
Chapter 4 (Part 2): Artificial Neural Networks

CS 536: Machine Learning
Littman (Wu, TA)

Grading Components

- HWs (not handed in): 0%
- Project paper: 25%
 - document (15%), review (5%), revision(5%)
- Project presentation: 20%
 - oral (10%), slides (10%)
- Midterm (take home): 20%
- Final: 35%

Administration

Assignment...

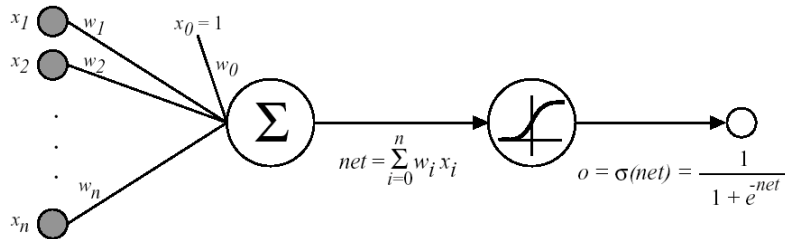
Artificial Neural Networks

[Read Ch. 4]

[Review exercises 4.1, 4.2, 4.5, 4.9, 4.11]

- Threshold units [last time]
- Gradient descent [last time]
- Multilayer networks
- Backpropagation
- Hidden layer representations
- Example: Face Recognition
- Advanced topics

Sigmoid Unit



$\sigma(x)$ is the sigmoid (s-like) function
 $1/(1 + e^{-x})$

Derivatives of Sigmoids

Nice property:

$$d \sigma(x) / dx = \sigma(x) (1 - \sigma(x))$$

We can derive gradient decent rules to train

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

Error Gradient for Sigmoid

$$\begin{aligned} \partial E / \partial w_i &= \partial / \partial w_i \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \partial / \partial w_i (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2 (t_d - o_d) \partial / \partial w_i (t_d - o_d) \\ &= \sum_d (t_d - o_d) (-\partial o_d / \partial w_i) \\ &= - \sum_d (t_d - o_d) (\partial o_d / \partial net_d \partial net_d / \partial w_i) \end{aligned}$$

Even more...

But we know:

$$\begin{aligned} \partial o_d / \partial net_d &= \partial \sigma(net_d) / \partial net_d = o_d (1 - o_d) \\ \partial net_d / \partial w_i &= \partial (\mathbf{w} \cdot \mathbf{x}_d) / \partial w_i = x_{i,d} \end{aligned}$$

So:

$$\partial E / \partial w_i = - \sum_d (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

Backpropagation Algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do
 1. Input the training example to the network and compute the network outputs
 2. For each output unit k
$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$
 3. For each hidden unit h
$$\delta_h = o_h(1 - o_h) \sum_{k \text{ in outputs}} w_{h,k} \delta_k$$
 4. Update each network weight $w_{i,j}$
$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \text{ where } \Delta w_{i,j} = \eta \delta_j x_{i,j}$$

More more

- Often include weight *momentum* α
$$\Delta w_{i,j}(\mathbf{n}) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(\mathbf{n}-1)$$
- Minimizes error over training examples
 - Will it generalize well to subsequent examples?
- Training can take thousands of iterations
→ slow!
- Using network after training is very fast

More on Backpropagation

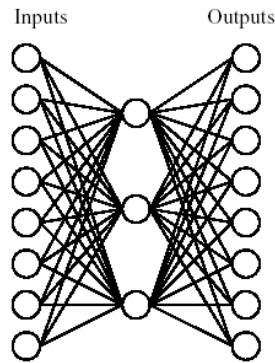
- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
 - In practice, often works well (can run multiple times)

Hidden Layer Reps

Simple target function:

- | Input | Output |
|------------|------------|
| • 10000000 | → 10000000 |
| • 01000000 | → 01000000 |
| • 00100000 | → 00100000 |
| • 00010000 | → 00010000 |
| • 00001000 | → 00001000 |
| • 00000100 | → 00000100 |
| • 00000010 | → 00000010 |
| • 00000001 | → 00000001 |

Autoencoder

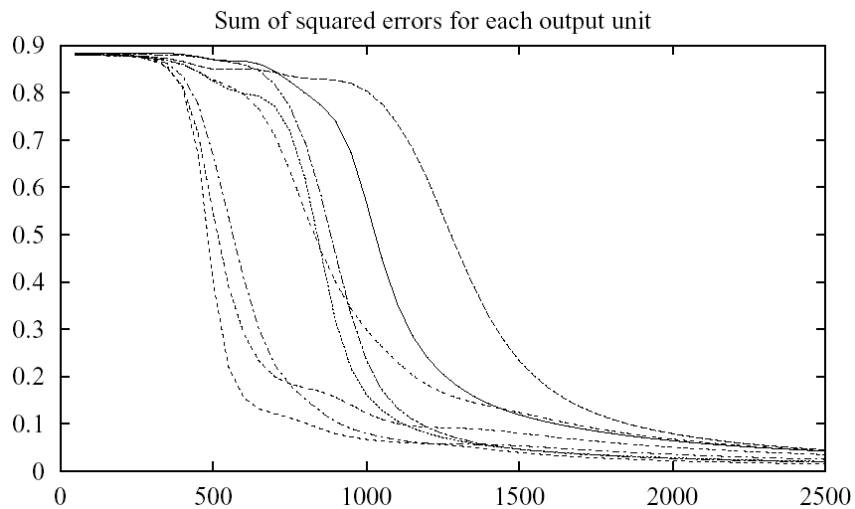


Can the mapping be learned with this network??

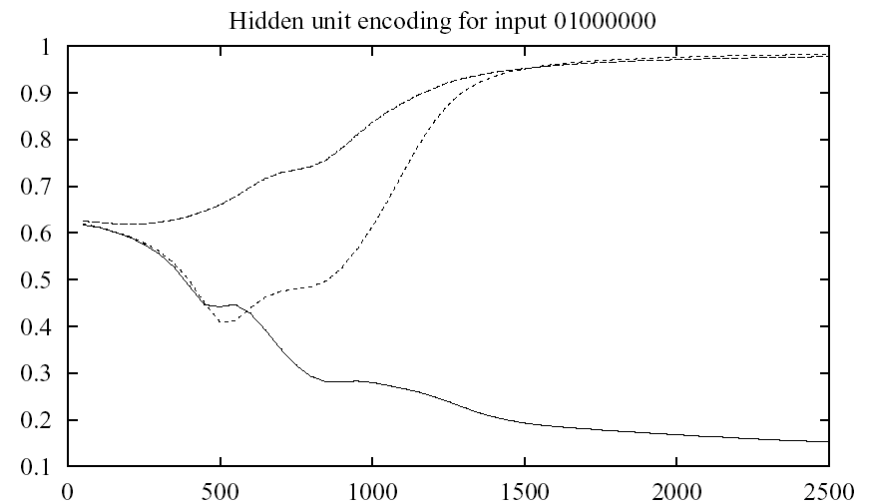
Hidden Layer Rep.

- | Input | Hidden Values | 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 |

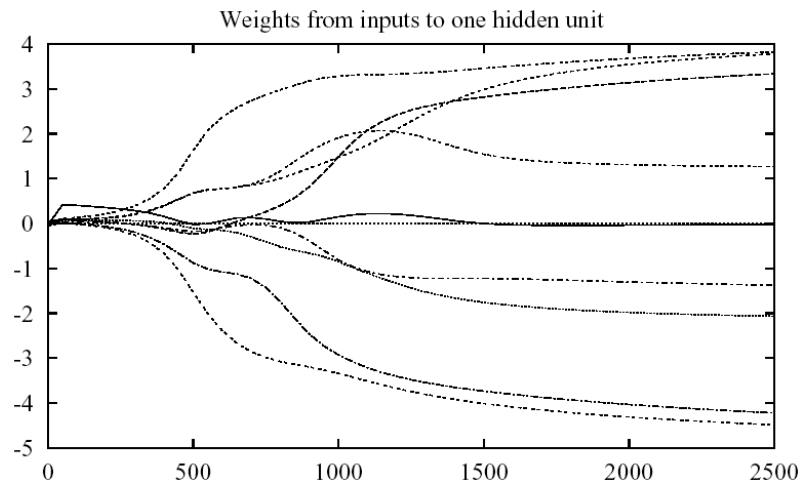
Training



Training



Training



More on Convergence

Nature of convergence

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

Convergence of Backprop

Gradient descent to some local minimum

- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train multiple nets with different initial weights

Expressiveness of ANNs

Boolean functions:

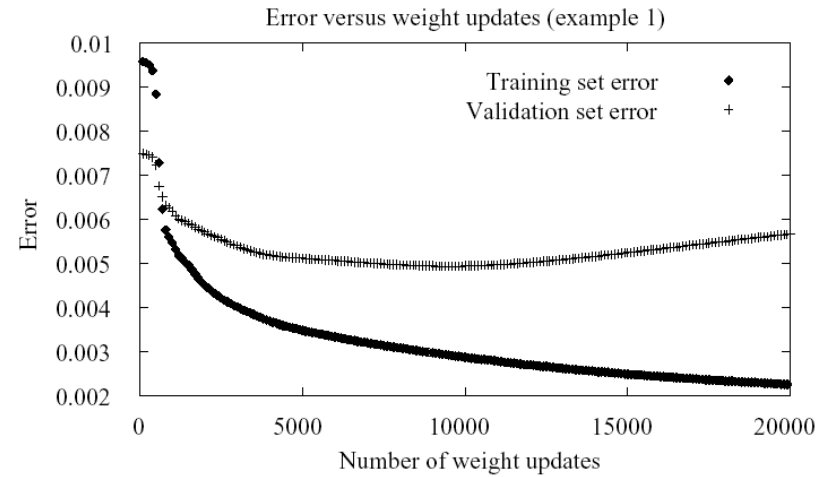
- Every Boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Real-valued Functions

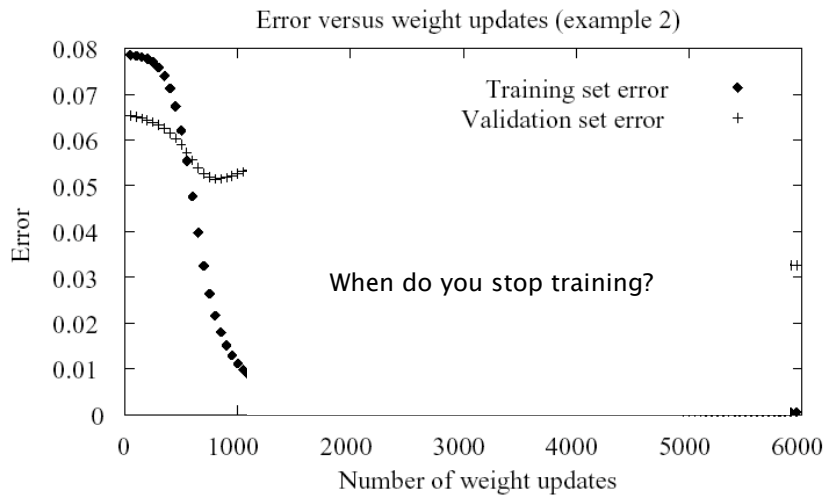
Continuous functions:

- Every bounded continuous function can be approximated, with arbitrarily small error, by 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].

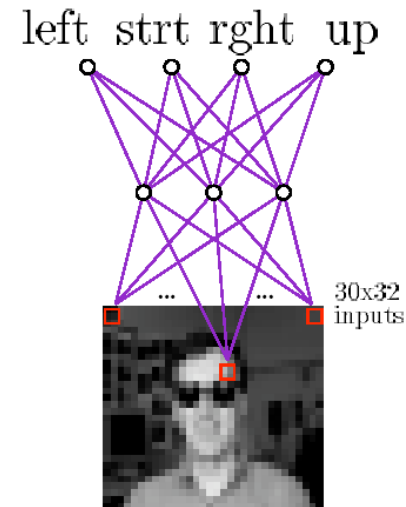
Overfitting in ANNs (Ex. 1)



Overfitting in ANNs (Ex. 2)



Face Recognition

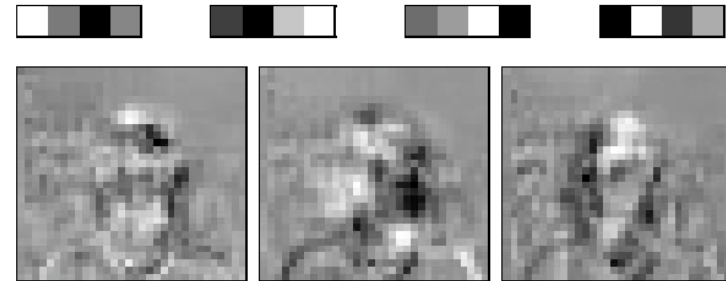


Typical input images



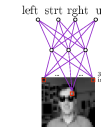
- 90% accurate learning head pose, and recognizing 1-of-20 faces

Learned Weights



<http://www.cs.cmu.edu/tom/faces.html>

Bias first?



Alternative Error Functions

Penalize large weights:

$$E(\mathbf{w}) \equiv 1/2 \sum_{d \text{ in } D} \sum_{k \text{ in outputs}} (t_{kd} - o_{kd})^2 + \gamma \sum_{ij} w_{ji}^2$$

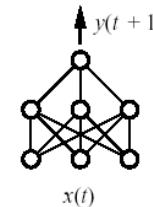
Train on target slopes as well as values:

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

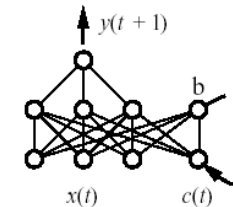
Tie together weights:

- e.g., in phoneme recognition network

Recurrent Networks



(a) Feedforward network



(b) Recurrent network

Unfolding: BPTT

