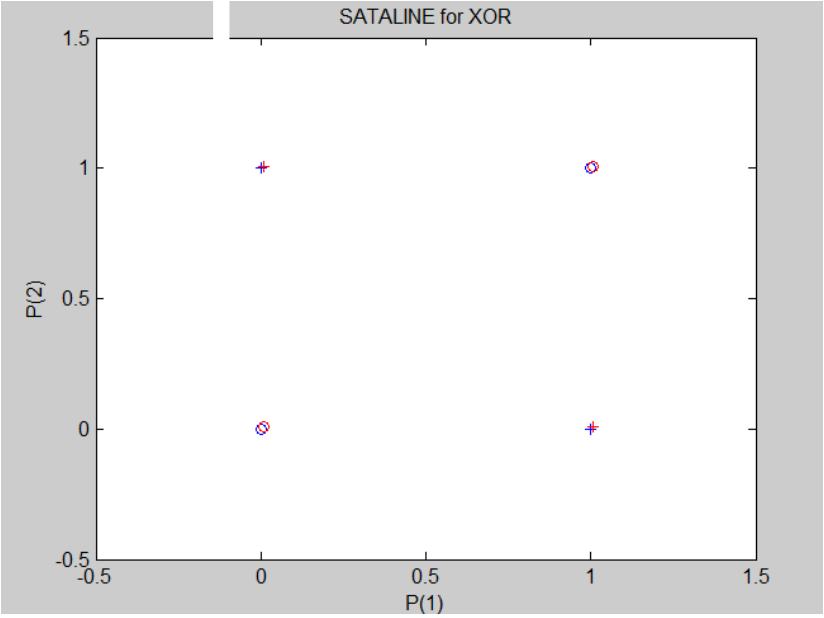
$a = satlins(n)$

- Using the satlins, form a SATALINE (Saturated Adaptive Linear Network) and try on XOR…

SATALINE for XOR



- This is a SATALINE with 3 Hidden Nodes…



- The SATALINE works perfectly to solve the XOR Problem in an automated fashion using the Delta Learning Rule.

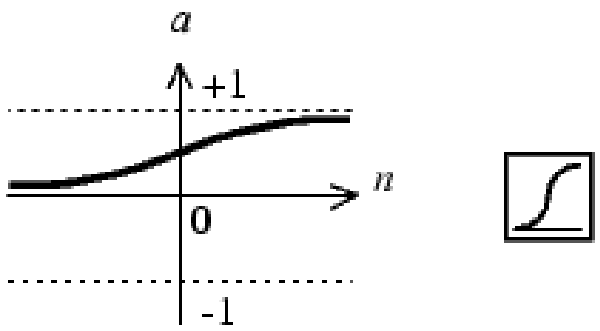
Universal Approximators

- 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 any mapping to an arbitrary degree of accuracy.
- These Universal Approximators require ***Hidden Layers*** (as we hinted at by the MLPs).
- However, they require ***Non-Linear Activations***, as hinted at by the SATALINES.
- And, for automated learning, they require ***Smooth (Differentiable) Activations***, as with ADALINEs.

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

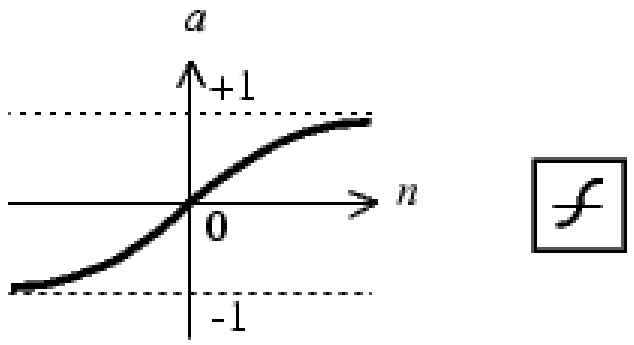
Feedforward Neural Networks

- Multilayer Feedforward Neural Networks are like Multilayer Perceptrons but with *Non-linear Continuous Activations*, instead of Hard-Limit Activations.
- They Typically use *Sigmoidal* Activations; *e.g. Logistic* (LogSig) or *Hyperbolic Tangent* (TanSig):



$a = logsig(n)$

Log-Sigmoid Transfer Function



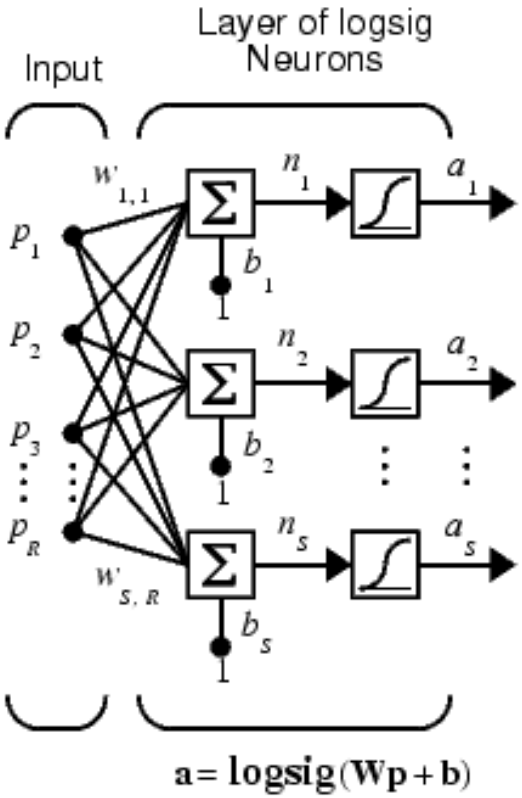
$a = tansig(n)$

Tan-Sigmoid Transfer Function

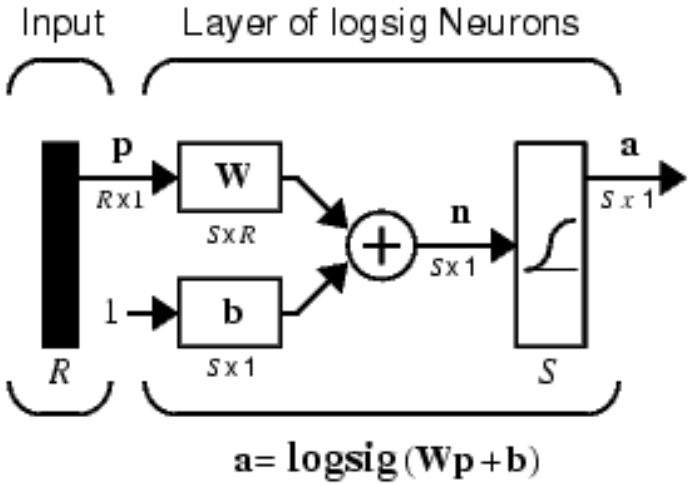
Non-Linear Activation Function Examples

- Logistic: $a(x) = \frac{1}{1+e^{-x}}$, Range: (0,1)
- Tanh: $a(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, Range: (-1,1)
- Modified Logistic: $a(x) = \frac{1-e^{-x}}{1+e^{-x}}$, Range: (-1,1).
- There are many others…

Feedforward NN Layer Architecture

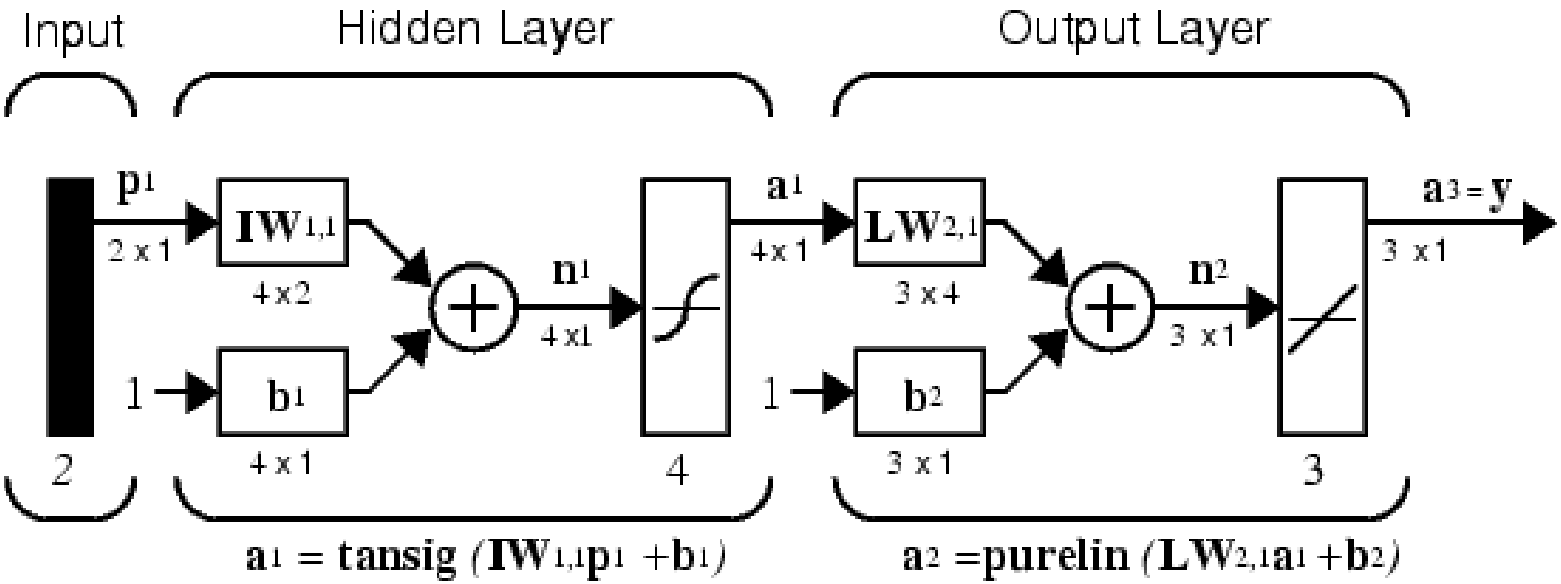


Layer Diagram



- Called "Feedforward" because outputs don't loop back into the inputs.

Feedforward NN Architecture



- Single Hidden Layer with Tanh Activation (tansig).
- Linear Activation Outputs (purelin).
- This is the **Workhorse Architecture**!
- Can solve most problems with it, including function approximation, classification, clustering, forecasting, etc.

Learning Rule: Backpropagation of Errors

- In 1986 Rumelhart, Hinton, and Williams (*) popularized the general-purpose Learning Algorithm for Multilayer Feedforward Networks, known as: *Backpropagation*.
- This (along with increased computational power) led to a resurgence in the field of Neural Networks.
- 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.

(*) Rumelhart, D.E., Hinton, G.E., and Williams, R.J.: *Learning representations by back-propagating errors*. **Nature**, **323**, 533-536 (1986)

Backpropagation

- Define the error (or cost function) as the squared difference between target $\boldsymbol{t}$ and network output $\boldsymbol{a}$:

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

- 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" (this is similar to Stochastic Gradient Descent).

Backpropagation

- η Is the *learning rate*, which specifies how fast "*learning*," *i.e.* convergence to an optimum, occurs.
- If η is too large, convergence is fast, but coarse, and large steps can overshoot the desired minimum in the error surface.
- If η is too small, we may be able to accurately locate the desired (global) minimum, but convergence can be slow (small steps). Moreover, we could get stuck in local minima and miss the global minimum.
- Choosing η is often a trial-and-error process; it can be automated and adaptive, however.
- Problems with plain-vanilla *Backprop* include slow convergence, and getting stuck in local minima (missing the global minimum).
- Plain-vanilla Backprop is still very useful in the presence of noisy or incomplete training data!
- Backprop is *parallelizable*.

Questions/Comments

- …