CSSE463: Image Recognition Day 17

- See schedule for reminders

Neural networks

- “Biologically inspired” model of computation
- Can model arbitrary real-valued functions for classification and association between patterns
- *Discriminative model*
  - Models decision boundary directly
  - Less memory than nearest neighbor
  - Fast!
- Can be parallelized easily for large problems
- We will take a practical approach to classification

Perceptron model

- Computational model of a single neuron
  - Inputs
  - Outputs
  - Function and threshold

- Will be connected to form a complete network

Example: Modeling logic gates

- We'll do OR together.
  - Inputs: $x_1 = \{0,1\}$, $x_2 = \{0,1\}$
  - We need to pick weights w_i and x_0 (= -t, the threshold) such that it outputs 0 or 1 appropriately

- Quiz: You do AND, NOT, and XOR.
- Note that a single perceptron is limited in what it can classify. What is the limitation?

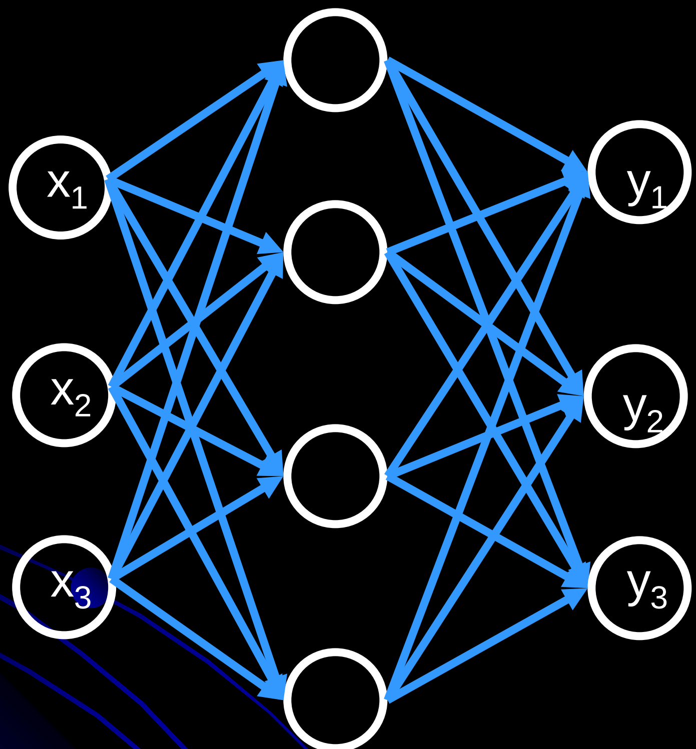
Perceptron training

- Each misclassified sample is used to change the weight "a little bit" so that the classification is better the next time.
- Consider inputs in form $x = [x_1, x_2, \ldots x_n]$
- Target label is $y = \{+1,-1\}$

Algorithm (Hebbian Learning)

- Randomize weights
- Loop until converge
  - If $wx + b > 0$ and y is -1:
    - w_i -= $\varepsilon * x_i$ for all *i*
    - b -= εy
  - else if $wx + b < 0$ and y is +1:
    - w_i += $\varepsilon * x_i$ for all *i*
    - b += εy
  - Else (it's classified correctly, do nothing)
  - ε is the learning rate (a parameter that can be tuned).

Multilayer feedforward neural nets

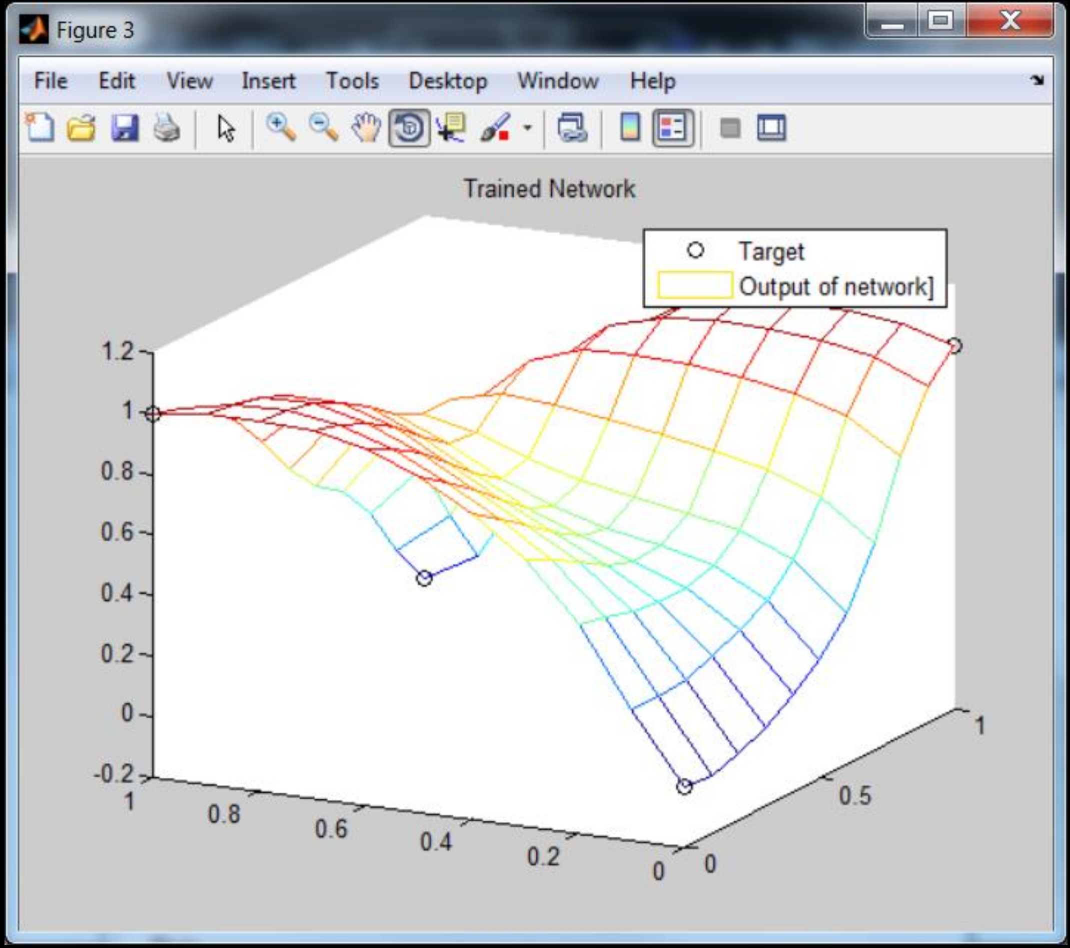


Sensory (HSV) | Hidden (functions) | Classification (apple/orange/banana)

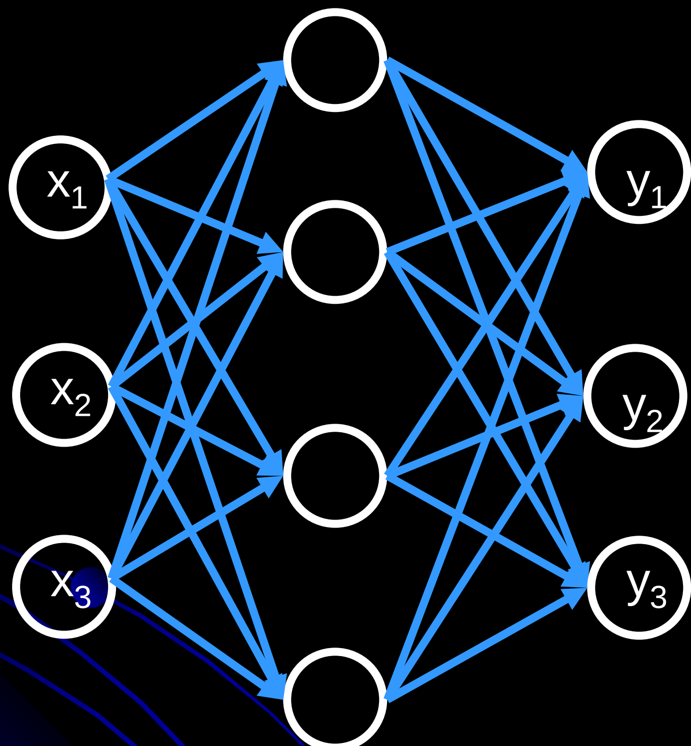
- Many perceptrons
- Organized into layers
  - Input (sensory) layer
  - Hidden layer(s): 2 proven sufficient to model any arbitrary function
  - Output (classification) layer
- Powerful!
- Calculates functions of input, maps to output layers
- Example

XOR example

- 2 inputs
- 1 hidden layer of 5 neurons
- 1 output



Backpropagation algorithm



Initialize all weights randomly

- For each labeled example:
  - Calculate output using current network
  - Update weights across network, from output to input, using Hebbian learning
- Iterate until convergence
  - Epsilon decreases at every iteration

- Matlab does this for you. ☺
- matlabNeuralNetDemo.m

a. Calculate output (feedforward)

Repeat

b. Update weights (feedback)

Parameters

- Most networks are reasonably robust with respect to learning rate and how weights are initialized
- However, figuring out how to
  - normalize your input, and
  - determine the architecture of your net
- is a black art. You might need to experiment. One hint:
  - Re-run network with different initial weights and different architectures, and test performance each time on a validation set. Pick best.