# Neural Networks

Pradeep Ravikumar

Co-instructor: Ziv Bar Joseph

Machine Learning 10-701

Slides Courtesy: Previous Instructors

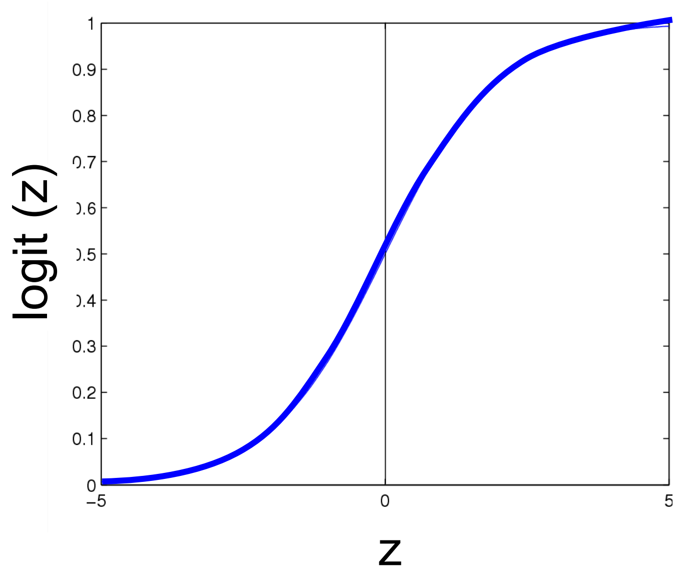
## Logistic Regression

Assumes the following functional form for P(Y|X):

$$P(Y=1|X) = \frac{1}{1+\exp(-(w_0+\sum_i w_i X_i))}$$

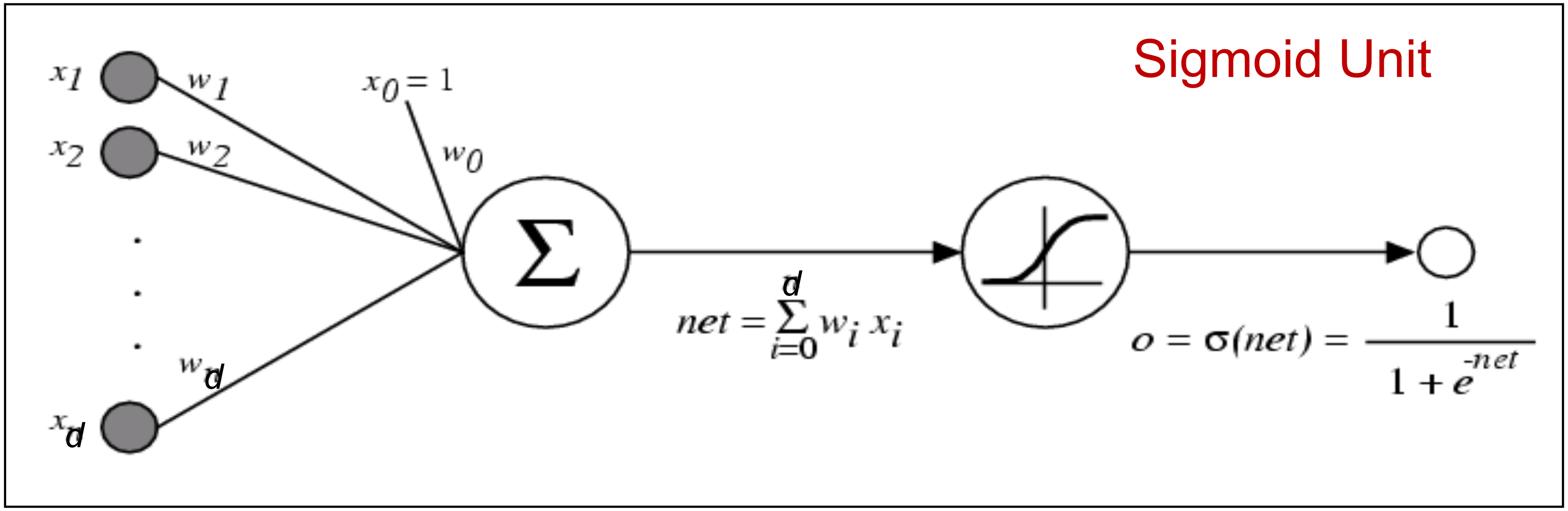
Logistic function applied to a linear function of the data

**Logistic function (or Sigmoid):** $\frac{1}{1+exp(-z)}$



# Logistic function as a Graph

$$\text{Output, } o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$

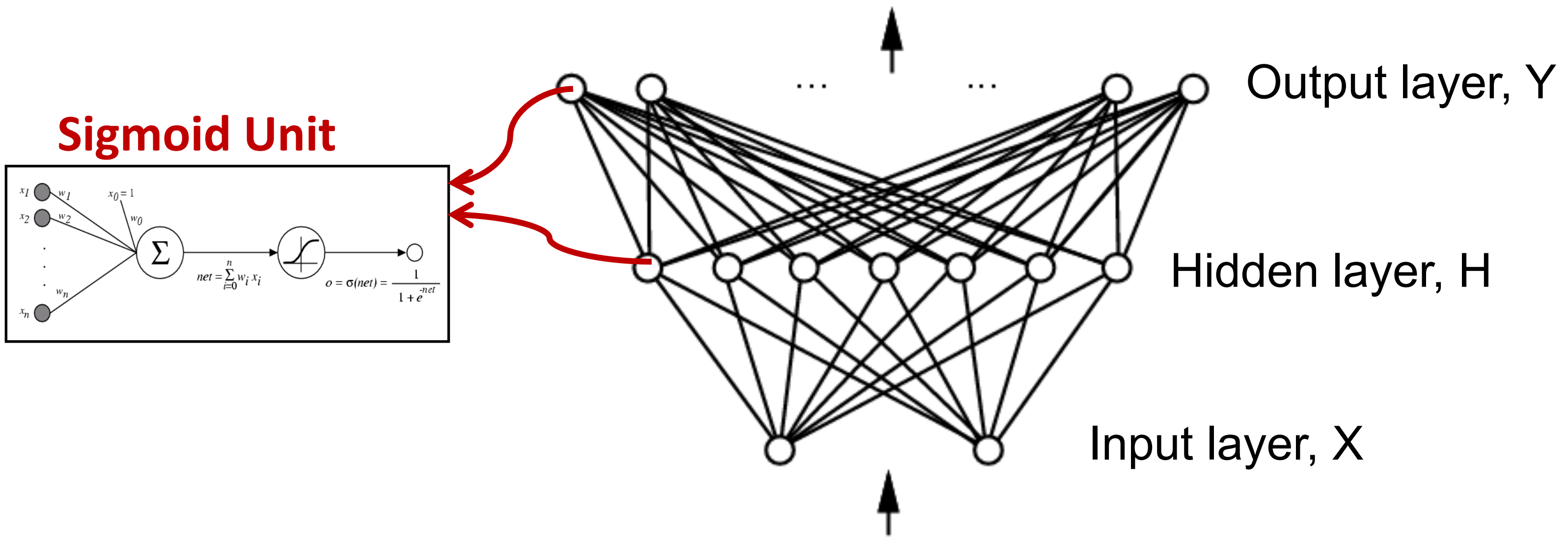


Sigmoid Unit
$x_1$
$x_2$
$x_d$
$w_1$
$w_2$
$w_d$
$x_0 = 1$
$w_0$
$\Sigma$
$net = \sum_{i=0}^{d} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

# Neural Networks to learn f: X → Y

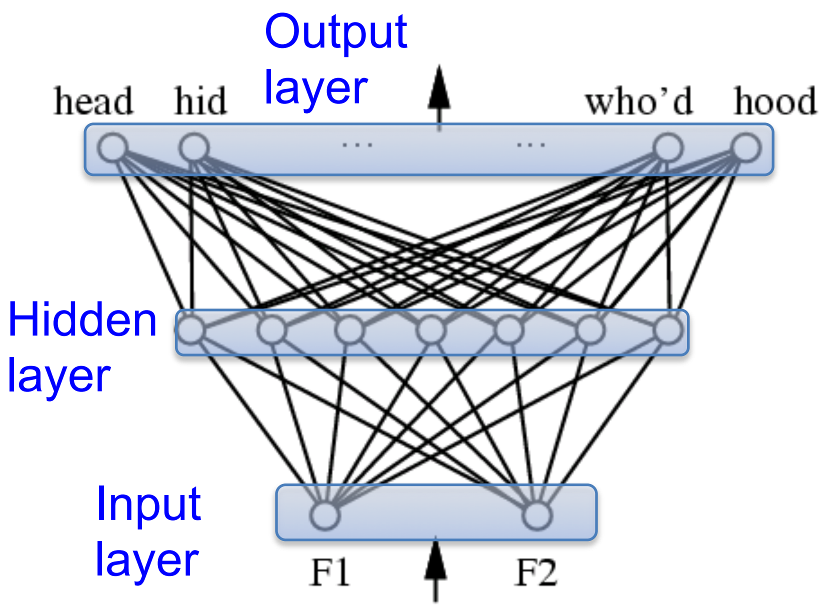
- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

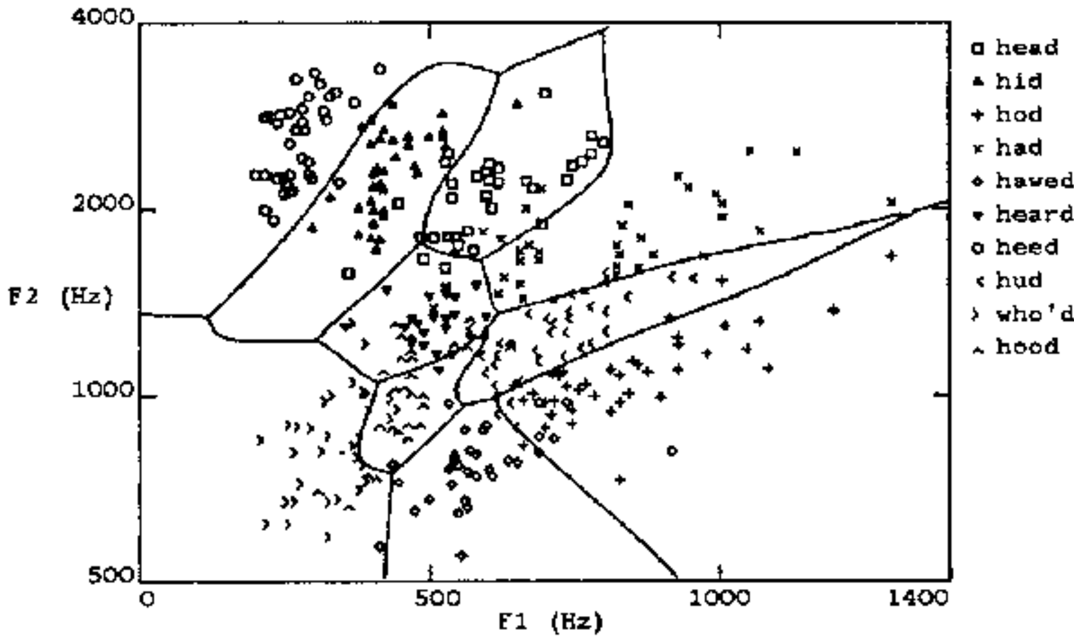
- Neural networks - Represent f by *network* of logistic/sigmoid units:



# Multilayer Networks of Sigmoid Units

Neural Network trained to distinguish vowel sounds using 2 formants (features)

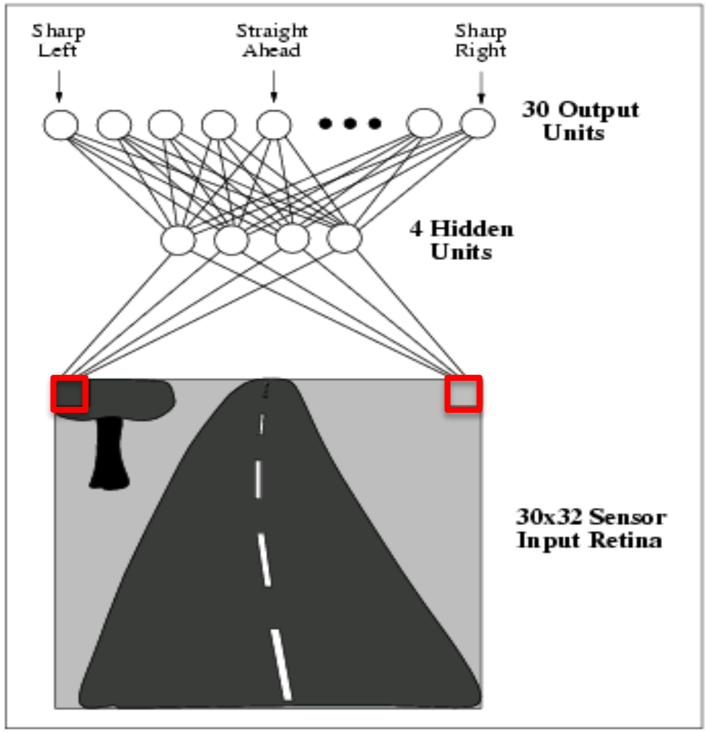




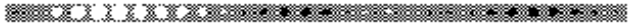
Two layers of logistic units

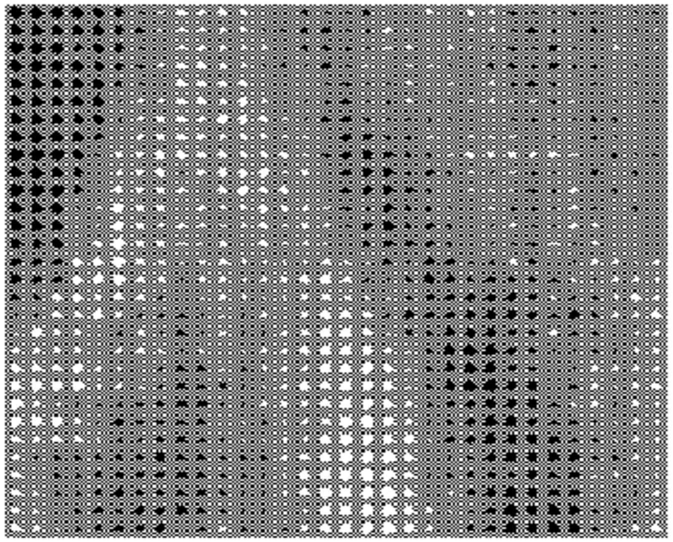
Highly non-linear decision surface

Neural Network trained to drive a car!



Weights to output units from one hidden unit





Weights of each pixel for one hidden unit

## 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 steps doesn't seem like enough

$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process

# Prediction using Neural Networks

**Prediction** – Given neural network (hidden units and weights), use it to predict the label of a test point

**Forward Propagation** –
Start from input layer
For each subsequent layer, compute output of sigmoid unit

Sigmoid unit:

$$o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i x_i)$$

1-Hidden layer,
1 output NN:

$$o(\mathbf{x}) = \underbrace{\sigma\left(w_0 + \sum_h w_h \underbrace{\sigma(w_0^h + \sum_i w_i^h x_i)}_{o_h}\right)}$$

# M(C)LE Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

$y = f(x) + \varepsilon \longleftarrow$ assume noise $N(0,\sigma_\varepsilon)$, iid

deterministic

- Let's maximize the conditional data likelihood

$$W \leftarrow \arg\max_W \ \ln \prod_l P(Y^l|X^l, W)$$

$$W \leftarrow \arg\min_W \ \sum_l (y^l - \hat{f}_W(x^l))^2$$

Learned neural network

Train weights of all units to minimize sum of squared errors of predicted network outputs

# MAP Training for Neural Networks

- Consider regression problem f:X→Y , for scalar Y

$y = f(x) + \varepsilon$ ← noise $N(0,\sigma_\varepsilon)$

deterministic

Gaussian $P(W) = N(0,\sigma I)$

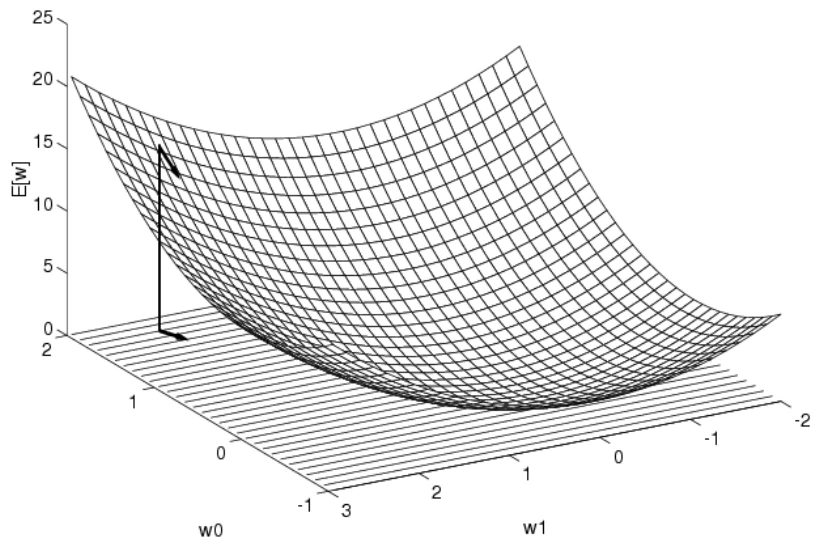
$$W \leftarrow \arg\max_W \ \ln \ P(W) \prod_l P(Y^l|X^l, W)$$

$$W \leftarrow \arg\min_W \left[ c \sum_i w_i^2 \right] + \left[ \sum_l (y^l - \hat{f}_W(x^l))^2 \right]$$

$\ln P(W) \leftrightarrow c \sum_i w_i^2$

Train weights of all units to minimize sum of squared errors of predicted network outputs plus weight magnitudes

## Gradient Descent



*E* – Mean Square Error

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_d}\right]$$

Training rule:

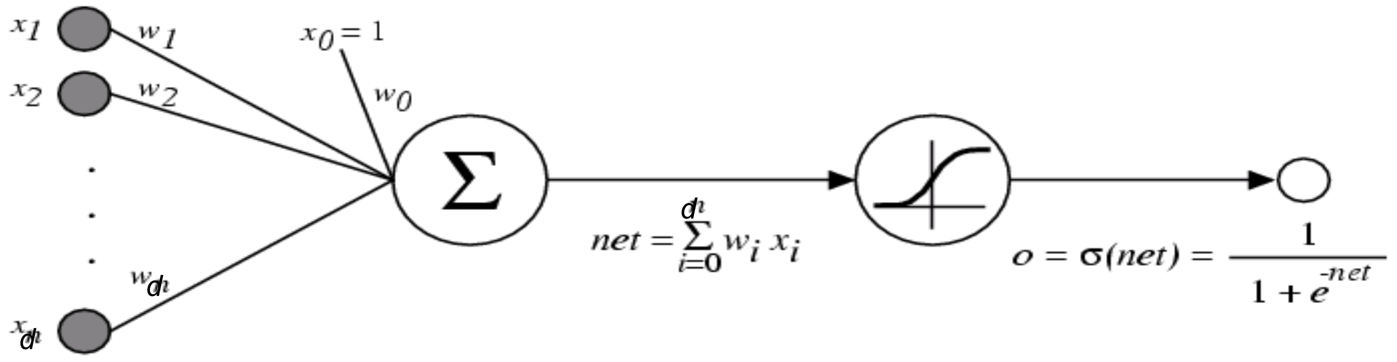
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

**For Neural Networks,**
***E*[*w*] no longer convex in w**

## Training Neural Networks



$x_1$
$x_2$
$x_d$
$w_1$
$w_2$
$w_d$
$x_0 = 1$
$w_0$
$\Sigma$
$net = \sum_{i=0}^{d} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

$\sigma(x)$ is the sigmoid function

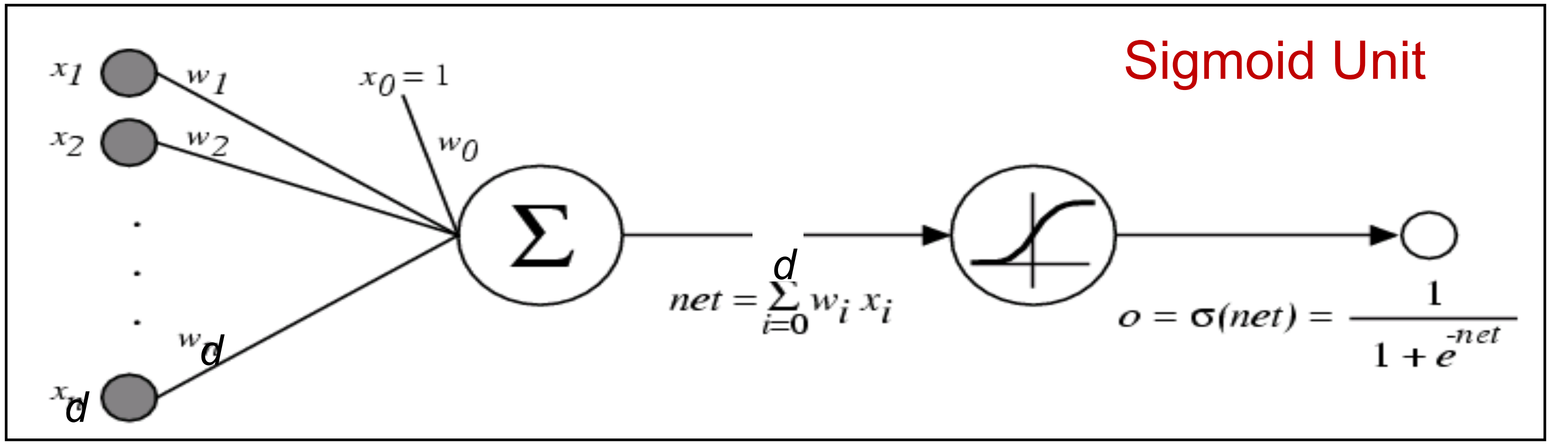
$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$ Differentiable

We can derive gradient decent rules to train

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

# Error Gradient for a Sigmoid Unit


Sigmoid Unit
$x_1$
$x_2$
$x_d$
$w_1$
$w_2$
$w_d$
$x_0 = 1$
$w_0$
$net = \sum_{i=0}^{d} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^{l} - o^{l})^2 \\
&= \frac{1}{2} \sum_{l} \frac{\partial}{\partial w_i} (y^{l} - o^{l})^2 \\
&= \frac{1}{2} \sum_{l} 2 (y^{l} - o^{l}) \frac{\partial}{\partial w_i} (y^{l} - o^{l}) \\
&= \sum_{l} (y^{l} - o^{l}) \left( - \frac{\partial o^{l}}{\partial w_i} \right) \\
&= - \sum_{l} (y^{l} - o^{l}) \frac{\partial o^{l}}{\partial net^{l}} \frac{\partial net^{l}}{\partial w_i}
\end{aligned}$$

But we know:

$$\frac{\partial o^{l}}{\partial net^{l}} = \frac{\partial \sigma(net^{l})}{\partial net^{l}} = o^{l}(1 - o^{l})$$

$$\frac{\partial net^{l}}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}^{l})}{\partial w_i} = x_i^{l}$$

So:

$$\frac{\partial E}{\partial w_i} = - \sum_{l \in D} (y^{l} - o^{l}) o^{l} (1 - o^{l}) x_i^{l}$$

## Incremental (Stochastic) Gradient Descent

**Batch mode** Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Using all training data *D*

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2$$

**Incremental mode** Gradient Descent:
Do until satisfied

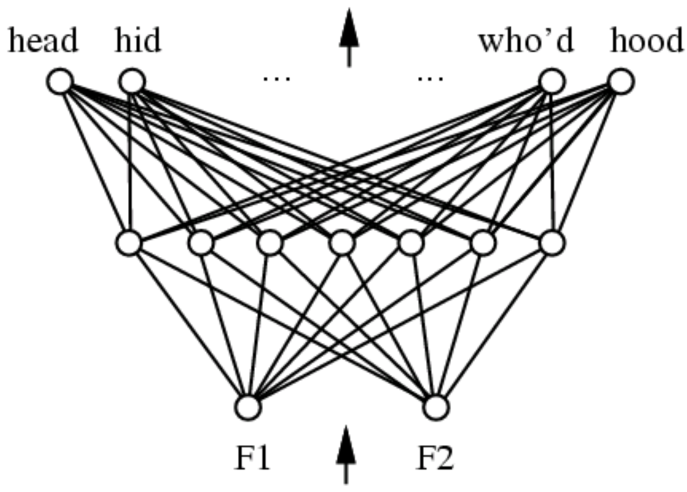
- For each training example $l$ in $D$
  1. Compute the gradient $\nabla E_l[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_l[\vec{w}]$

$$E_l[\vec{w}] \equiv \frac{1}{2} (y^l - o^l)^2$$

also known as Stochastic Gradient Descent (SGD)

*Incremental Gradient Descent* can approximate *Batch Gradient Descent* arbitrarily closely if $\eta$ made small enough

## Backpropagation Algorithm (MLE)



```
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
             $\delta_k^l \leftarrow o_k^l(1 - o_k^l)(y_k^l - o_k^l)$
      3. For each hidden unit h
             $\delta_h^l \leftarrow o_h^l(1 - o_h^l) \sum_{k \in outputs} w_{h,k} \delta_k^l$
      4. Update each network weight $w_{i,j}$
             $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}^l$
         where
             $\Delta w_{i,j}^l = \eta \delta_j^l o_i^l$
```

$\longrightarrow$ Using Forward propagation

l = training example

$y_k$ = target output (label) of output unit k

$o_{k(h)}$ = unit output (obtained by forward propagation)

$w_{ij}$ = wt from i to j

Note: if i is input variable, $o_i = x_i$

## 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* $\alpha$

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

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

Objective/Error no longer convex in weights

## Expressive Capabilities 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

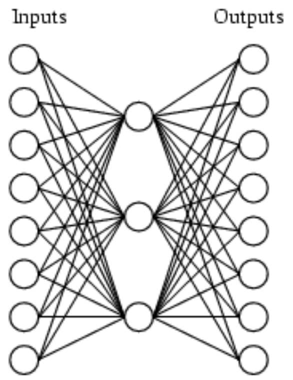
Continuous functions:

- 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].

Limited by amount of labeled data.
What about unsupervised problems?

## Auto-Encoders
## Deep Generative Models

## Learning Hidden Layer Representations

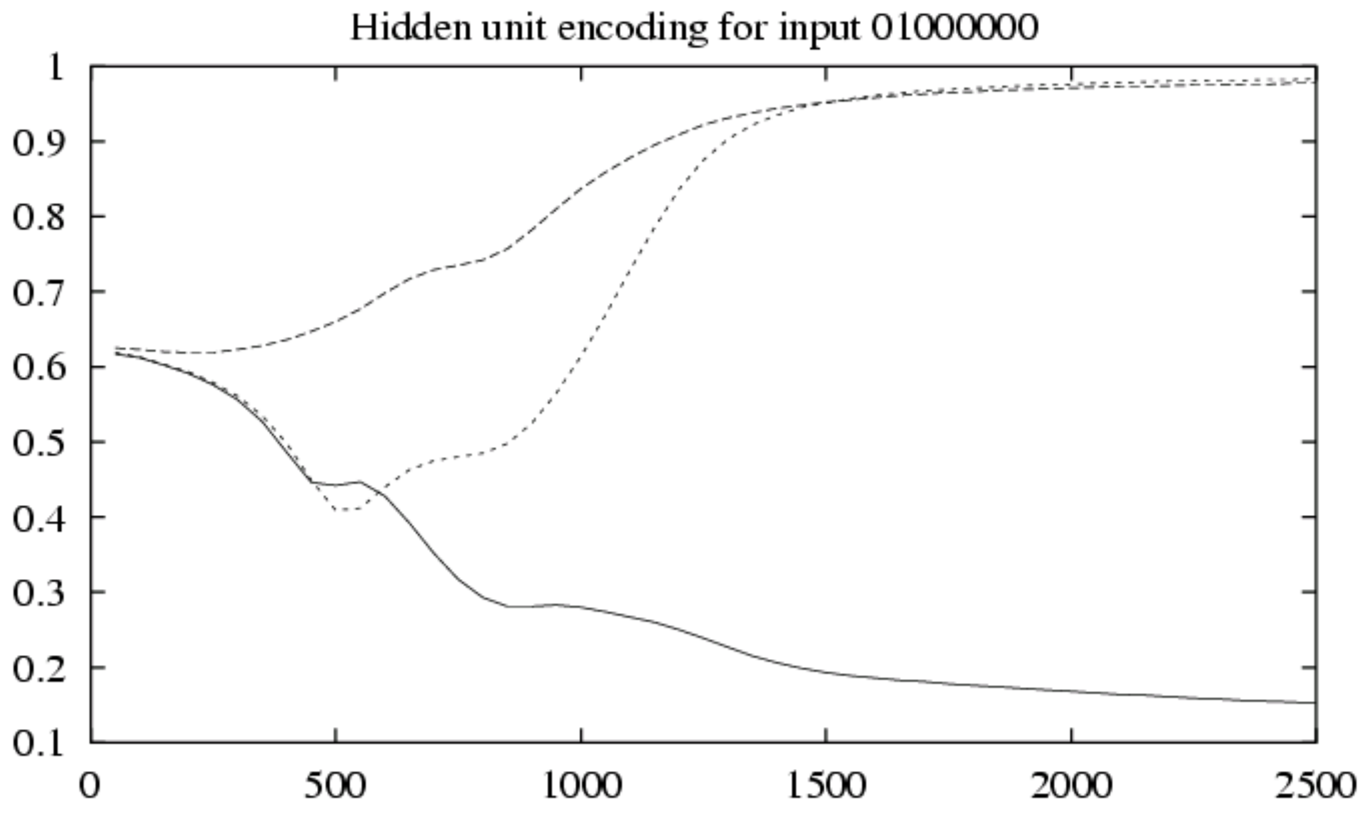


A target function:

| Input | | Output |
|---|---|---|
| 10000000 | $\rightarrow$ | 10000000 |
| 01000000 | $\rightarrow$ | 01000000 |
| 00100000 | $\rightarrow$ | 00100000 |
| 00010000 | $\rightarrow$ | 00010000 |
| 00001000 | $\rightarrow$ | 00001000 |
| 00000100 | $\rightarrow$ | 00000100 |
| 00000010 | $\rightarrow$ | 00000010 |
| 00000001 | $\rightarrow$ | 00000001 |

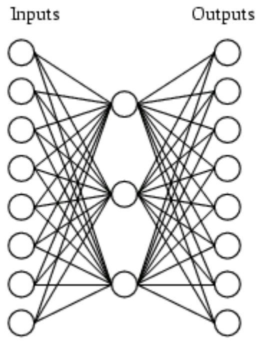
Can this be learned??

# Training


Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

# Learning Hidden Layer Representations

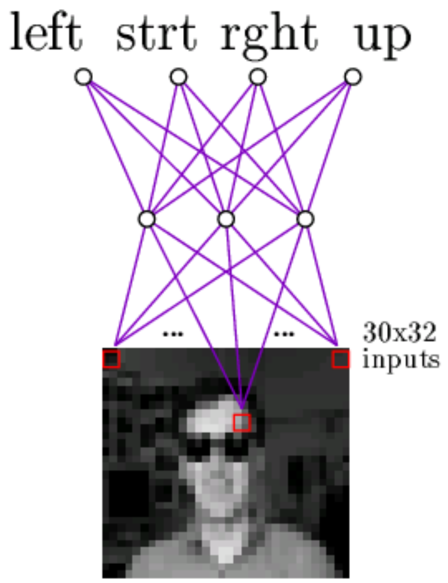
A network:



Learned hidden layer representation:

| Input | | Hidden Values | | | | Output |
|---|---|---|---|---|---|---|
| 10000000 | $\rightarrow$ | .89 | .04 | .08 | $\rightarrow$ | 10000000 |
| 01000000 | $\rightarrow$ | .01 | .11 | .88 | $\rightarrow$ | 01000000 |
| 00100000 | $\rightarrow$ | .01 | .97 | .27 | $\rightarrow$ | 00100000 |
| 00010000 | $\rightarrow$ | .99 | .97 | .71 | $\rightarrow$ | 00010000 |
| 00001000 | $\rightarrow$ | .03 | .05 | .02 | $\rightarrow$ | 00001000 |
| 00000100 | $\rightarrow$ | .22 | .99 | .99 | $\rightarrow$ | 00000100 |
| 00000010 | $\rightarrow$ | .80 | .01 | .98 | $\rightarrow$ | 00000010 |
| 00000001 | $\rightarrow$ | .60 | .94 | .01 | $\rightarrow$ | 00000001 |

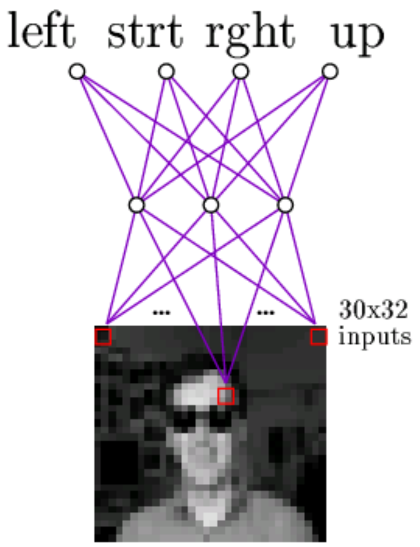
# Neural Nets for Face Recognition

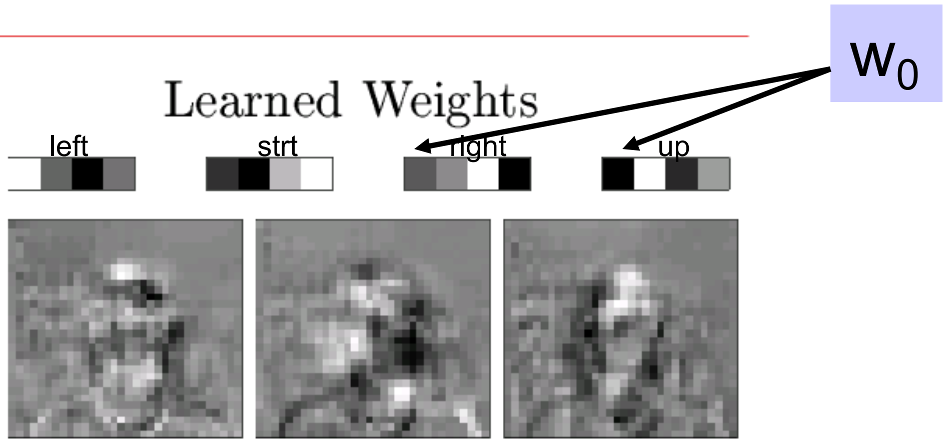


Typical input images

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

# Learned Hidden Unit Weights





Typical input images

http://www.cs.cmu.edu/~tom/faces.html

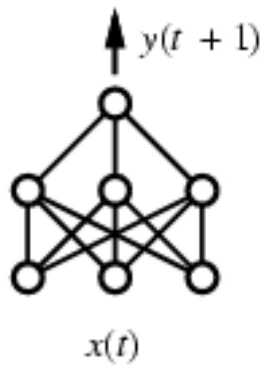
## Training Networks on Time Series

- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns

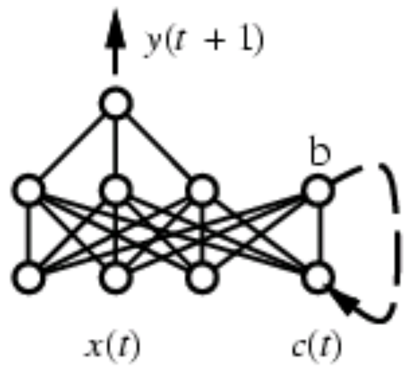
## Training Networks on Time Series

- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns

- Idea: use hidden layer in network to capture state history



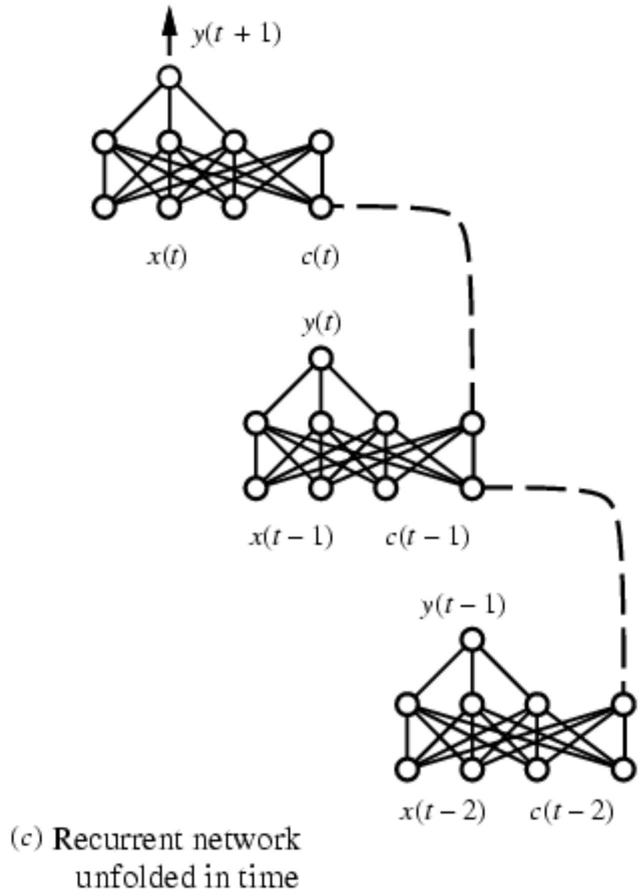
(a) Feedforward network



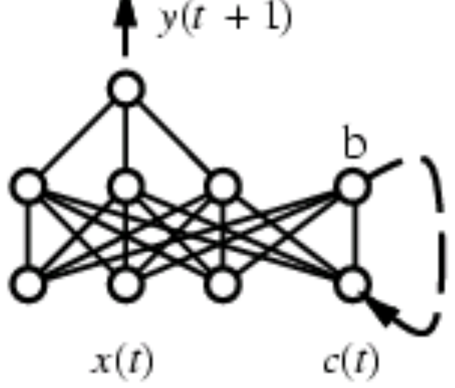
(b) Recurrent network

## Training Networks on Time Series

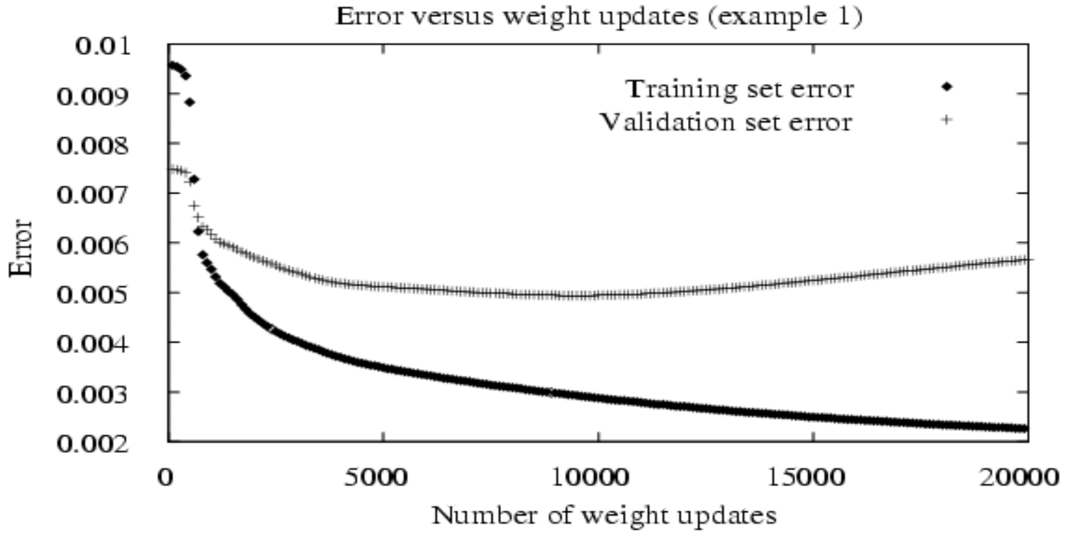
How can we train recurrent net??

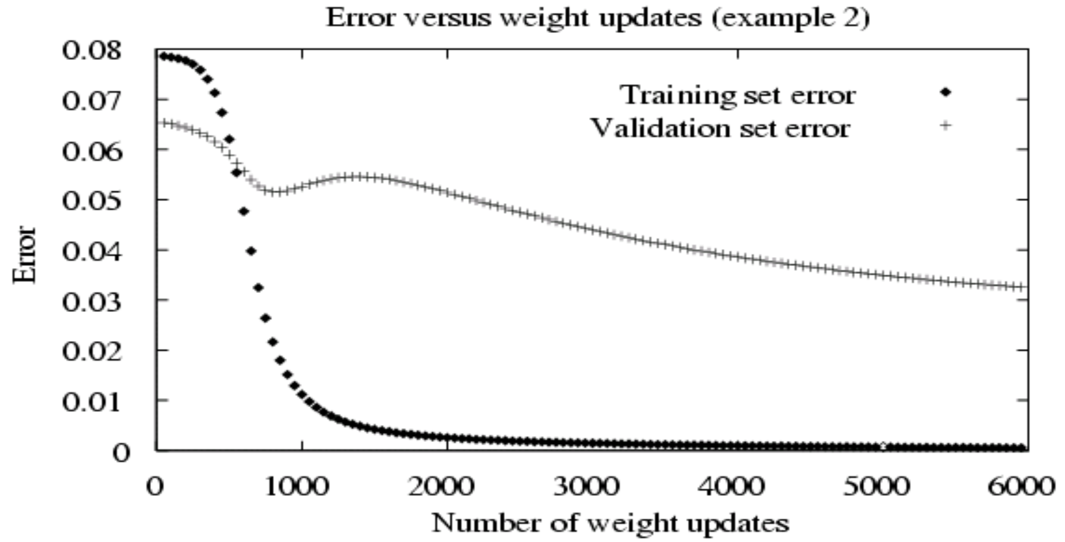


(c) Recurrent network unfolded in time



# Overfitting in ANNs





## How to avoid overfitting?

Regularization – train neural network by maximize M(C)AP

Early stopping

Regulate # hidden units – prevents overly complex models
≡ dimensionality reduction

# Artificial Neural Networks: Summary

- Actively used to model distributed computation in brain
- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially millions of parameters to estimate - overfitting
- Hidden layers learn intermediate representations – how many to use?

- Prediction – Forward propagation
- Gradient descent (Back-propagation), local minima problems

- Coming back in new form as deep networks