CMSC 726
Lecture 9: Neural Networks

Lise Getoor

September 28, 2010

ACKNOWLEDGEMENTS: The material in this course is a synthesis of materials from many sources, including: Hal Daume III, Mark Drezde, Carlos Guestrin, Andrew Ng, Ben Taskar, Eric Xing, and others. I am very grateful for their generous sharing of insights and materials.

Today's Topics

- Neural Networks
  - History and Motivation
  - Threshold units
  - Gradient descent
  - Multilayer networks
  - Backpropagation
- Throughout, lots of examples….

Connectionist Models

Consider humans

- Neuron switching time ~.001 second
- Number of neurons ~10^{10}
- Connections per neuron ~10^{4-5}
- Scene recognition time ~.1 second
- 100 inference step does not seem like enough

must use lots of parallel computation!

Properties of artificial neural nets (ANNs):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is *unimportant*

Examples:

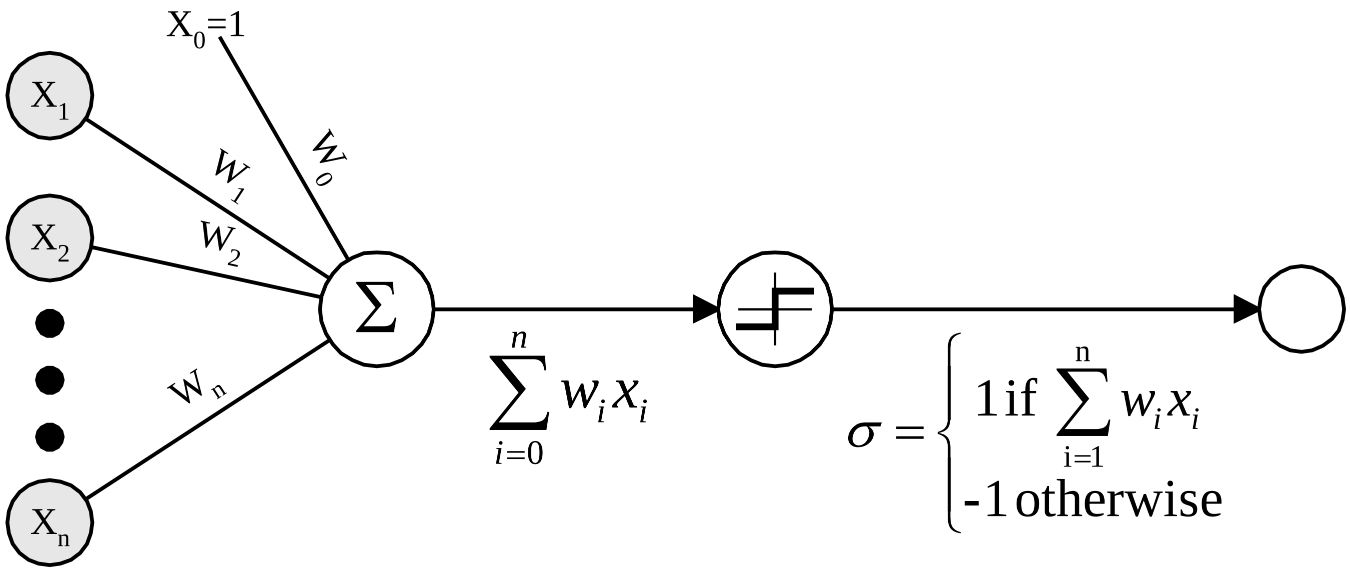
- Speech phoneme recognition
- Image classification
- Financial prediction
- Complex controllers

History

- McColloch & Pitts 1943
- Widrow & Hoff 1960
- Rosenblatt 1962
- Minsky & Papert 1969
- Rumelhart et al. 1986

Perceptron Video

Perceptron

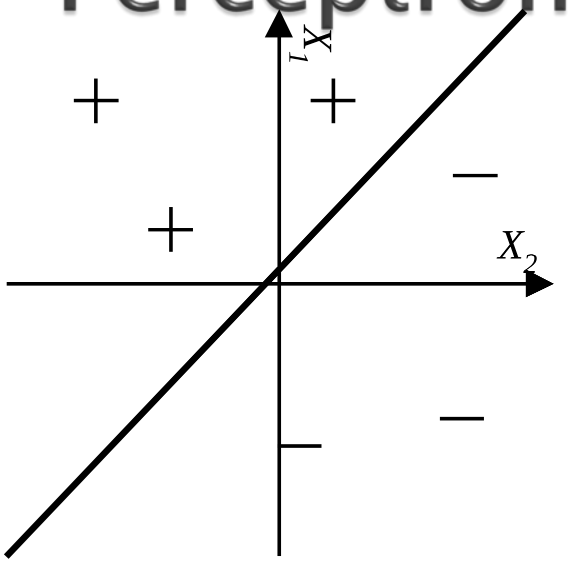


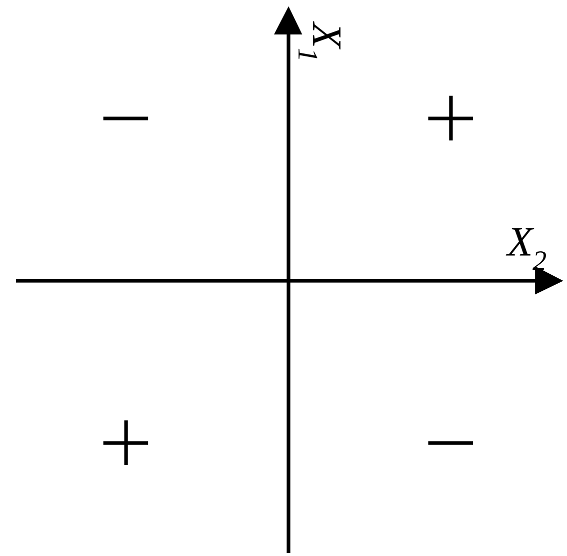
$$o(x_1,\ldots,x_n) = \begin{cases} 1 \text{ if } \; w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

in other words:

$$o(\vec{x}) \; = \; \begin{cases} 1 \text{ if } \; \vec{w} \cdot \vec{x} > 0 \\ -1 \text{ otherwise} \end{cases}$$

Decision Surface of Perceptron





Represents some useful functions

- What weights represent $g(x_1,x_2) = AND(x_1,x_2)$?

But some functions not representable

- e.g., not linearly separable
- therefore, we will want networks of these ...

Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta\,(t - o)x_i$$

- $t =$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called learning rate

Can prove it will converge

- If training data is linearly separable
- and η is sufficiently small

Linear Threshold Unit

To understand, consider simple linear unit, where

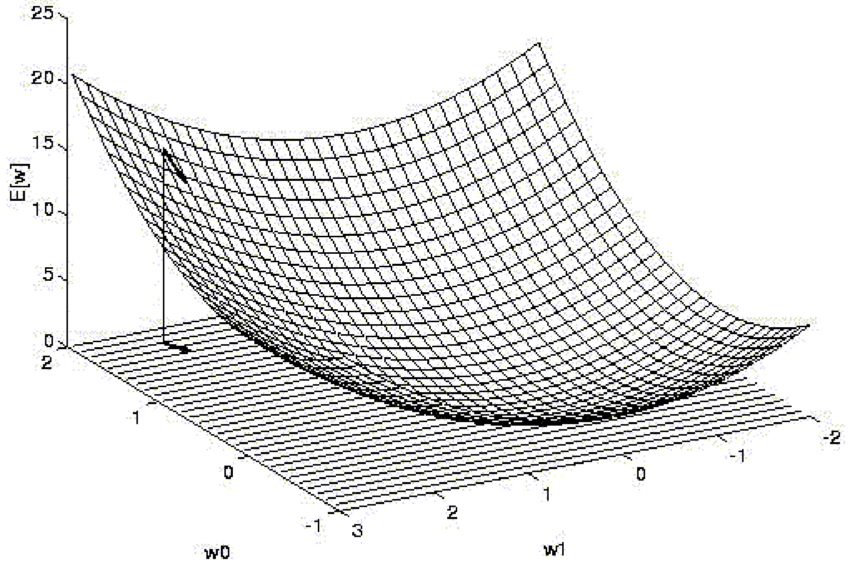
$$o = w_0 + w_1 x_1 + \ldots + w_n x_n$$

Idea : learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \tfrac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is the set of training examples

Gradient Descent



Gradient $\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$

Training rule: $\Delta w_i = -\eta \nabla E[\vec{w}]$

i.e., $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

Gradient Descent

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

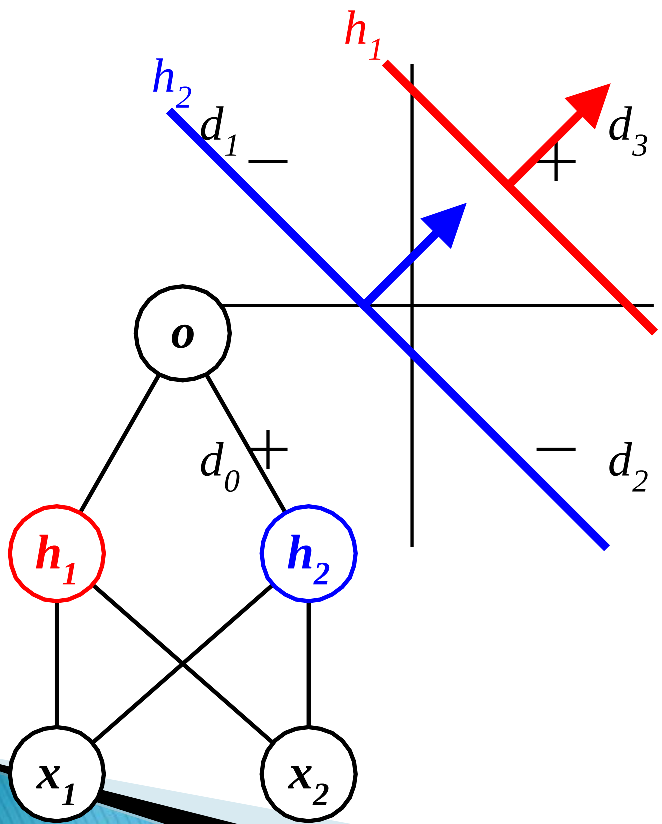
Linear unit training rule uses gradient descent

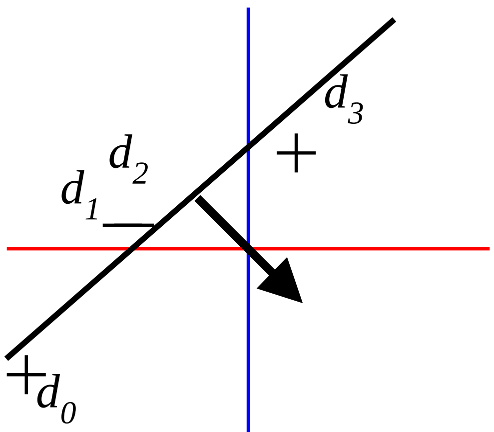
- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Neural Networks II

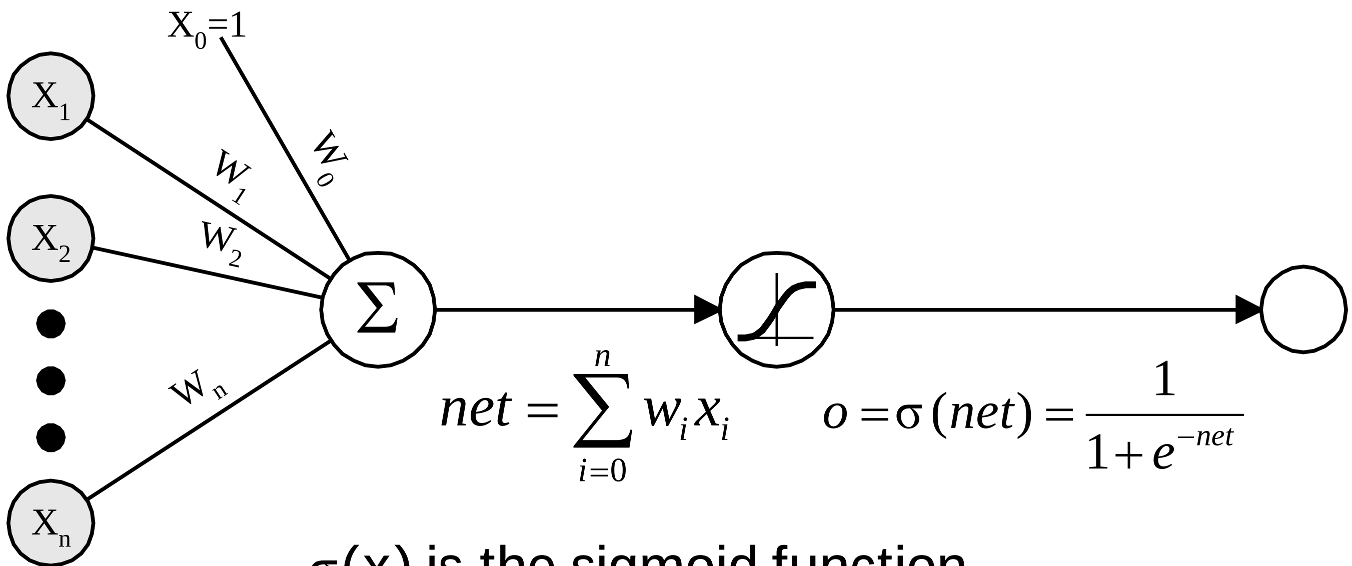
- Multi-layer networks
- History – took off in the 80s
  - Nettalk Video

Nonlinear Models: Multilayer Networks of Sigmoid Units





Sigmoid Unit



$$net = \sum_{i=0}^{n} w_i x_i \qquad o = \sigma(net) = \frac{1}{1+e^{-net}}$$

$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

$$\text{Recall}: \frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$$

We can derive gradient descent rules to train

- One sigmoid unit
- Multilayer networks of sigmoid units $\rightarrow$ Backpropagation

Error Gradient for a Sigmoid Unit

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d (t_d - o_d)\left(-\frac{\partial o_d}{\partial w_i}\right)$$

$$= -\sum_d (t_d - o_d)\frac{\partial o_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}$$

But we know :

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\vec{w}\cdot\vec{x}_d)}{\partial w_i} = x_{i,d}$$

So :

$$\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1 - o_d)x_{i,d}$$

Backpropagation Algorithm

Initialize all weights to small random numbers. Until satisfied, do

- For each training example, do
  1. Input the training example and compute the outputs
  2. For each output unit k

     $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

     $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{k,h}\delta_k$$

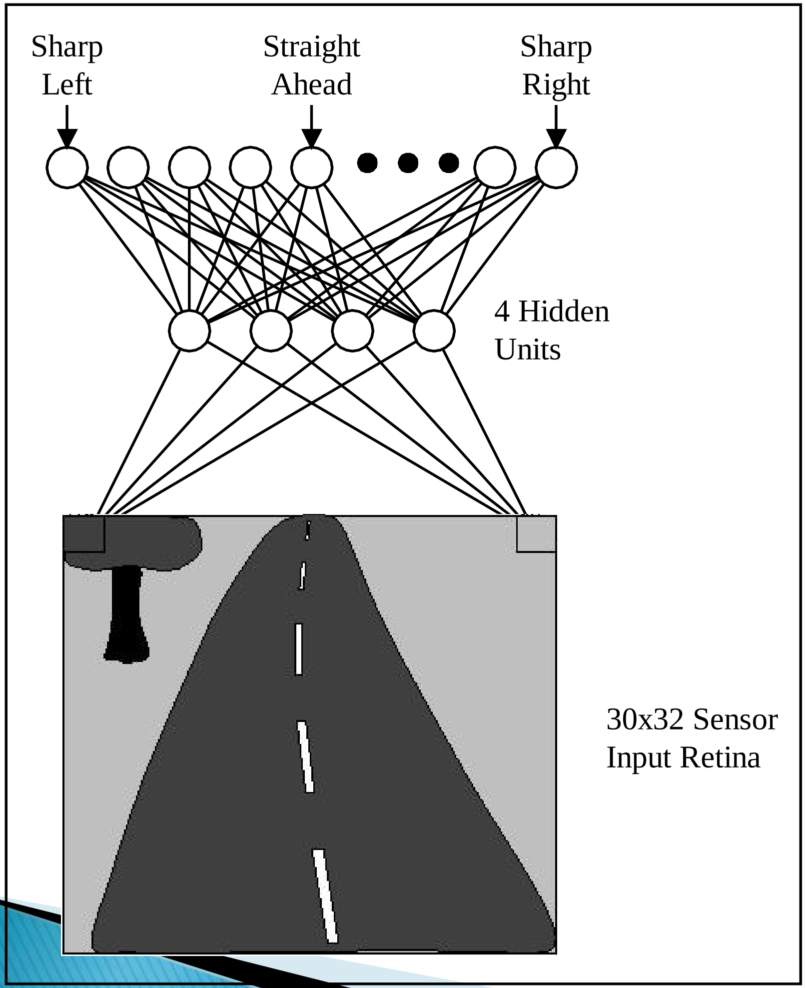
  4. Update each network weight $w_{i,j}$

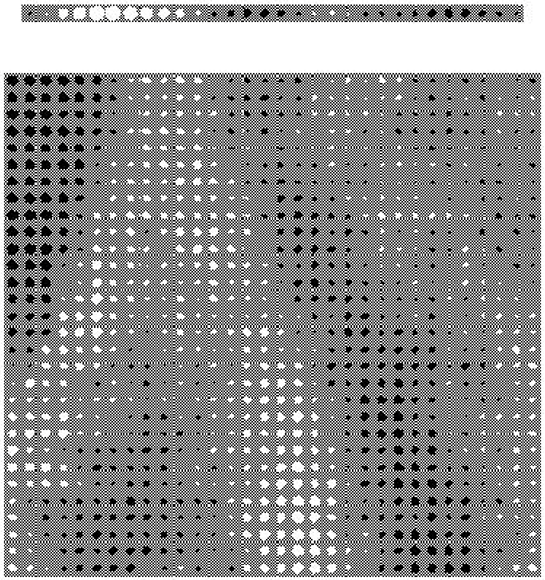
     $$w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i}$$

     where

     $$\Delta w_{j,i} = \eta\, \delta_j x_{j,i}$$

ALVINN drives 70 mph on highways





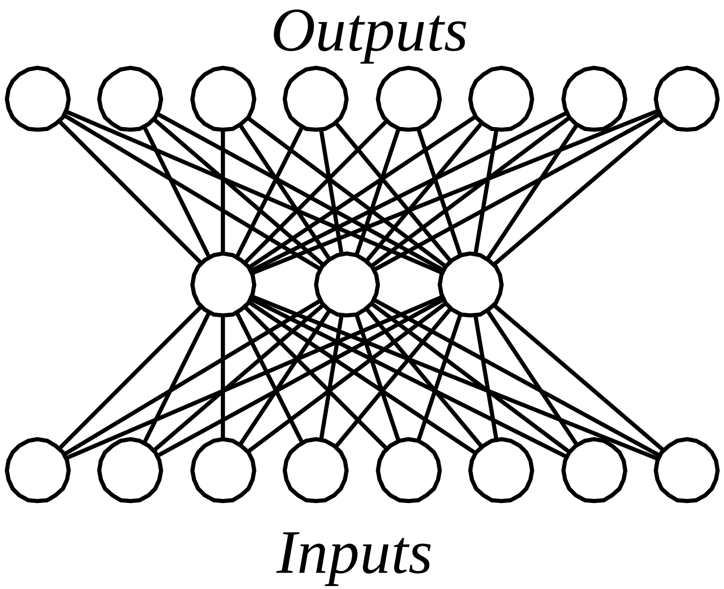
More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{j,i}(n) = \eta\, \delta_j x_{j,i} + \alpha\, \Delta w_{j,i}(n-1)$$

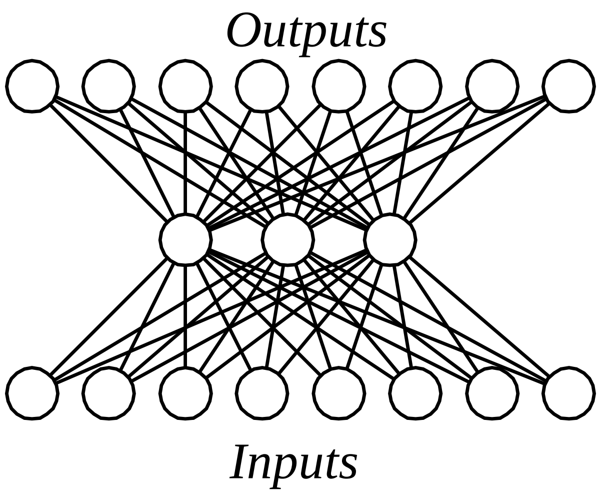
- Minimizes error over training examples
- Will it generalize well to subsequent examples?
- Training can take thousands of iterations -- slow!
  - Using network after training is fast

Learning Hidden Layer Representations



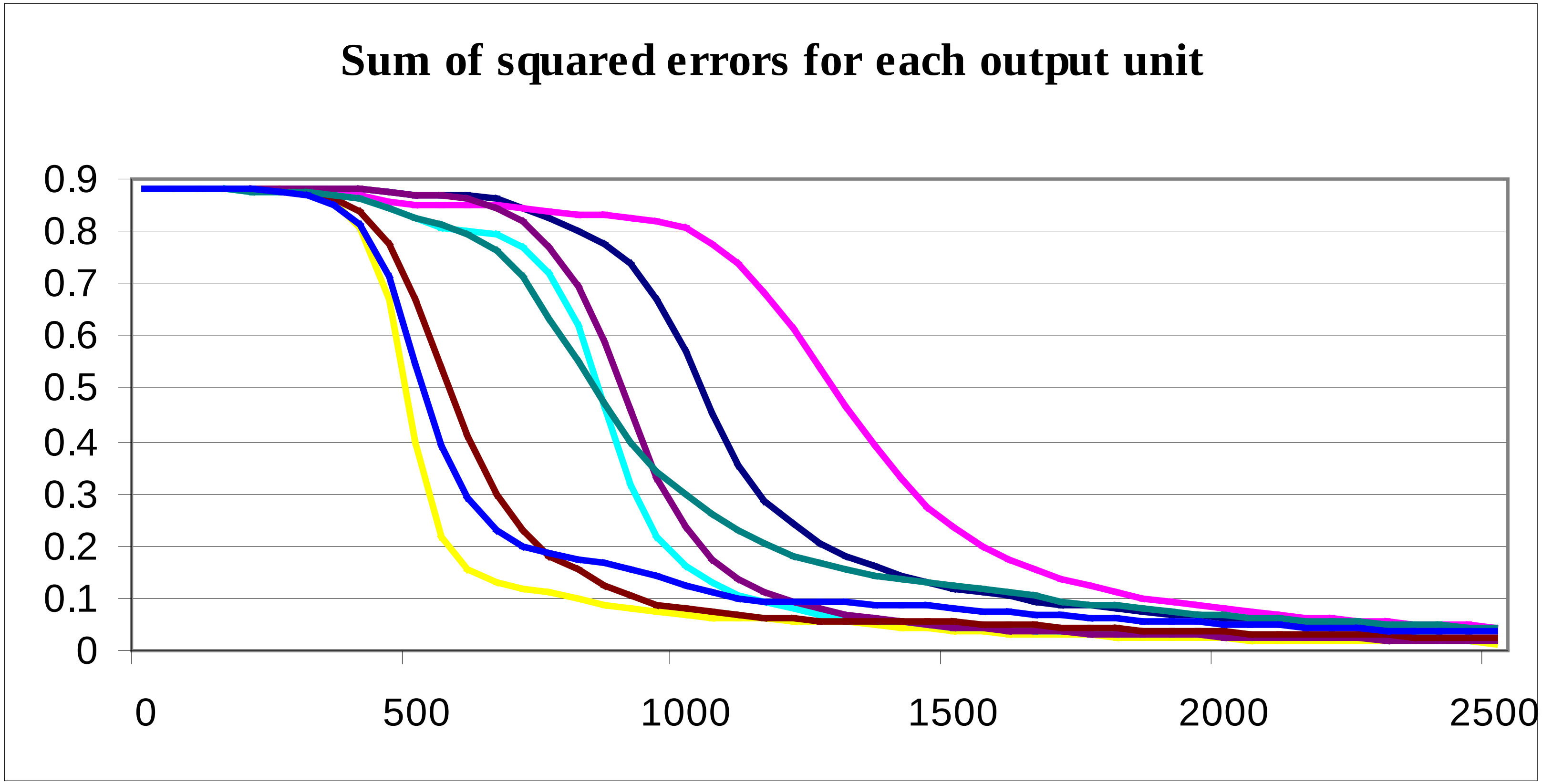
Input	Output
10000000 →	10000000
01000000 →	01000000
00100000 →	00100000
00010000 →	00010000
00001000 →	00001000
00000100 →	00000100
00000010 →	00000010
00000001 →	00000001

Learning Hidden Layer Representations

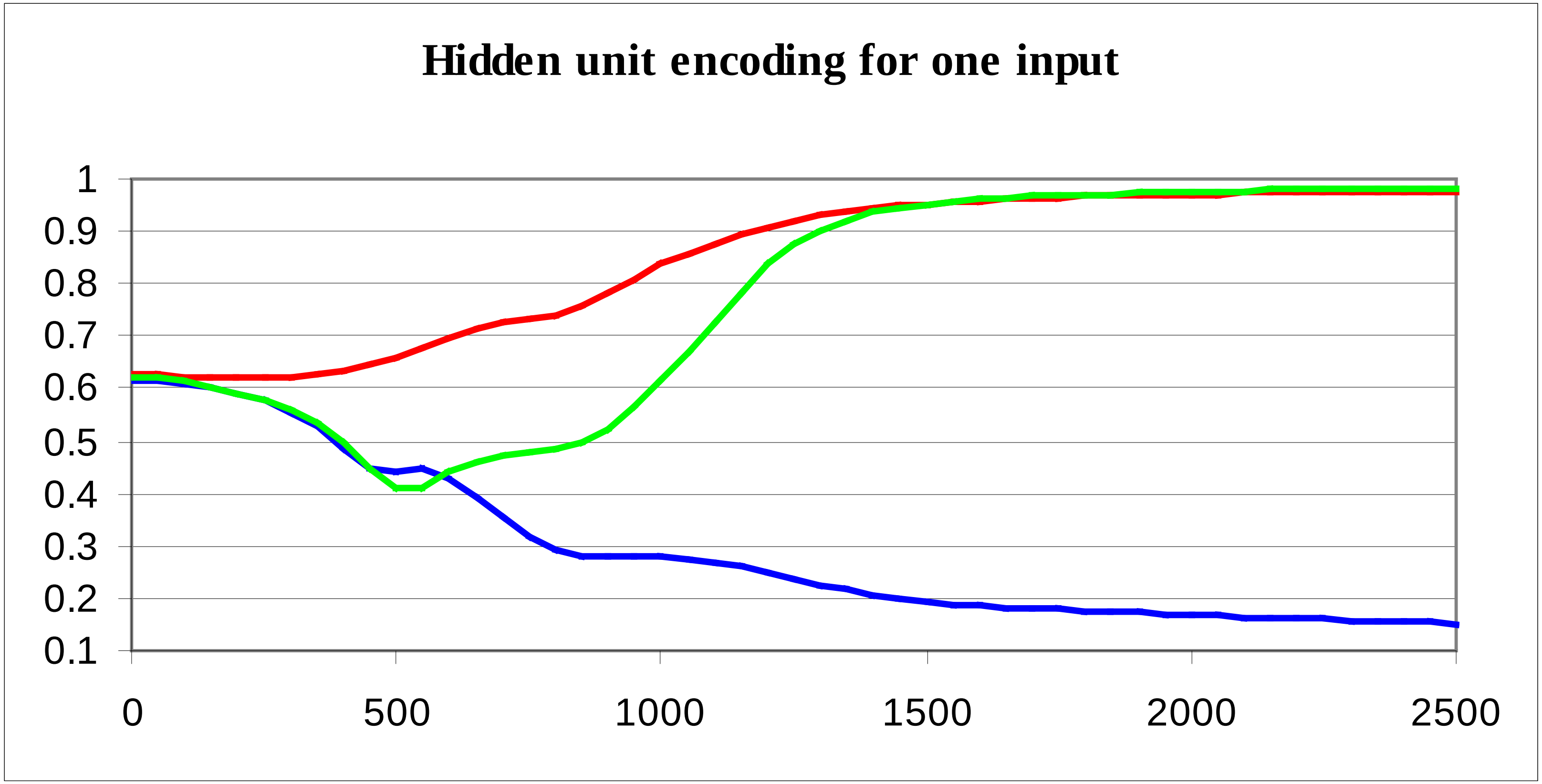


Input			Output
10000000	→	.89 .04 .08	→ 10000000
01000000	→	.01 .11 .88	→ 01000000
00100000	→	.01 .97 .27	→ 00100000
00010000	→	.99 .97 .71	→ 00010000
00001000	→	.03 .05 .02	→ 00001000
00000100	→	.22 .99 .99	→ 00000100
00000010	→	.80 .01 .98	→ 00000010
00000001	→	.60 .94 .01	→ 00000001

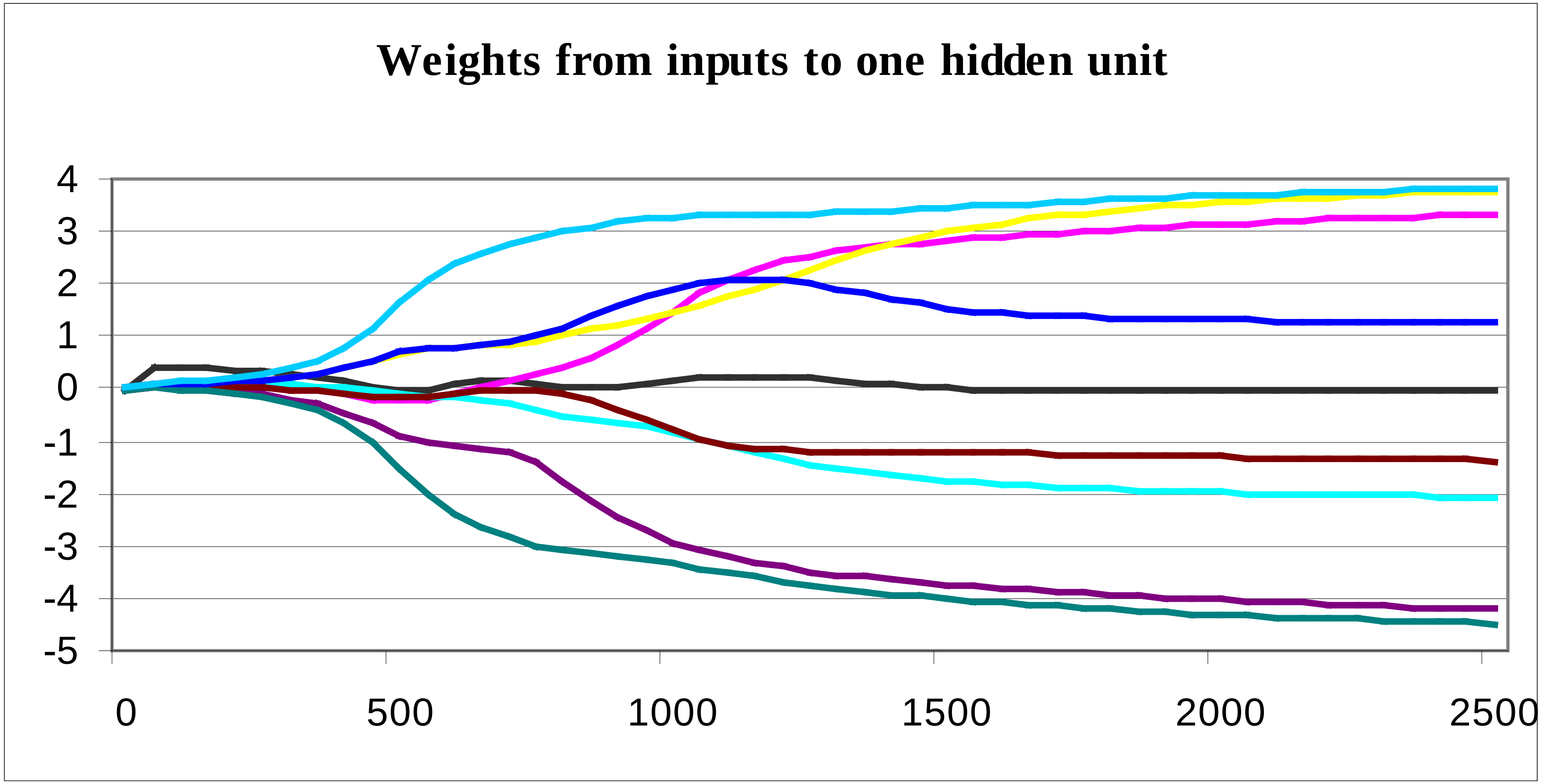
Output Unit Error during Training



Hidden Unit Encoding



Input to Hidden Weights



Convergence of Backpropagation

Gradient descent to some local minimum

- Perhaps not global minimum
- Momentum can cause quicker convergence
- Stochastic gradient descent also results in faster convergence
- Can train multiple networks and get different results (using different initial weights)

Nature of convergence

- Initialize weights near zero
- Therefore, initial networks near-linear
- Increasingly non-linear functions as training progresses

Expressive Capabilities of ANNs

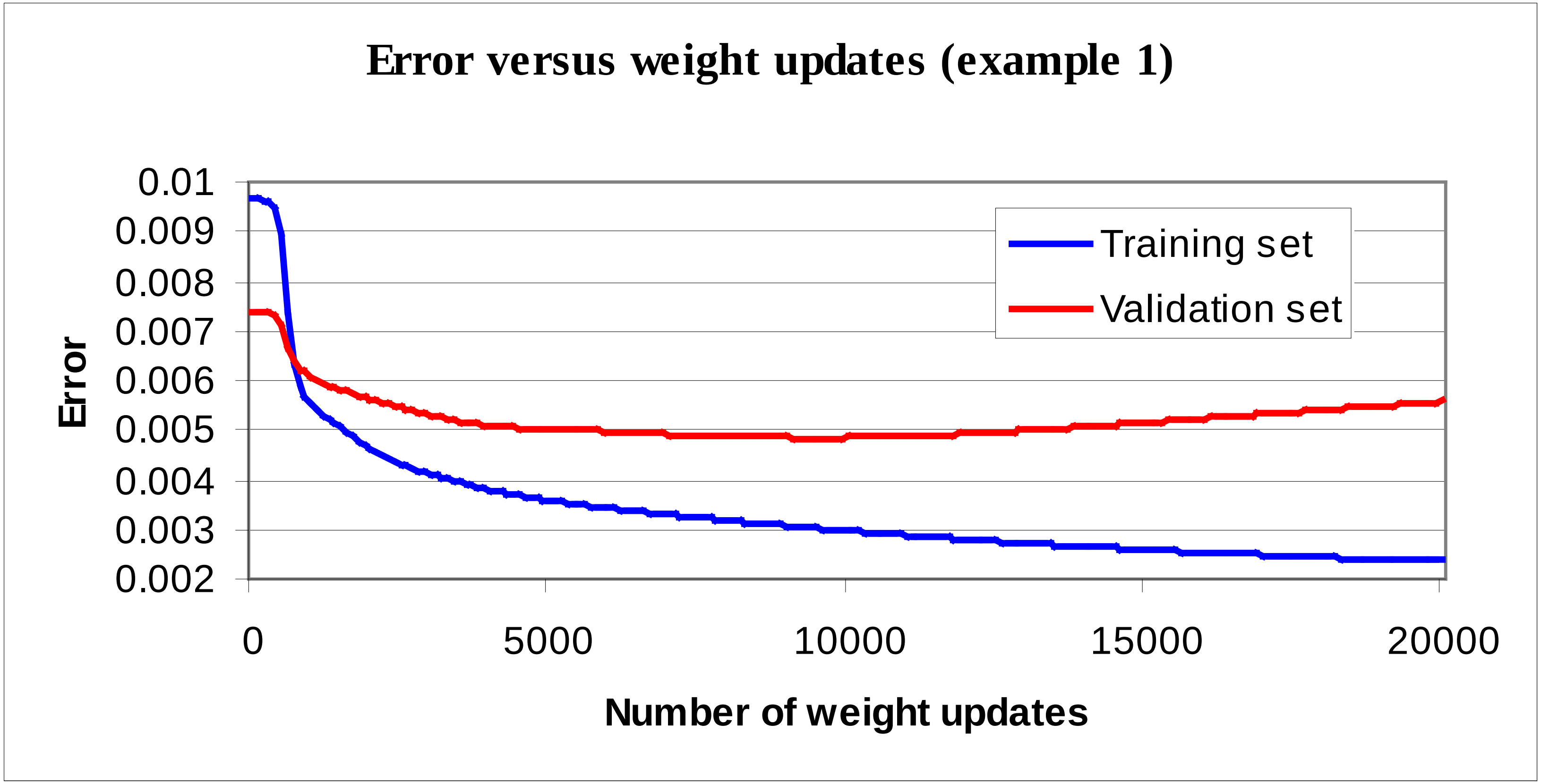
Boolean functions:

- Every Boolean function can be represented by network with a single hidden layer
- But that might require an exponential (in the number of inputs) hidden units

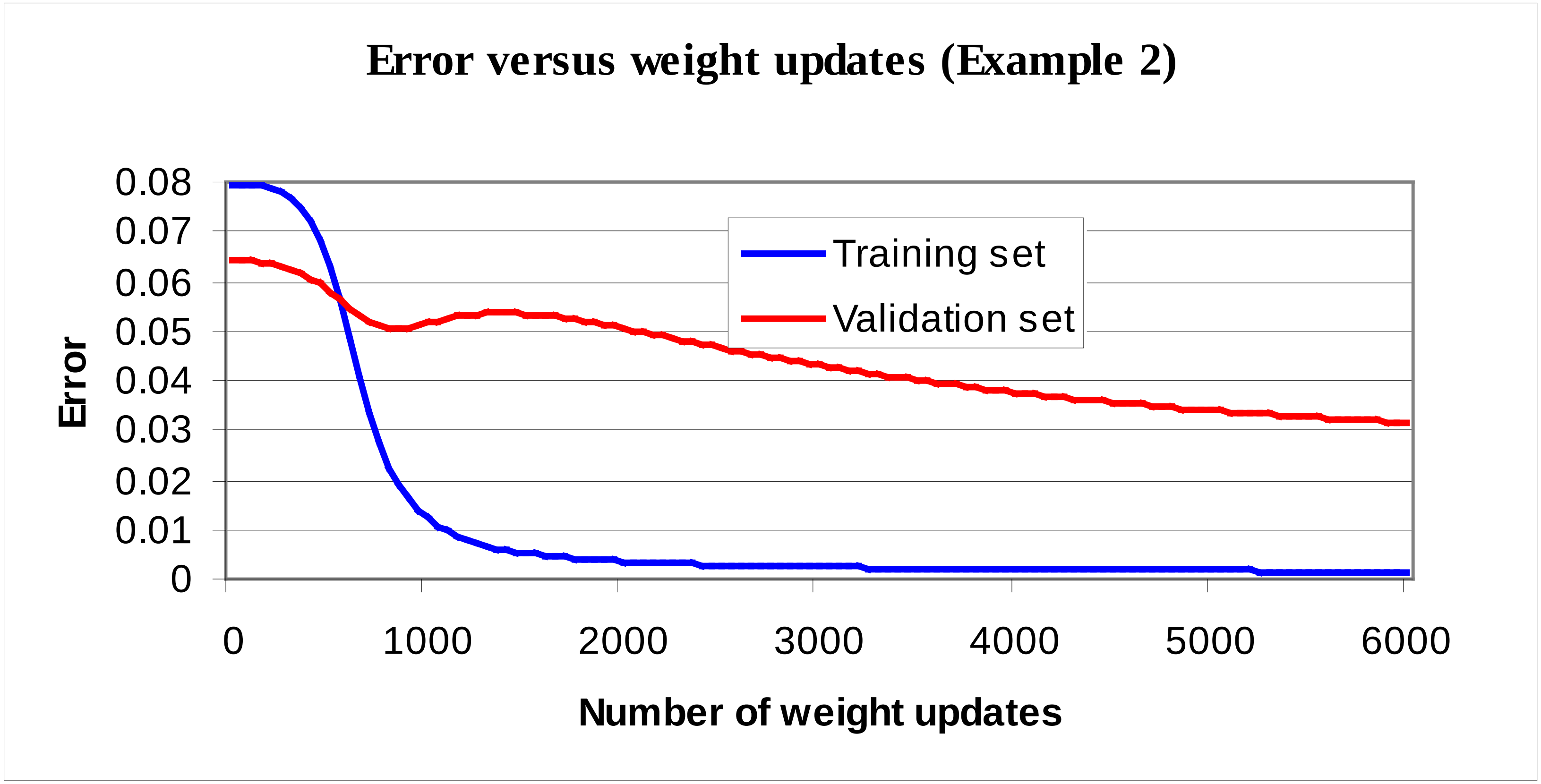
Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error by a network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988]

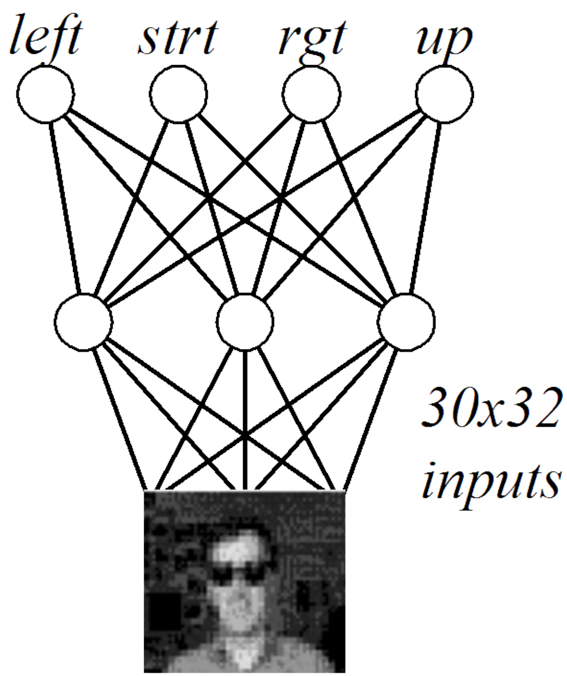
Overfitting in ANNs



Overfitting in ANNs



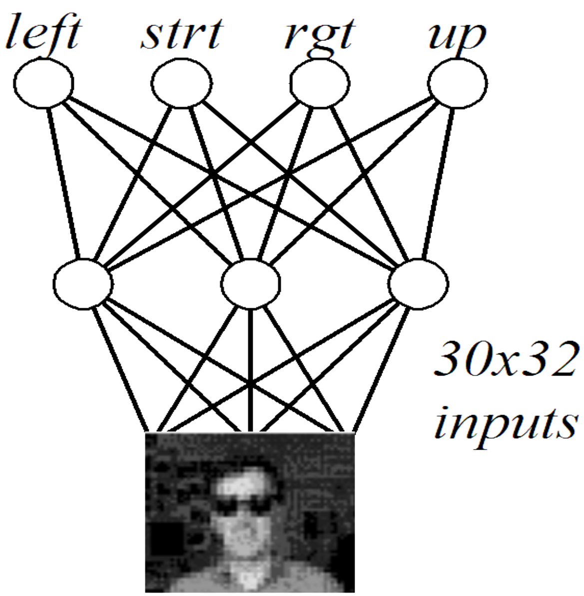
Neural Nets for Face Recognition



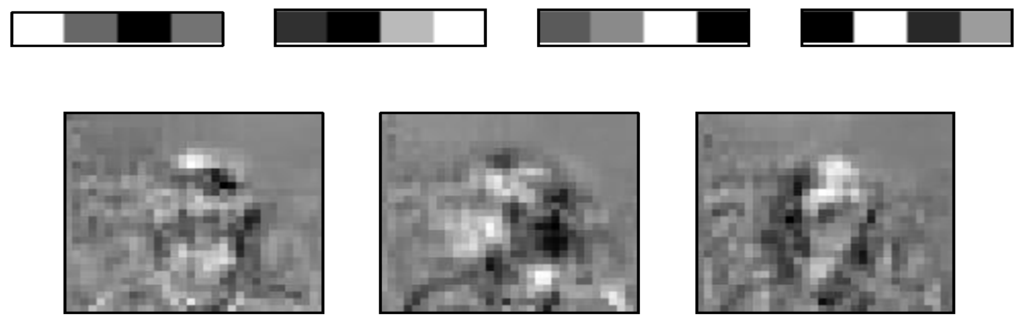
90% accurate learning head pose, and recognizing 1-of-20 faces

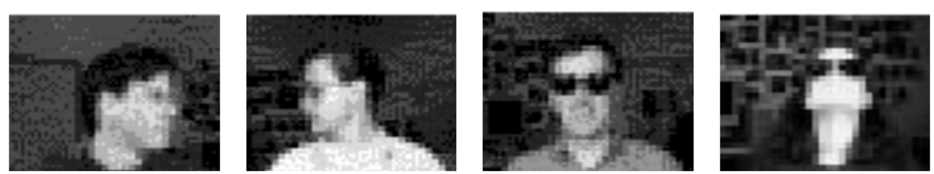
Typical Input Images

Learned Network Weights



Learned Weights





Typical Input Images

Alternative Error Functions

Penalize large weights:

$$E(\vec{w}) \equiv \tfrac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2+\gamma\sum_{i,j}w^2{}_{ji}$$

Train on target slopes as well as values:

$$E(\vec{w}) \equiv \tfrac{1}{2}\sum_{d\in D}\sum_{k\in outputs}\left[(t_{kd}-o_{kd})^2+\mu\left(\frac{\partial t_{kd}}{\partial x^j{}_d}-\frac{\partial o_{kd}}{\partial x^j{}_d}\right)^2\right]$$

Tie together weights:

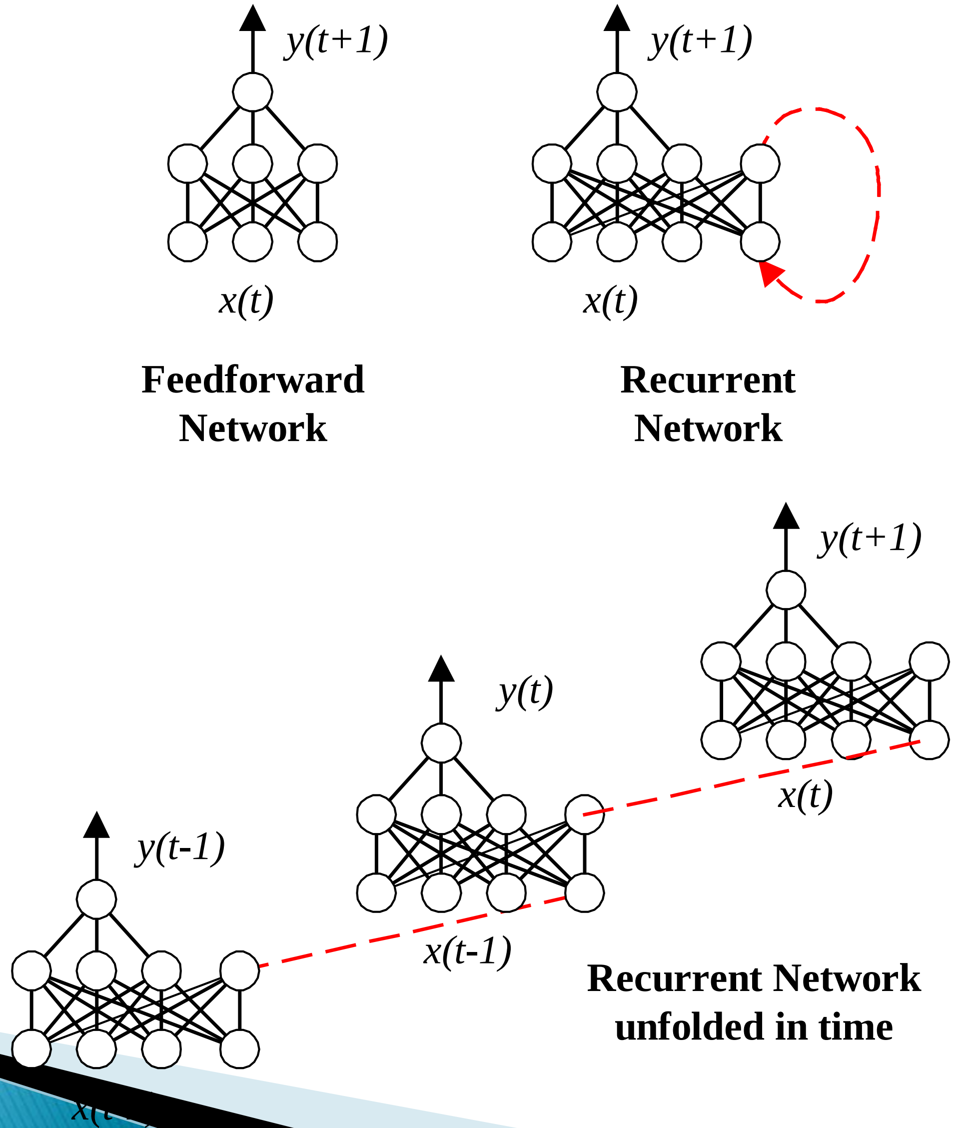
- e.g., in phoneme recognition

NNs for Handwritten Character Recognition

- LeNet, Yann LeCun et al.
- Convolutional Neural Networks
  - Uses backprop
  - Representation optimized for pixel processing; handling extreme variability, and robust to distortions and simple transformations
- http://yann.lecun.com/exdb/lenet/

- Lenet Video

Recurrent Networks



What you should know

- ANNs are practical method for learning real-valued and vector-valued functions over continuous and discrete inputs
- Backpropagation can be used to find weights for multi-layer ANNs
- Overfitting is an issue for ANNs
- Many, many variants that we have not covered