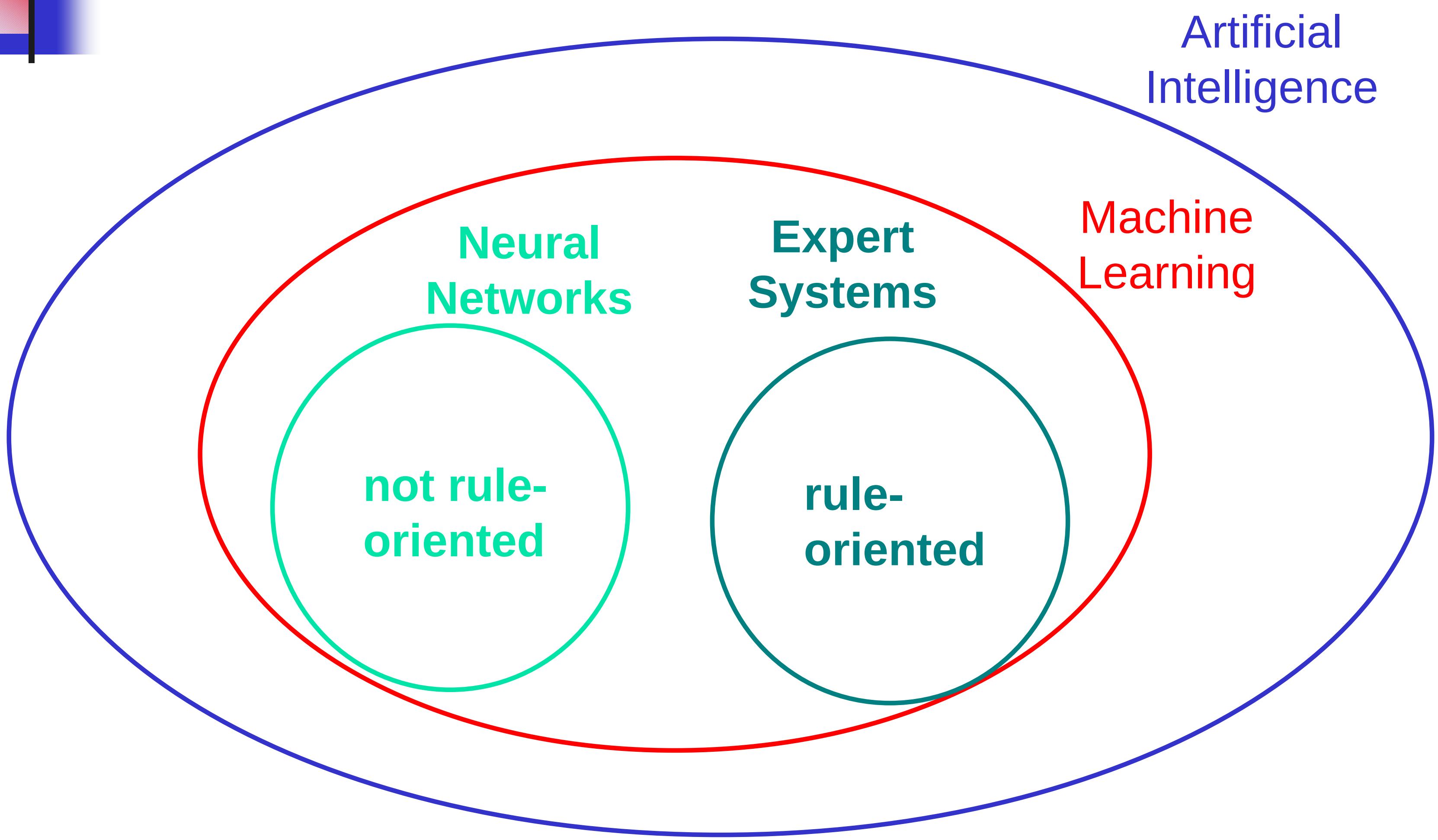
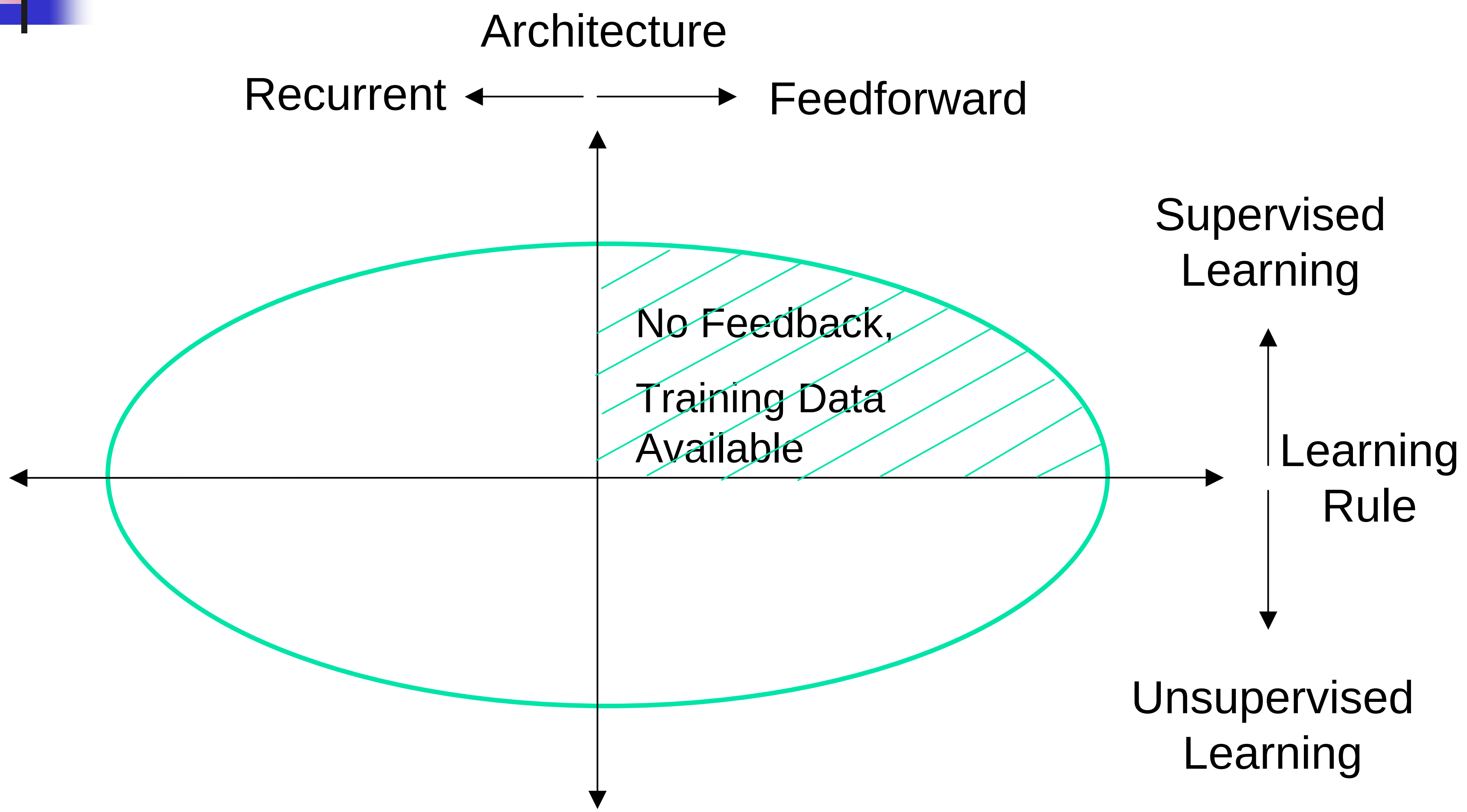
# Introduction to Neural Networks for Senior Design

# Neural Networks: The Big Picture



# Types of Neural Networks

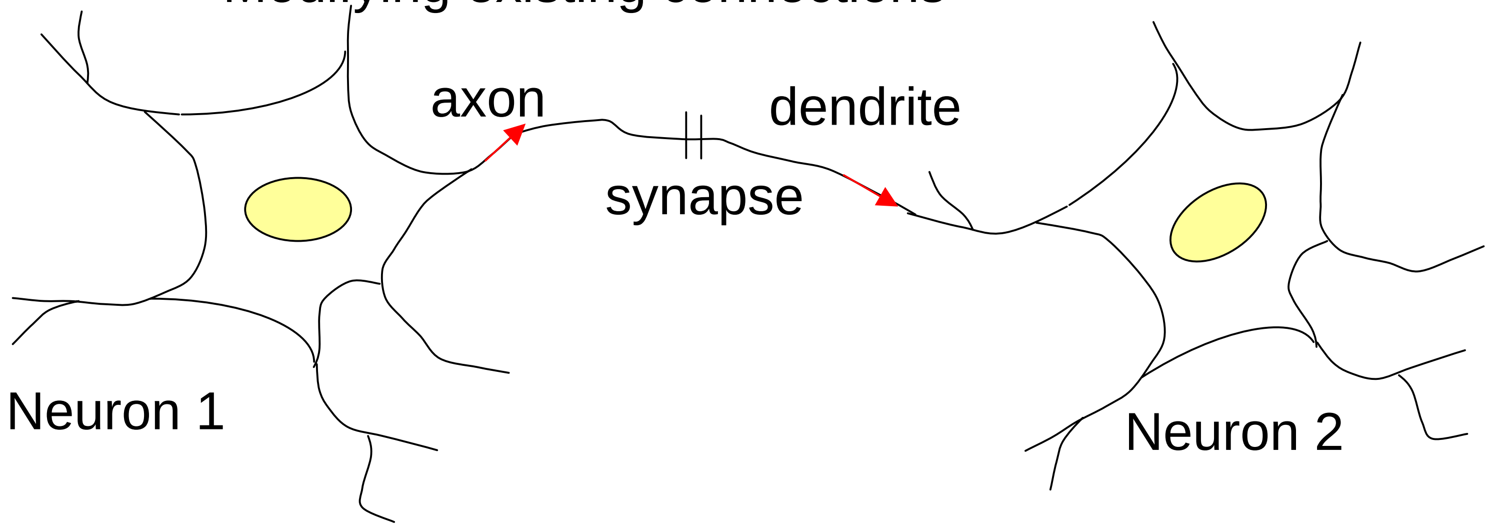


# What Is a Neural Network?

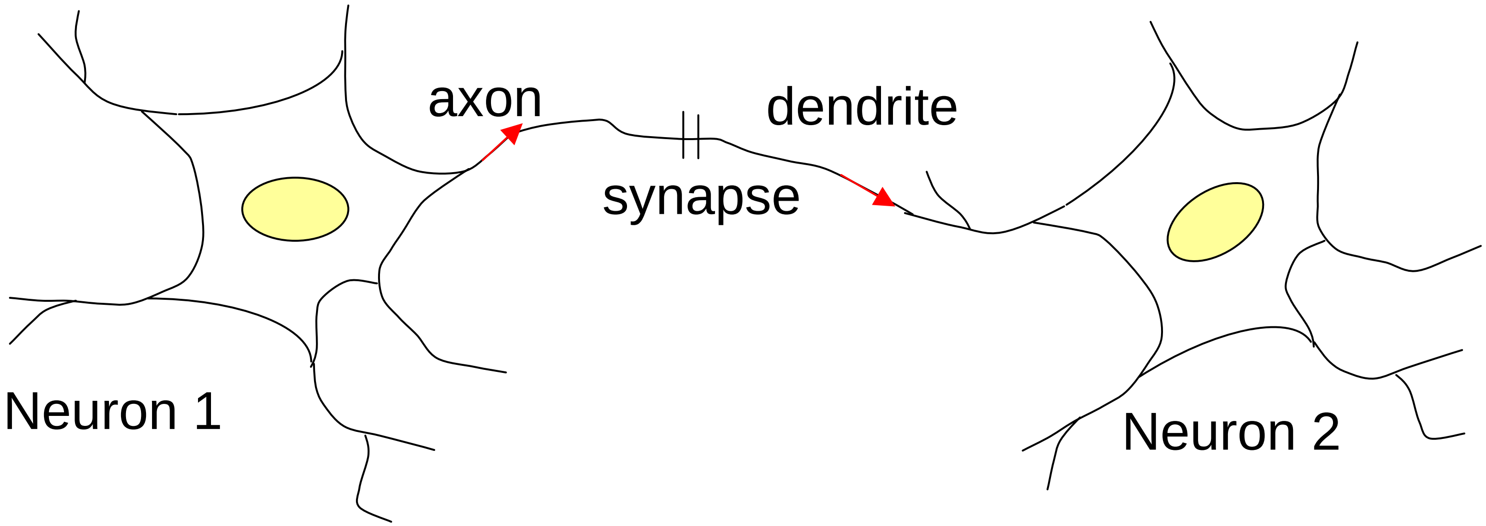
- (Artificial) neural network, or (A)NN:
  - Information processing system loosely based on the model of biological neural networks
  - Implemented in software or electronic circuits
  - Defining properties
    - Consists of simple building blocks (neurons)
      - Connectivity determines functionality
    - Must be able to learn
    - Must be able to generalize

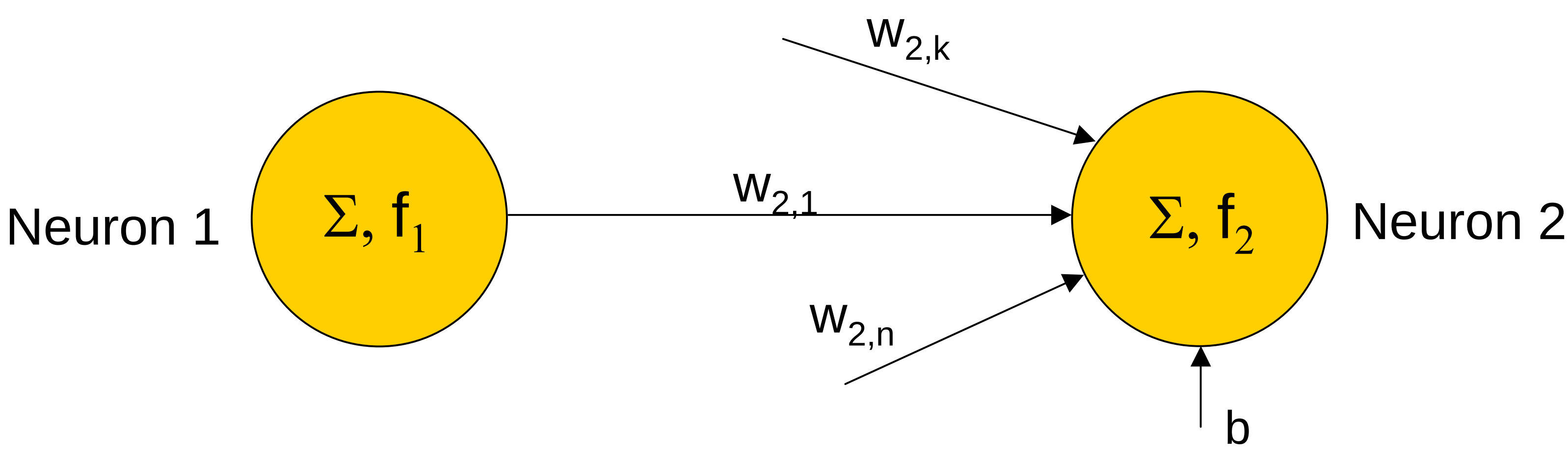
# Biological Inspiration for Neural Networks

- Human Brain: $\approx 10^{11}$ neurons (or nerve cells)
  - Dendrites: incoming extensions, carry signals in
  - Axons: outgoing extensions, carry signals out
  - Synapse: connection between 2 neurons
- Learning:
  - Forming new connections between the neurons, or
  - Modifying existing connections

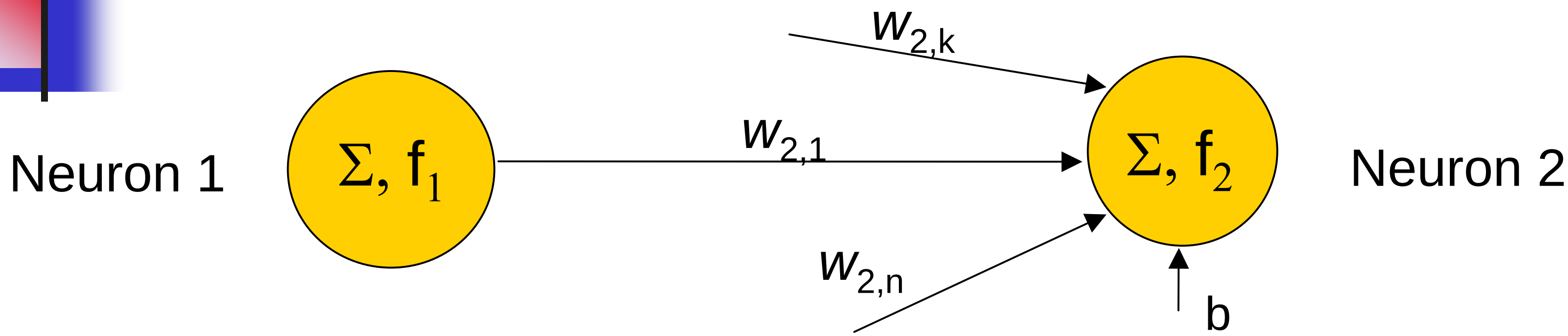


# From Biology to the Artificial Neuron, 1



# From Biology to the Artificial Neuron, 2




- The weight *w* models the synapse between two biological neurons.
- Each neuron has a threshold that must be met to activate the neuron, causing it to “fire.” The threshold is modeled with the transfer function, f.
- Neurons can be
  - excitatory, causing other neurons to fire when they are stimulated; or
  - inhibitory, preventing other neurons from firing when they are stimulated.

# Applications of Neural Networks

- Aerospace: aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations
- Banking: credit application evaluators
- Defense: guidance and control, **target detection and tracking**, object discrimination, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification
- Financial: real estate appraisal, loan advisor, mortgage screening, stock market analysis, stock trading advisory systems
- Manufacturing: process control, process and machine diagnosis, visual quality inspection systems, computer chip quality analysis

# Applications of Neural Networks, cont.'

- Medical: cancer cell detection and analysis, EEG and ECG analysis, disease pathway analysis
- Communications: adaptive echo cancellation, image and data compression, speech synthesis, signal filtering
- Robotics: Trajectory control, manipulator controllers, vision systems
- Pattern Recognition: character recognition, speech recognition, voice recognition, facial recognition

# Problems Suitable for Solution by NN's

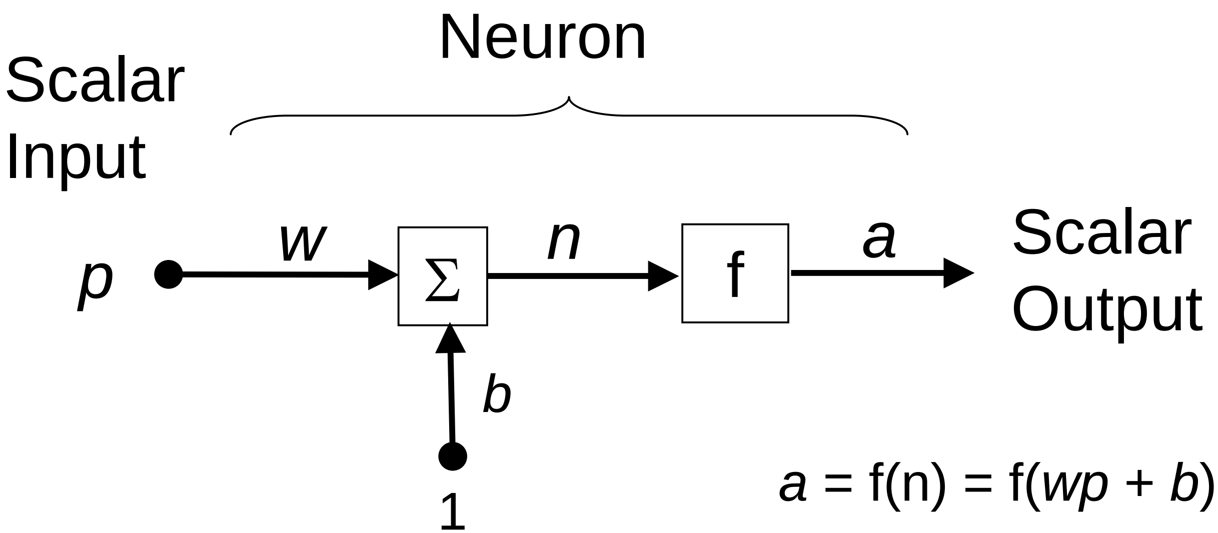
- Problems for which there is no clear-cut rule or algorithm
  - e.g., image interpretation
- Problems that humans do better than traditional computers
  - e.g., facial recognition
- Problems that are intractable due to size
  - e.g., weather forecasting

# Math Notation/Conventions

- Scalars: small italic letters
  - e.g., *p, a, w*
- Vectors: small, bold, non-italic letters
  - e.g., **p**, **a**, **w**
- Matrices: capital, bold, non-italic letters
  - e.g., **P**, **A**, **W**
- Assume vectors are column vectors (unless stated otherwise)
  - e.g., $\mathbf{p} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

# Network Architecture and Notation

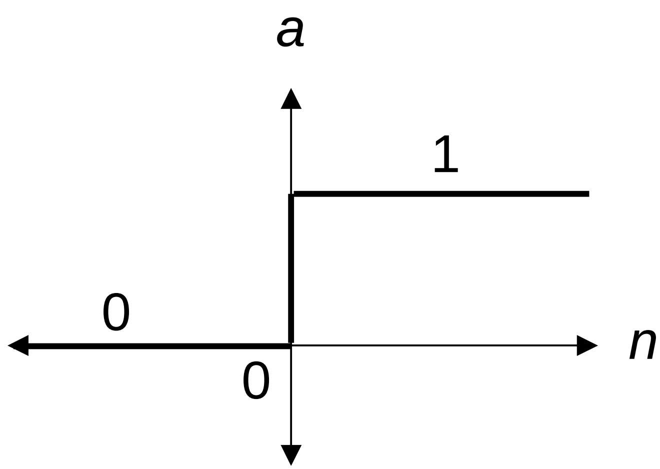
- Single-Input Neuron




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

- Network Parameters (weight: *w*, bias: *b*)
  - Adjusted via learning rule
- Net Input: *n*
- Transfer Function, f  (design choice)

# Transfer Functions – Hard Limiter

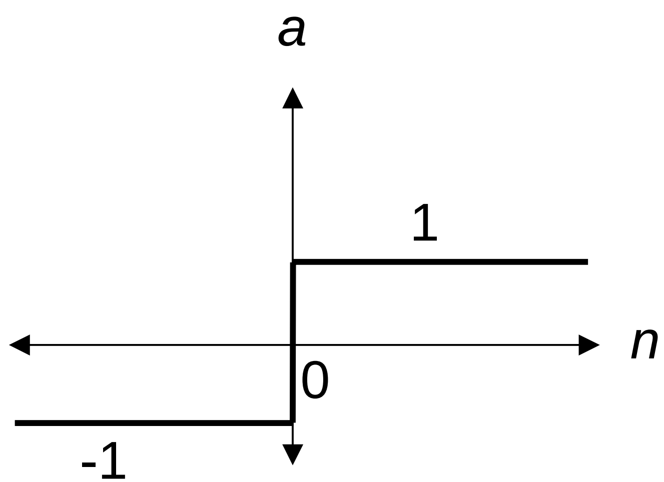



$$a = f(n) = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases}$$

MATLAB: *a* = hardlim(*n*)

(often used for binary classification problems)

# Transfer Functions – Symmetric Hard Limiter

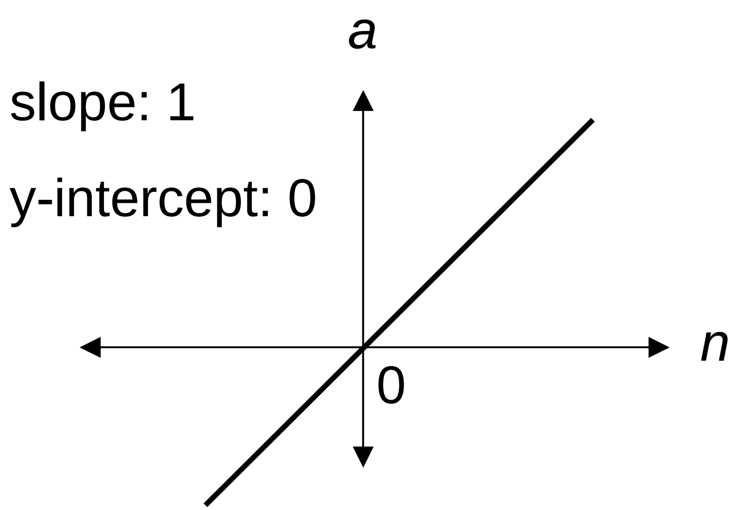



$$a = f(n) = \begin{cases} -1, & n < 0 \\ 1, & n \geq 0 \end{cases}$$

MATLAB: $a$ = hardlims($n$)

(often used for binary classification problems)

# Transfer Functions - Linear

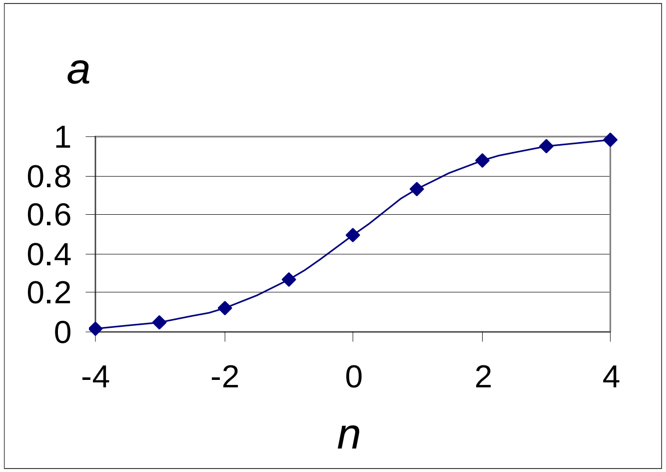



$a = \mathrm{f}(n) = n$

MATLAB: *a* = purelin(*n*)

(often used in network training for classification problems)

# Transfer Functions – Log-Sigmoid




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

MATLAB: $a$ = logsig($n$)

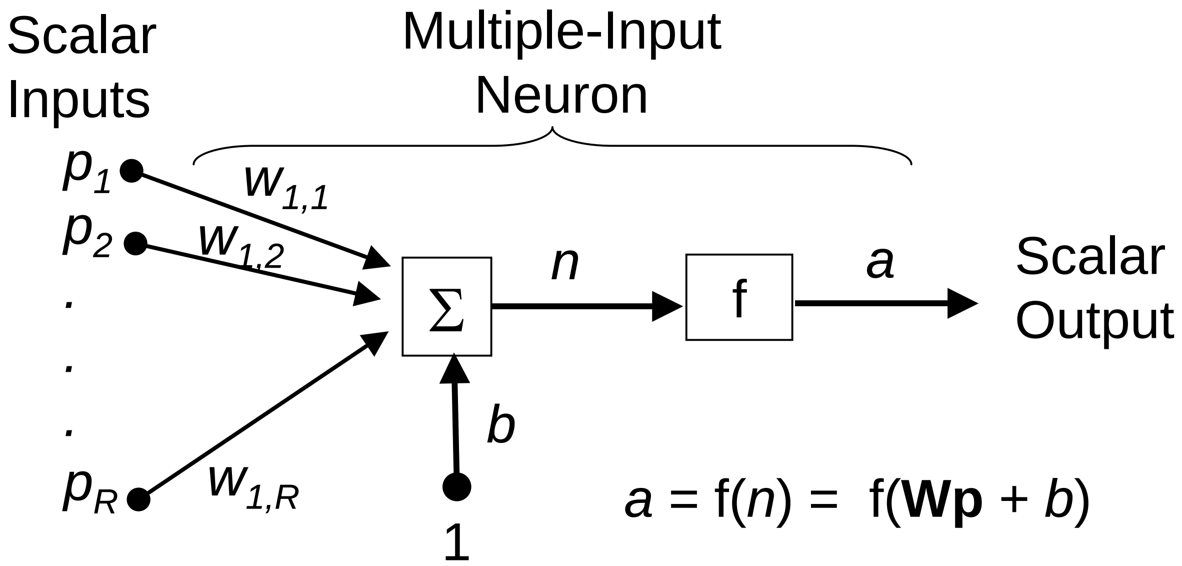
(often used for training multi-layer networks with backpropagation)

# Transfer Function Summary

| Function | Equation | Output Range | MATLAB |
|---|---|---|---|
| Hard limiter | $a = 0, \quad n < 0$<br>$\quad 1, \quad n \geq 0$ | Discrete: 0 or 1 | hardlim |
| Symmetric Hard Limiter | $a = -1, \quad n < 0$<br>$\quad 1, \quad n \geq 0$ | Discrete: 1, -1 | hardlims |
| Linear | $a = n$ | Continuous: range of n | purelin |
| Log-Sigmoid | $a = 1 / (1 + e^{-n})$ | Continuous: (0, 1) | logsig |
| Hyperbolic Tangent Sigmoid | $a = \frac{(e^{n} - e^{-n})}{(e^{n} + e^{-n})}$ | Continuous: (-1, 1) | tansig |
| Competitive | $a$ = 1, neuron w/<br>max n<br>(0 else) | Discrete: 0, 1 | compet |

# Multiple-Input Neurons

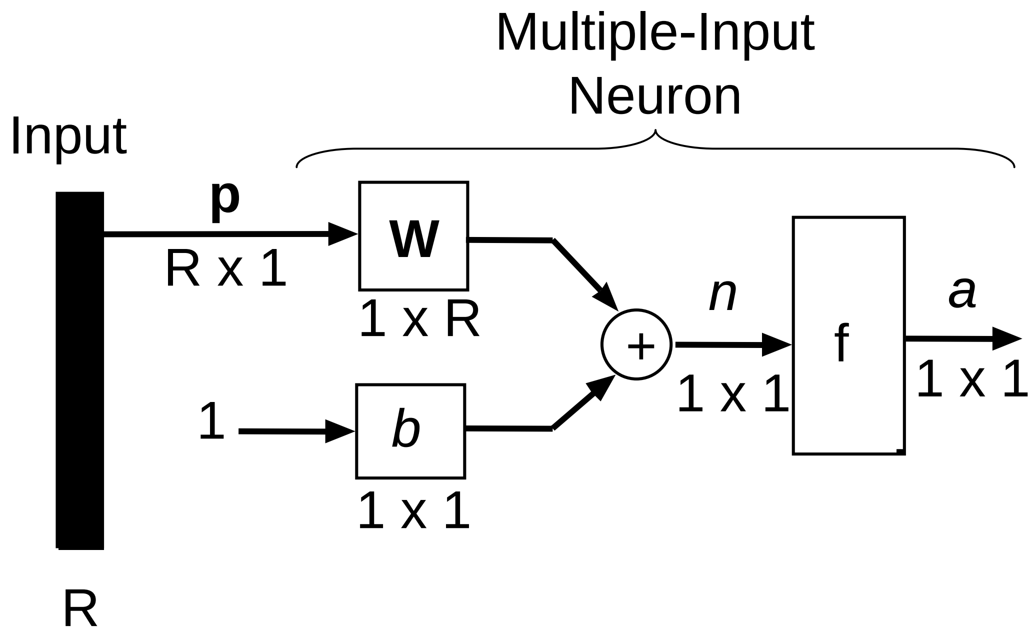
- Consider a network with R scalar inputs: $p_1, p_2, \ldots, p_R$ :




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

- where **W** is the weight matrix with one row: $\mathbf{W} = [w_{1,1} \; w_{1,2} \; \ldots \; w_{1,R}]$
- and **p** is the column vector: $\mathbf{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_R \end{bmatrix}$

# Multiple-Input Neurons: Matrix Form




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

# Binary Classification Example [Ref. #5]

- Consider the patterns:

- Features: # of vertices*, # of holes, symmetry (vertical or horizontal)

yes(1) or no (0)

- Noise-free observation vectors or feature vectors

$$A \Leftrightarrow \begin{bmatrix}4\\1\\1\end{bmatrix}, \qquad F \Leftrightarrow \begin{bmatrix}2\\0\\0\end{bmatrix}$$

---

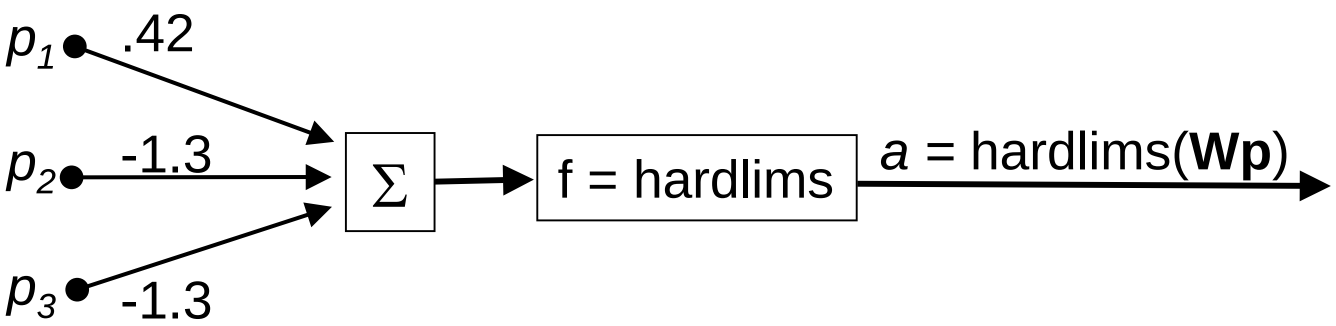
* Here a vertex is an intersection of 2 or more lines.

# Binary Classification Example, continued

- Targets (desired outputs for each input):

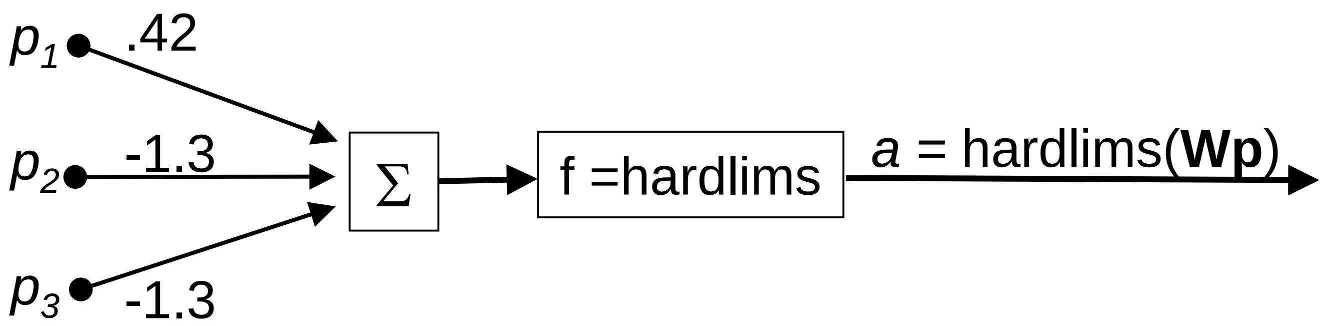
$t_1 = -1$ (for "A")    $t_2 = 1$ (for "F")

- Neural Network (Given):




**W** = [.42   -1.3   -1.3]

# Binary Classification: Sample Calculation




**W** = [.42 -1.3 -1.3]

Calculating the neuron output if the input pattern is the letter “A”:

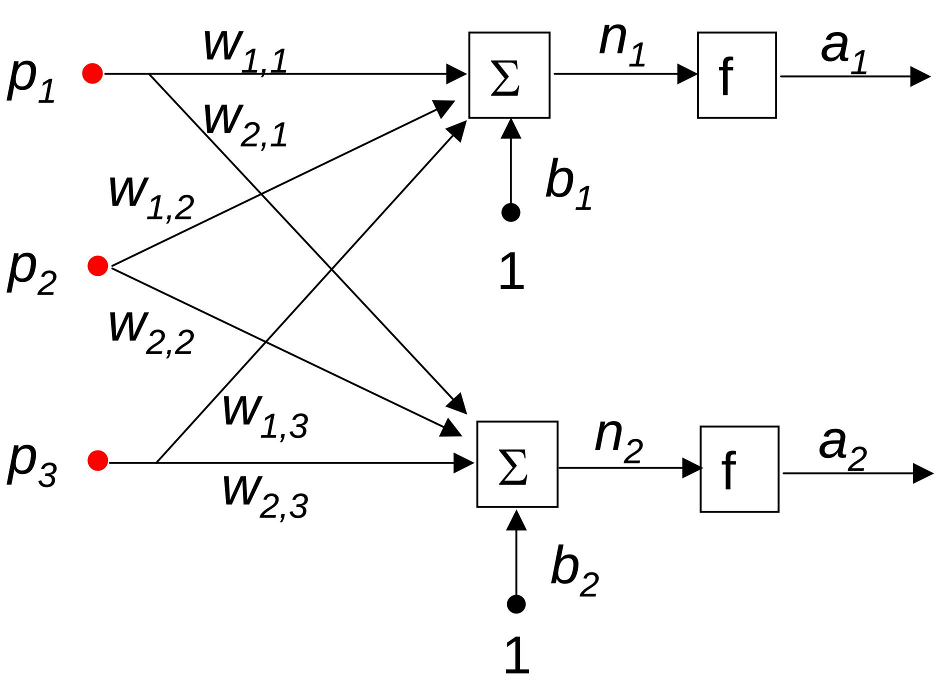
$$\mathbf{Wp} = [.42 \quad -1.3 \quad -1.3] \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix} = -.92$$

$a$ = hardlims(**Wp**) = -1

Find **Wp** and the neuron output $a$ if the input pattern is “F”?

# A Layer of 2 Neurons (Single-Layer Network)

- Sometimes we need more than one neuron to solve a problem.
- Example consider a problem with 3 inputs and 2 neurons:




$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \mathrm{f}(\mathbf{W}:\mathrm{row}(1)\cdot\mathbf{p} + b_1) \\ \mathrm{f}(\mathbf{W}:\mathrm{row}(2)\cdot\mathbf{p} + b_2) \end{bmatrix}$$

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

# Weight Matrix Notation

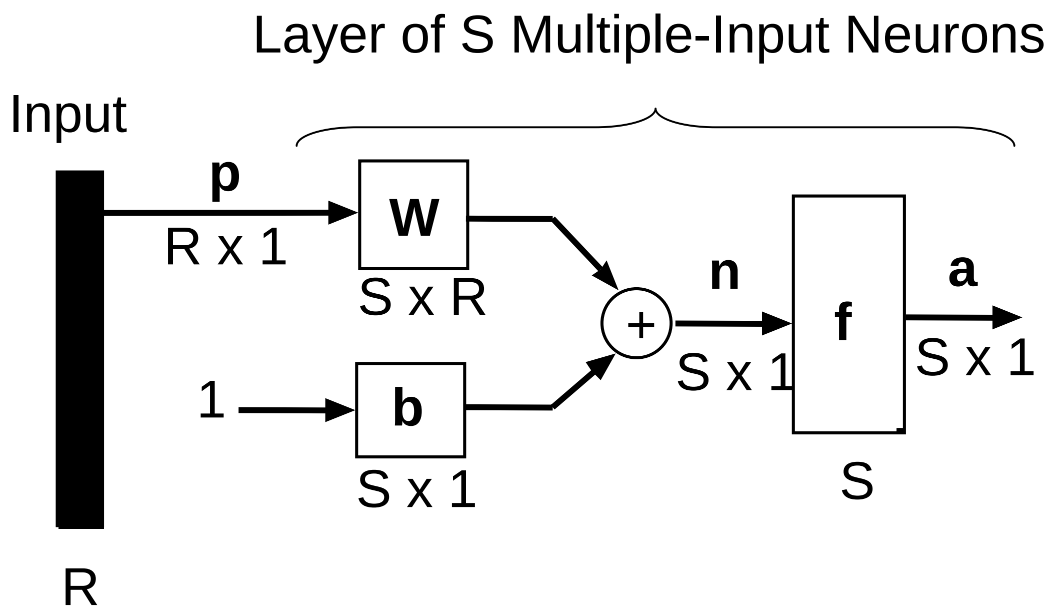
- Recall for our single neuron with multiple inputs, we used weight matrix **W** with one row: **W** = $[w_{1,1} \ \ w_{1,2} \ \ \ldots \ w_{1,R}]$

- General Case (multiple neurons): components of **W** are weights connecting some input element to the summer of some neuron

- Convention (as used in Hagan), for component $w_{i,j}$ of **W**
  - **First index** (i) indicates the **neuron #** the input is entering
    - the "to" index
  - **Second index** (j) indicates the **element # of input vector p** that will be entering the neuron
    - the "from" index"

$$\boxed{w_{i,j} = w_{to,from}}$$

# Single Layer of S Neurons: Matrix Form

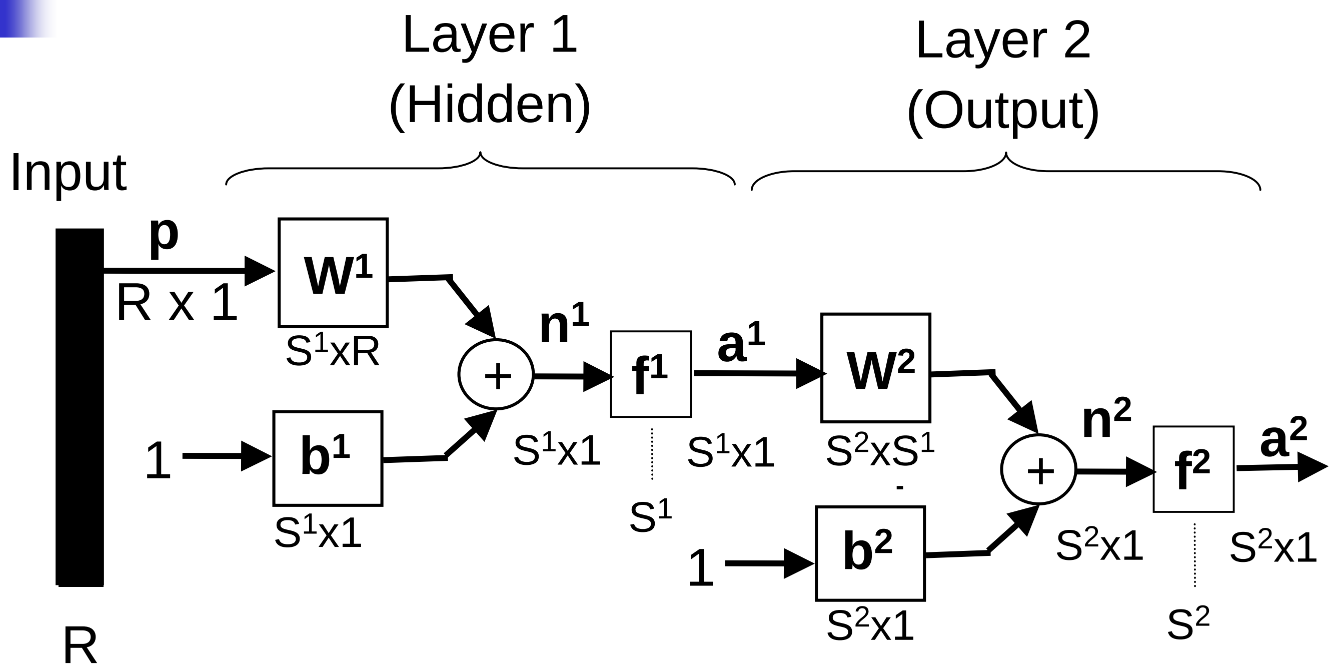



$$\mathbf{a} = \mathbf{f}(\mathbf{Wp} + \mathbf{b})$$

# Multiple Layers of Neurons

- Allow each layer to have its own weight matrix (**W**), bias vector (**b**), net input vector (**n**), output vector (**a**), and # of neurons (S)
- Notation: superscript on the variable name indicates the layer #:
  - e.g., $\mathbf{W}^1$: weight matrix for layer #1, $\mathbf{b}^2$: indicates bias vector for layer #2, $\mathbf{a}^3$: output vector for layer #3
  - e.g., $S^4$: # of neurons in the 4th layer
- Output of layer 1 is input to layer 2, etc.
- The last (right-most) layer of the network is called the output layer; the inputs are not counted as a layer at all (per Hagan); layers between the input and output are called hidden layers.

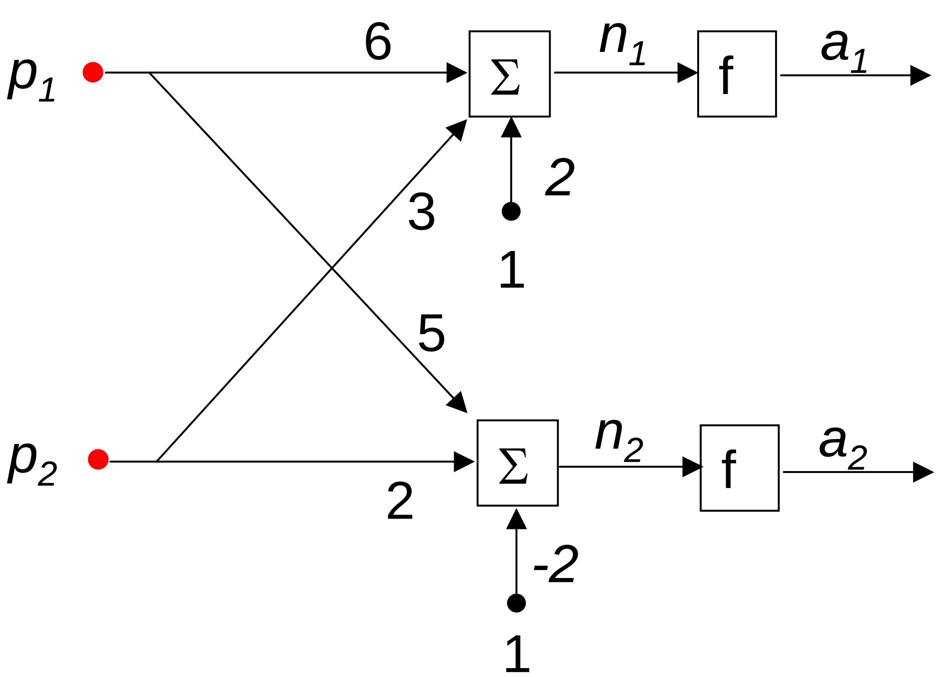
# 2-Layer Network: Matrix Form




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

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

# Introduction: Practice Problem



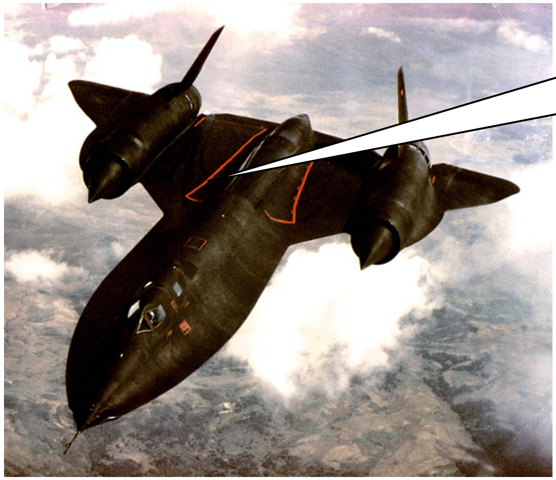

1) For the neural network shown, find the weight matrix **W** and the bias vector **b**.

2) Find the output if f = "compet" and the input vector is $\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

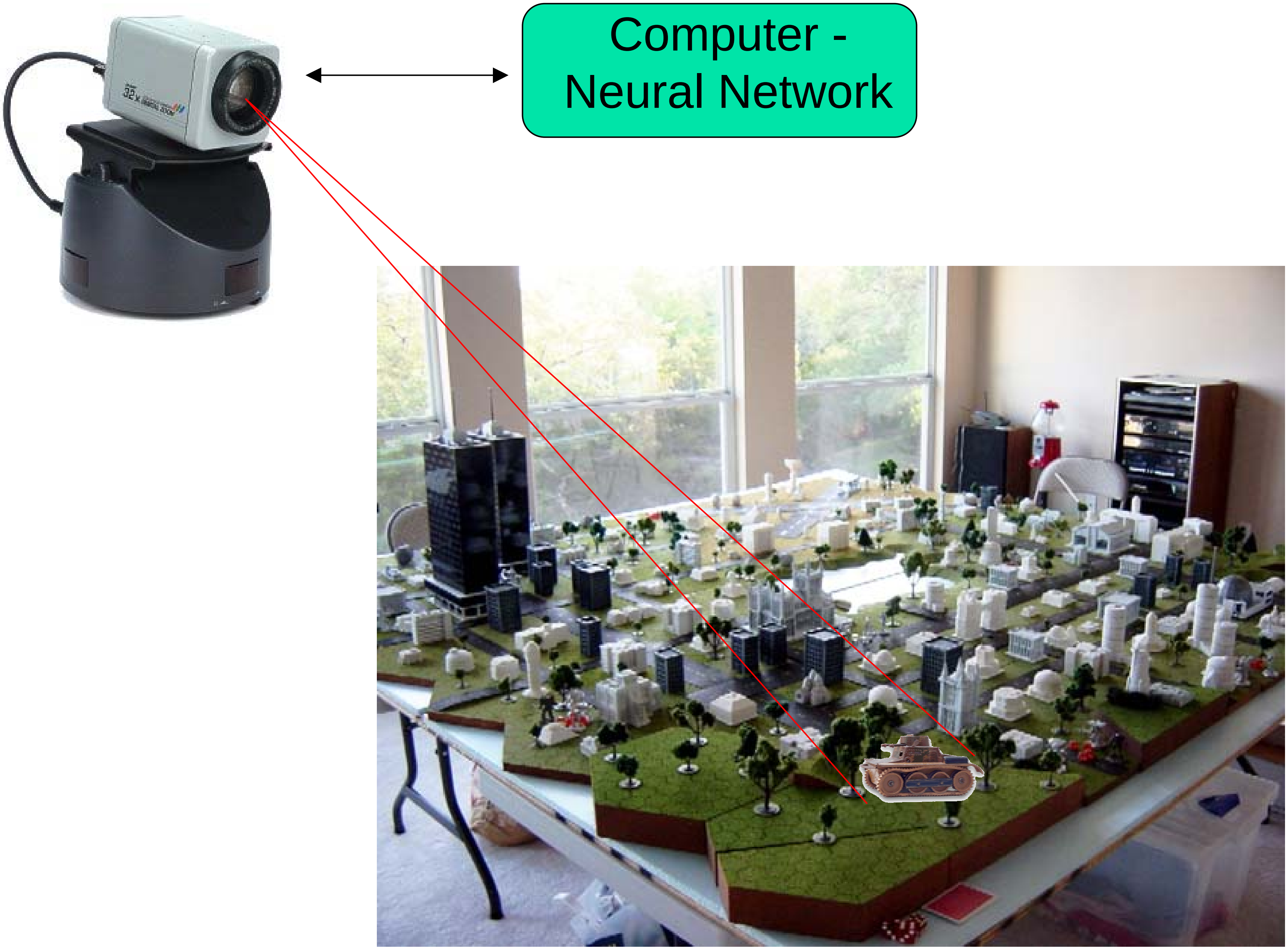
**a** = **compet**(**Wp** + **b**)

where compet($n$) = 1, neuron w/max n
0, else

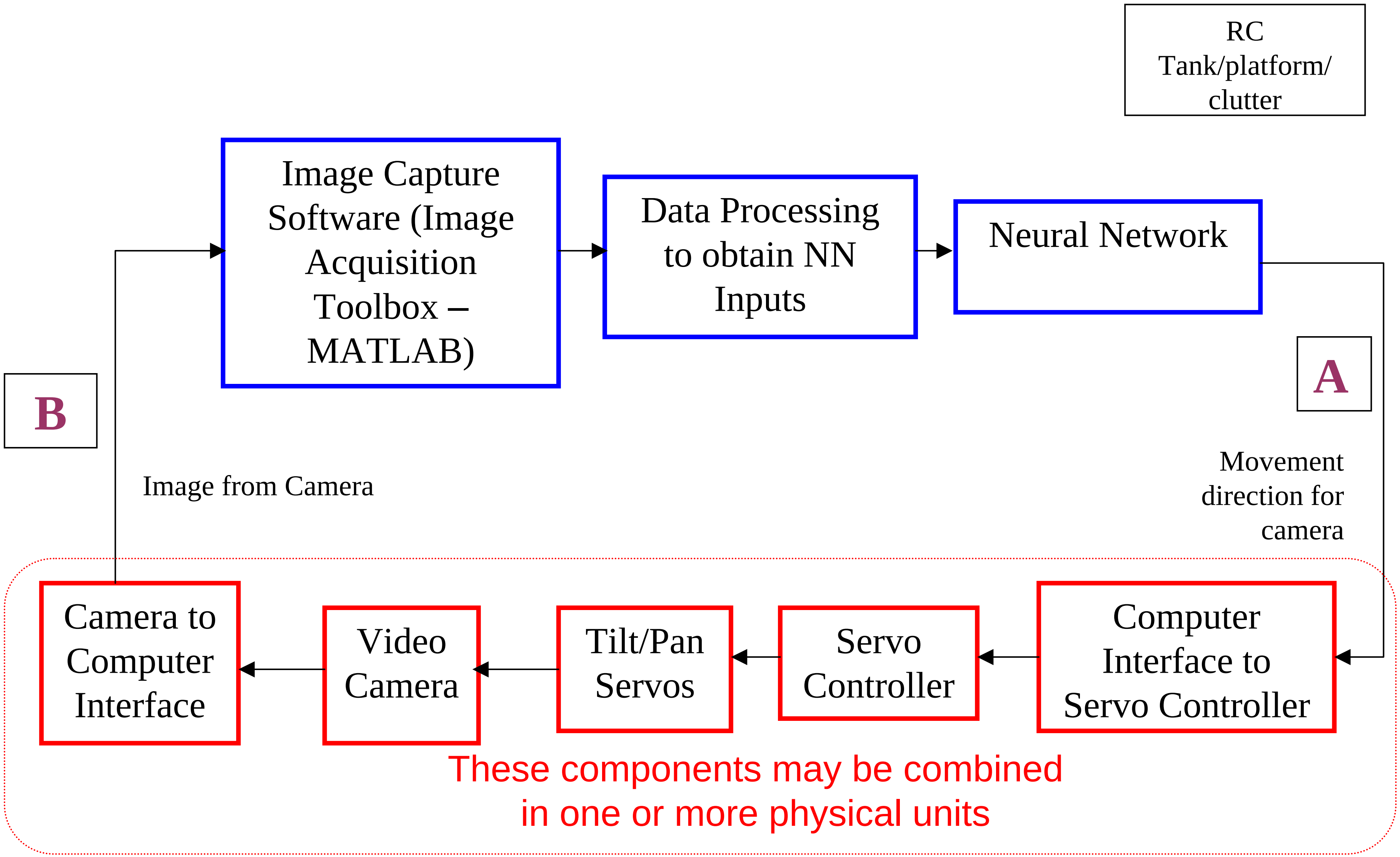
# Project Description



Is that a tank
or a tree?


Computer -
Neural Network

# Project Overview




Phase 1: How do we get from A to B?

Phase 2 (main project): How to we get from B to A?