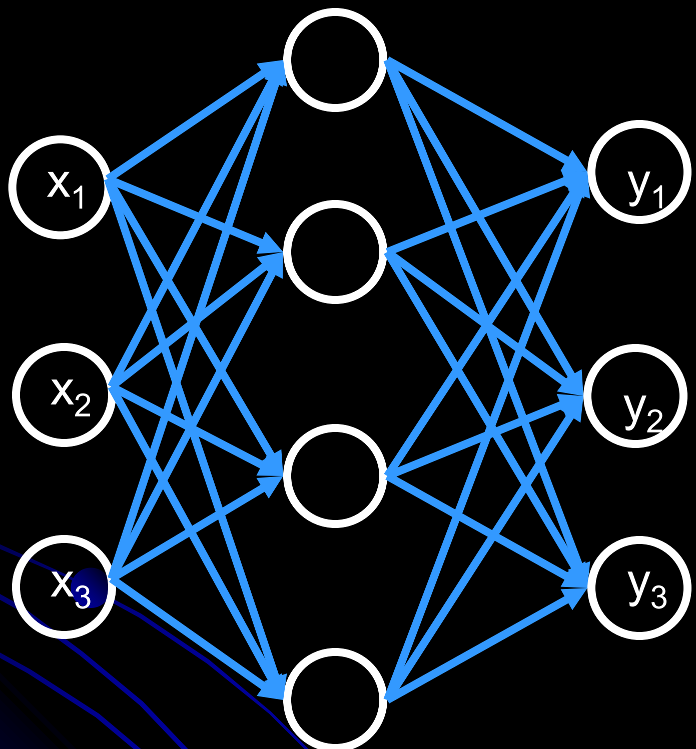
CSSE463: Image Recognition Day 18

- Upcoming schedule:
  - Lightning talks shortly
  - Midterm exam Monday
  - Sunset detector due Wednesday

Multilayer feedforward neural nets

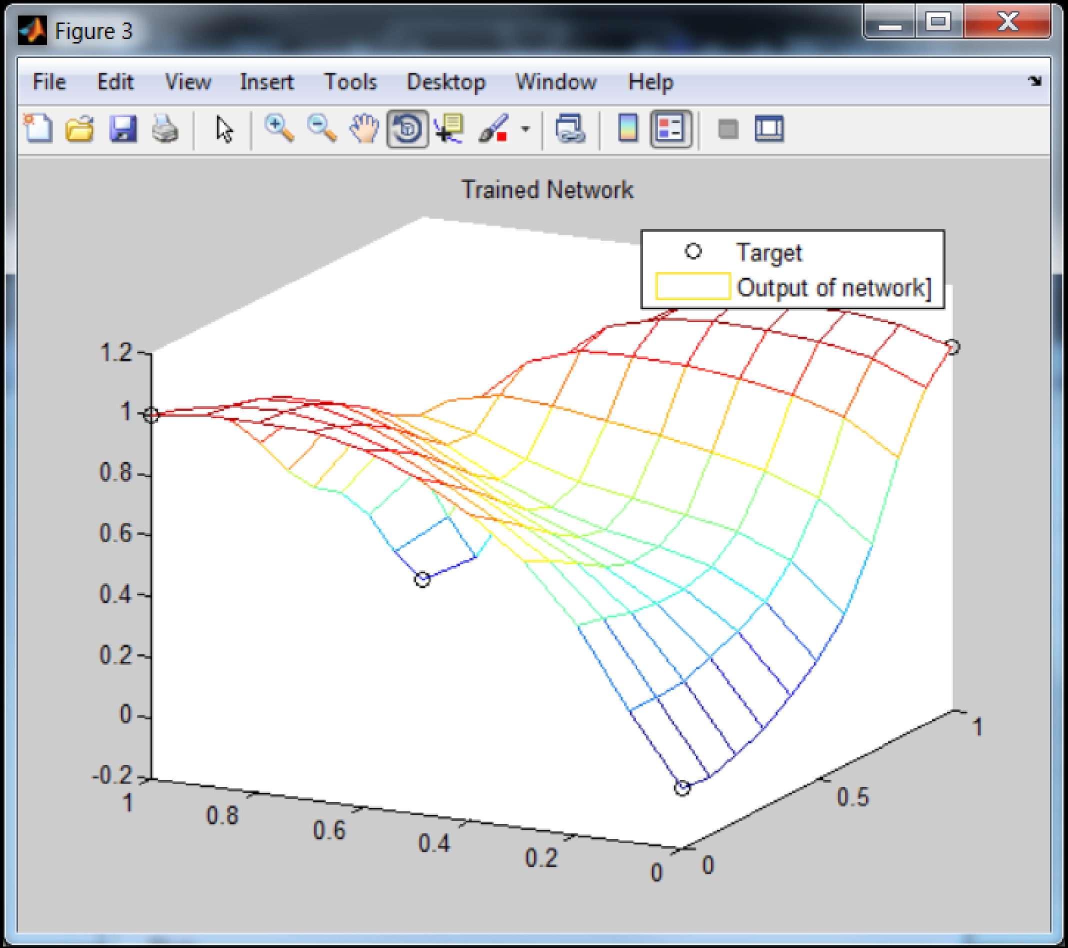


Sensory (HSV) | Hidden (functions) | Classification (apple/orange/banana)

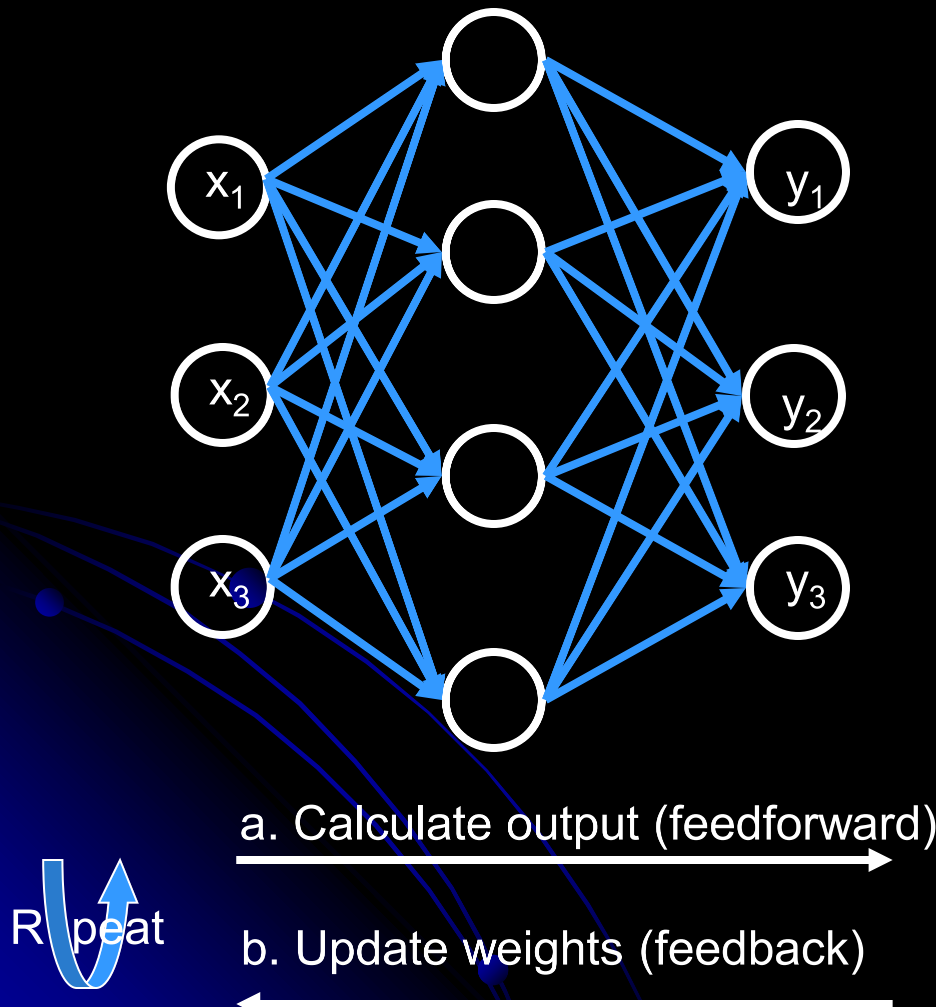
- Many perceptrons
- Organized into layers
  - Input (sensory) layer
  - Hidden layer(s): 2 proven sufficient to model any arbitrary function
  - Output (classification) layer
- Powerful!
- Calculates functions of input, maps to output layers
- Example

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

Parameters

- Most networks are reasonably robust with respect to learning rate and how weights are initialized
- However, figuring out how to
  - normalize your input
  - 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.

References

- This is just the tip of the iceberg! See:
  - Sonka, pp. 404-407
  - Laurene Fausett. *Fundamentals of Neural Networks*. Prentice Hall, 1994.
    - Approachable for beginner.
  - C.M. Bishop. *Neural Networks for Pattern Classification*. Oxford University Press, 1995.
    - Technical reference focused on the art of constructing networks (learning rate, # of hidden layers, etc.)
  - Matlab neural net help

SVMs vs. Neural Nets

- SVM: Training can be expensive
  - Training can take a *long* time with large data sets. Consider that you'll want to experiment with parameters…
  - But the classification runtime and space are *O(sd)*, where *s* is the number of support vectors, and d is the dimensionality of the feature vectors.
  - In the worst case, *s* = size of whole training set (like nearest neighbor)
  - But no worse than implementing a neural net with *s* perceptrons in the hidden layer.
  - Empirically shown to have good generalizability even with relatively-small training sets and no domain knowledge.
- Neural networks: can tune architecture.

Q3

How does svmfwd compute y1?

y1 is just the weighted sum of contributions of individual support vectors:
d = data dimension, e.g., 294, σ = kernel width.

$$y1 = \sum_{i=1}^{numSupVecs} \left(svcoeff_i * e^{(-1/d\sigma)*\|x - sv_i\|^2} \right) + bias$$

numSupVecs, svcoeff (alpha) and bias are learned during training.
Note: looking at which of your training examples are support vectors can be revealing! (Keep in mind for sunset detector and term project)

- Much easier computation than training
- Could implement on a device without MATLAB (*e.g.*, a smartphone) easily