Regularized Least Squares

What if $(\mathbf{A}^T\mathbf{A})$ is not invertible ?

r equations , p unknowns – underdetermined system of linear equations
many feasible solutions

Need to constrain solution further

e.g. bias solution to "small" values of β (small changes in input don't translate to large changes in output)

$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda\|\beta\|_2^2$$

Ridge Regression
(l2 penalty)

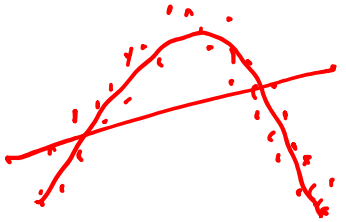
$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda\|\beta\|_1$$

$\lambda \geq 0$

Lasso
(l1 penalty)

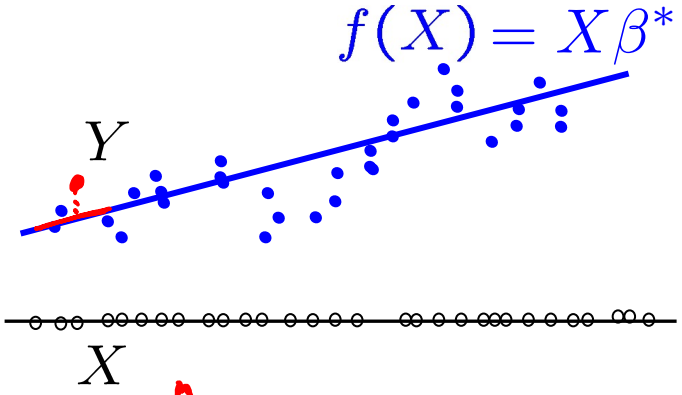
Many β can be zero – many inputs are irrelevant to prediction in high-dimensional settings (typically intercept term not penalized)

Least Squares and M(C)LE



Intuition: Signal plus (zero-mean) Noise model

$$Y = f^*(X) + \epsilon = X\beta^* + \epsilon$$



$$\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I}) \quad Y \underset{Y|X}{\sim} \mathcal{N}(X\beta^*, \sigma^2 \mathbf{I})$$

$$\widehat{\beta}_{\mathsf{MLE}} = \arg\max_{\beta} \underbrace{\log p(\{Y_i\}_{i=1}^n | \beta, \sigma^2, \{X_i\}_{i=1}^n)}_{\text{Conditional log likelihood}} \qquad \prod_{i=1}^{n} P(Y_i | X_i; \beta, \sigma^2)$$

$$= \arg\min_{\beta} \sum_{i=1}^{n} (X_i\beta - Y_i)^2 = \widehat{\beta}$$

is linear!

Least Square Estimate is same as Maximum Conditional Likelihood Estimate under a Gaussian noise model ! → 4 true Loss

Regularized Least Squares and M(C)AP

What if $(\mathbf{A}^T\mathbf{A})$ is not invertible ?

Prior for β = $p(\beta)$

Posterior for $\beta \propto p(D|\beta)\,p(\beta)$

$$\widehat{\beta}_{\text{MAP}} = \arg\max_{\beta} \underbrace{\log p(\{Y_i\}_{i=1}^n|\beta,\sigma^2,\{X_i\}_{i=}^n}_{\text{Conditional log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

$\log p(D|\beta)$

$\log p(\beta) \leftrightarrow \lambda\|\beta\|$

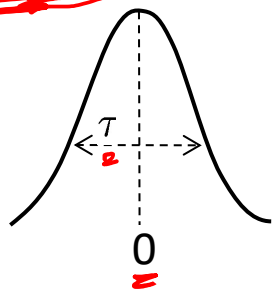
$\|\beta\|_2^2$

I) Gaussian Prior

$\|\beta\|_2^2 = \sum_j \beta_j^2$

$$\beta \sim \mathcal{N}(0,\tau^2\mathbf{I}) \qquad p(\beta) \propto e^{-\beta^T\beta/2\tau^2}$$

$\log p(\beta) \propto -\|\beta\|^2/2\tau^2$



$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n}(Y_i - X_i\beta)^2 + \lambda\|\beta\|_2^2$$

$\downarrow$

$\text{constant}(\sigma^2,\tau^2)$

Ridge Regression

Prior belief that β is Gaussian with zero-mean biases solution to “small” β

Regularized Least Squares and M(C)AP

What if $(\mathbf{A}^T\mathbf{A})$ is not invertible ?

$\log p(\beta) \equiv \lambda \|\beta\|_1$

$p(\beta) \propto e^{-\lambda\|\beta\|_1}$

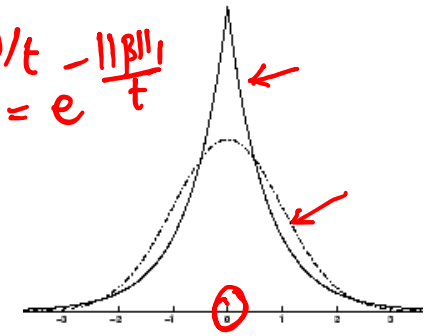
$$\widehat{\beta}_{\text{MAP}} = \arg\max_{\beta} \underbrace{\log p(\{Y_i\}_{i=1}^n | \beta, \sigma^2, \{X_i\}_{i=}^n}_{\text{Conditional log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

II) Laplace Prior

$$\beta_i \overset{iid}{\sim} \text{Laplace}(0, t)$$

$$p(\beta_i) \propto e^{-|\beta_i|/t}$$

$p(\beta) = \prod_i p(\beta_i) \propto e^{-\sum_i |\beta_i|/t} = e^{-\frac{\|\beta\|_1}{t}}$



$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda\|\beta\|_1$$

Lasso

$\lambda \downarrow$ constant(σ^2, t)

Prior belief that β is Laplace with zero-mean biases solution to "sparse" β

Polynomial Regression

degree m

Univariate (1-dim) case:

$$f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + \beta_m X^m = \mathbf{X}\beta$$

where $\mathbf{X} = [1 \;\; X \;\; X^2 \ldots X^m], \beta = [\beta_1 \ldots \beta_m]^T$

$$\widehat{\beta} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{Y}$$

$$\widehat{f}_n(X) = \mathbf{X}\widehat{\beta}$$

where $\mathbf{A} = \begin{bmatrix} 1 & X_1 & X_1^2 & \ldots & X_1^m \\ \vdots & & & \ddots & \vdots \\ 1 & X_n & X_n^2 & \ldots & X_n^m \end{bmatrix}$

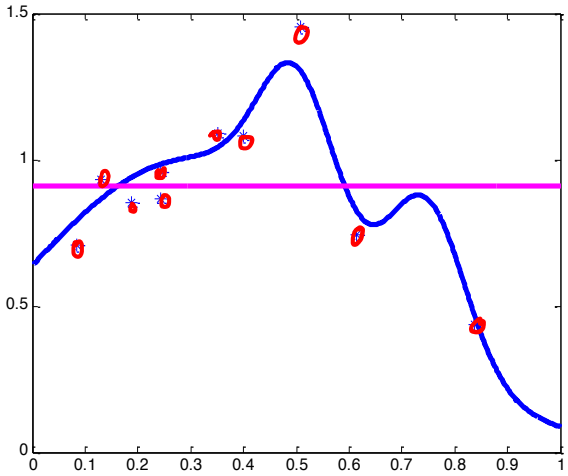
Multivariate (p-dim) case:

$$\begin{aligned} f(X) \;=\; & \beta_0 + \beta_1 X^{(1)} + \beta_2 X^{(2)} + \cdots + \beta_p X^{(p)} \\ & + \sum_{i=1}^{p}\sum_{j=1}^{p} \beta_{ij} X^{(i)} X^{(j)} + \sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{k=1}^{p} X^{(i)} X^{(j)} X^{(k)} \\ & + \ldots \text{terms up to degree m} \end{aligned}$$

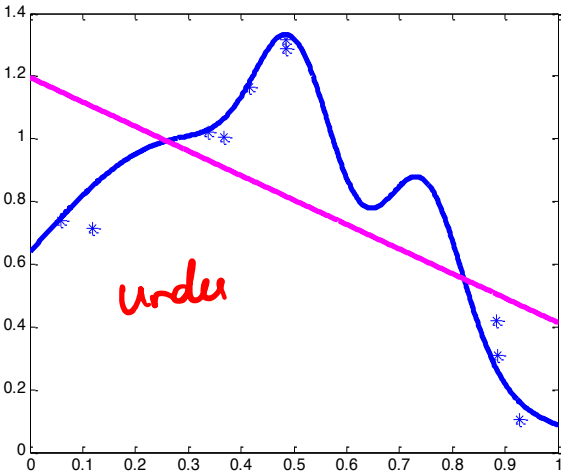
Polynomial Regression

Polynomial of order k, equivalently of degree up to k-1

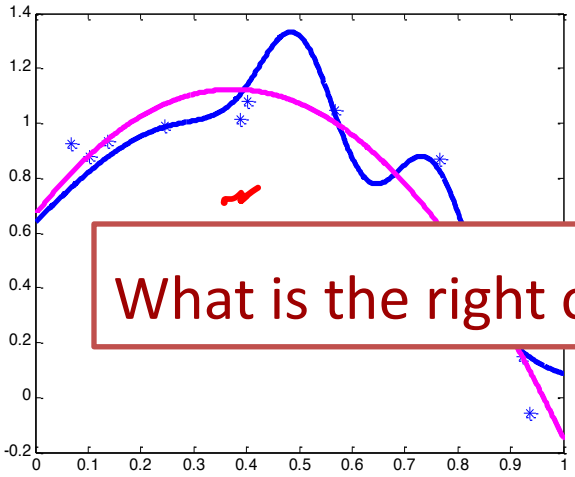
k=1



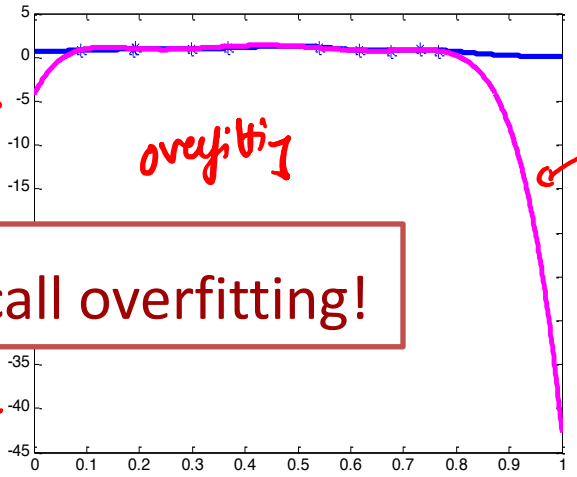
k=2



k=3



k=7



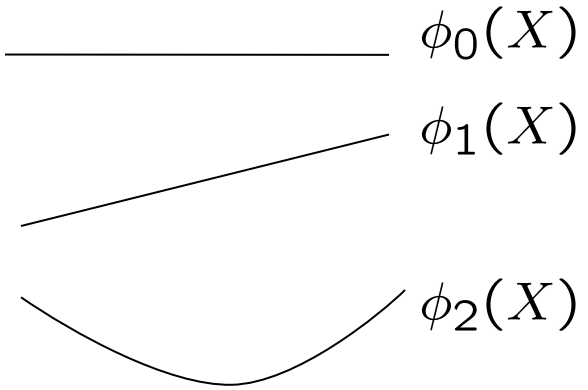
What is the right order? Recall overfitting!

Regression with nonlinear features

$$f(X) = \sum_{j=0}^{m} \beta_j X^j = \sum_{j=0}^{m} \beta_j \phi_j(X)$$

Weight of each feature

Nonlinear features



In general, use any nonlinear features

e.g. e^X, log X, 1/X, sin(X), …

$$\widehat{\beta} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{Y}$$

$$\mathbf{A} = \begin{bmatrix} \phi_0(X_1) & \phi_1(X_1) & \dots & \phi_m(X_1) \\ \vdots & & \ddots & \vdots \\ \phi_0(X_n) & \phi_1(X_n) & \dots & \phi_m(X_n) \end{bmatrix}$$

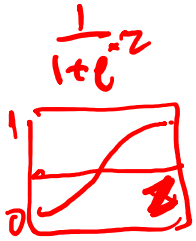
$$\widehat{f}_n(X) = \mathbf{X}\widehat{\beta}$$

$$\mathbf{X} = [\phi_0(X) \ \phi_1(X) \ \dots \ \phi_m(X)]$$

Neural Networks

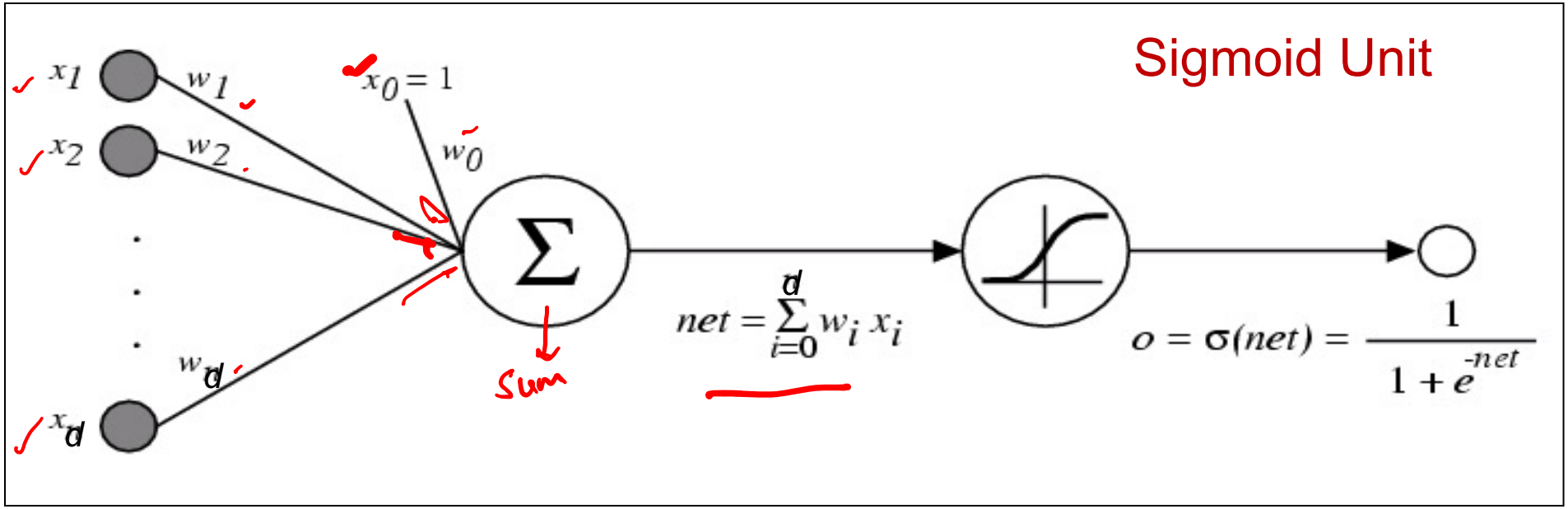
Aarti Singh

Machine Learning 10-315
Sept 27, 2021



Logistic function as a Graph

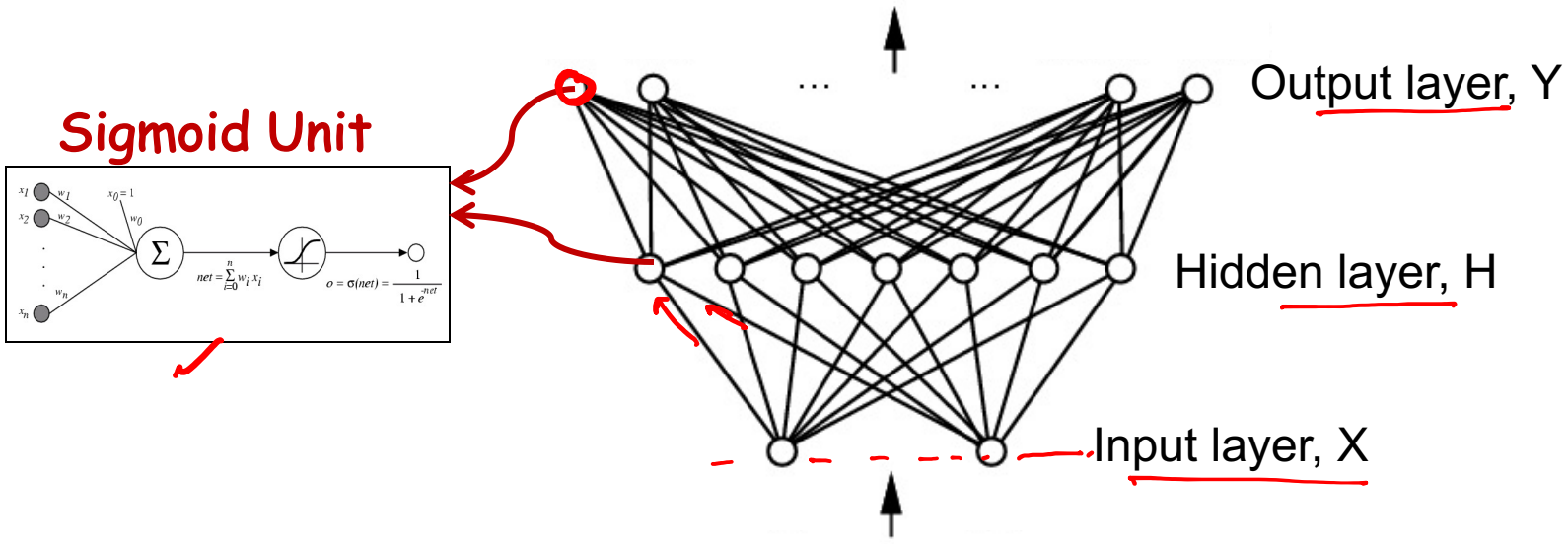
$$\text{Output, } o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$



Neural Networks to learn f: X → Y

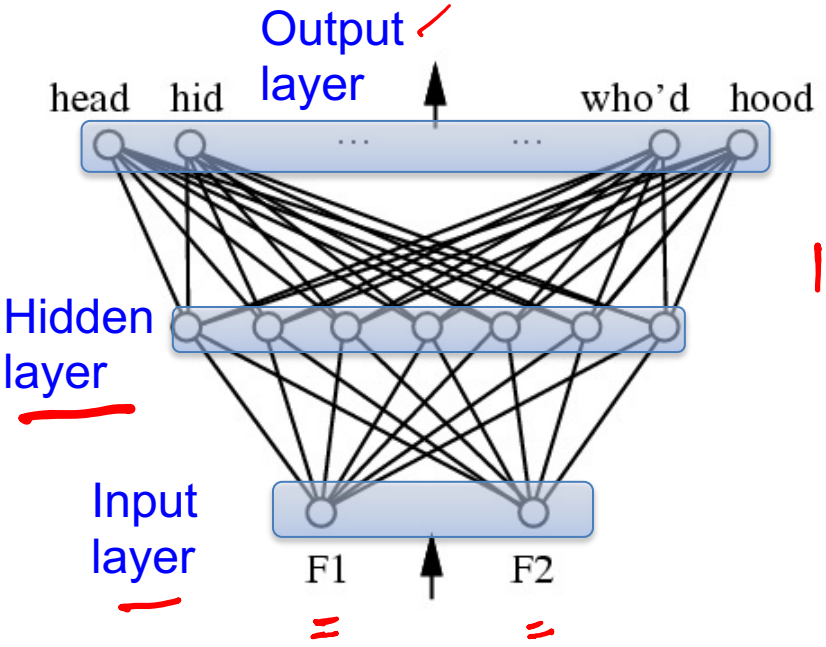
- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

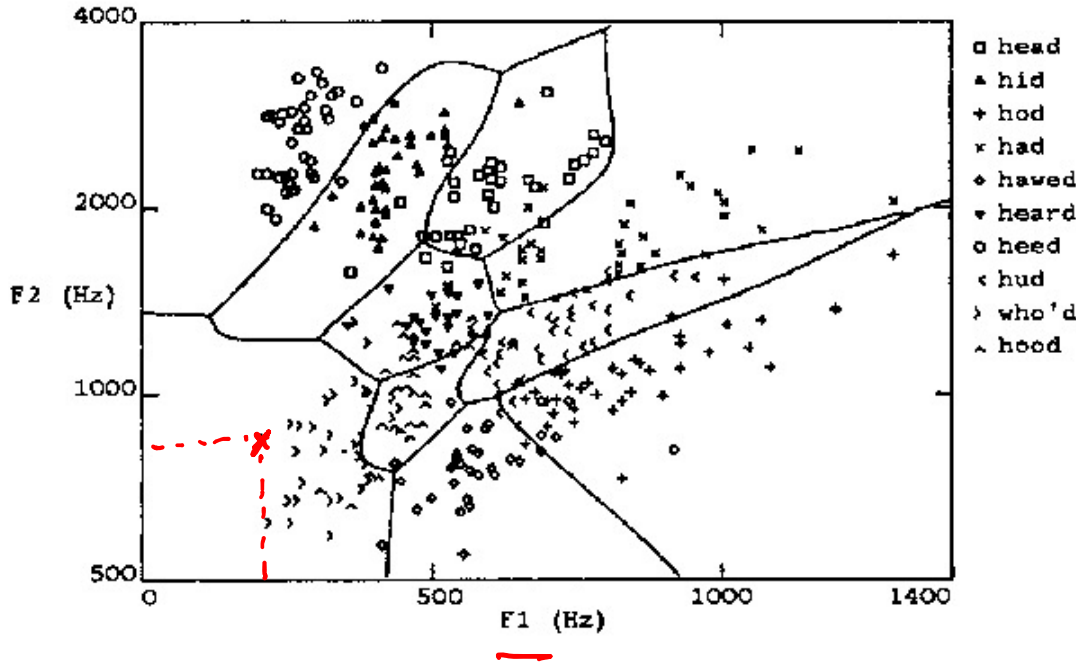
- Neural networks - Represent f by *network* of sigmoid (more recently ReLU – next lecture) units :



Multilayer Networks of Sigmoid Units

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

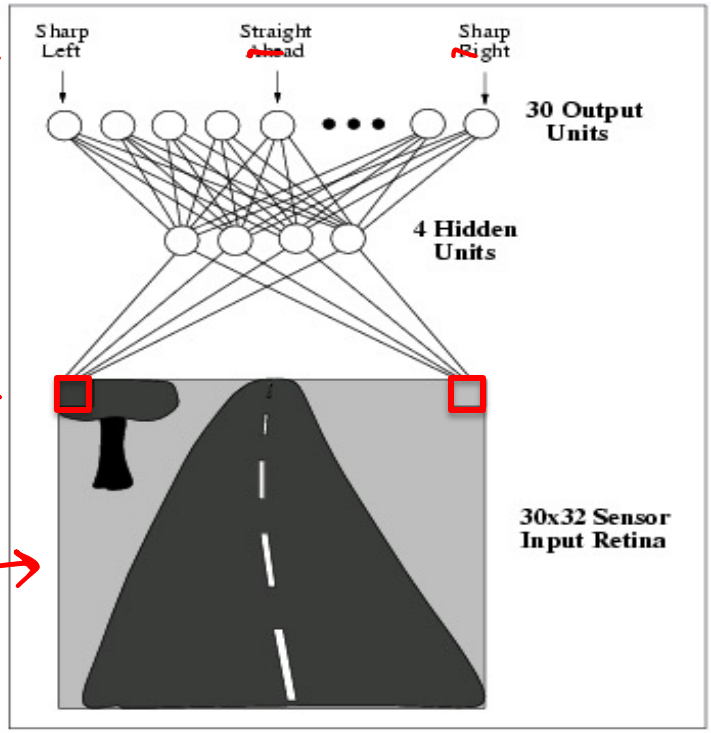




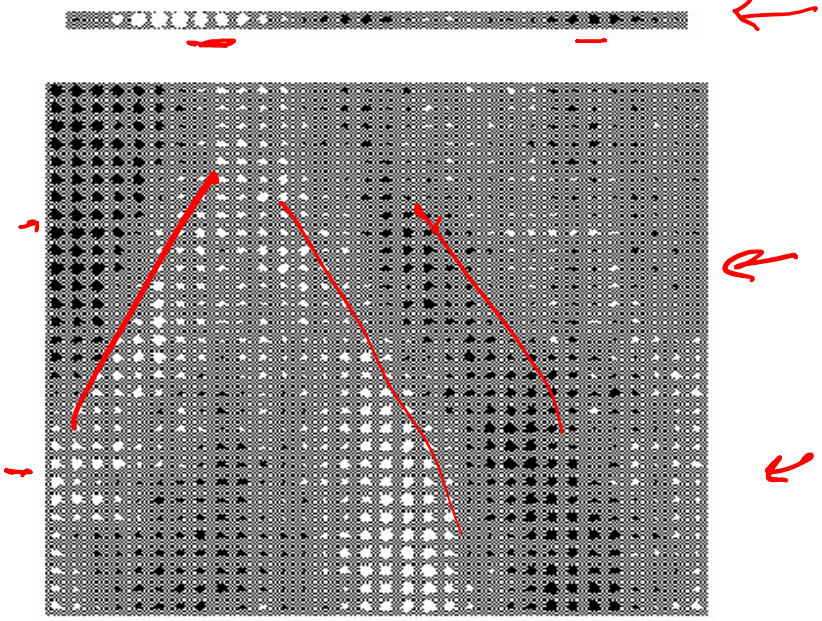
Two layers of logistic units

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

→ 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

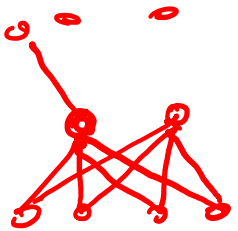
Forward Propagation –

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:

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

o_h

Training Neural Networks – l2 loss

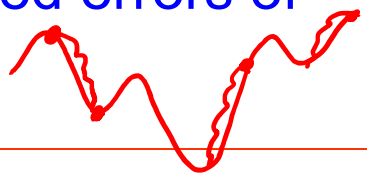
$$W \leftarrow \arg\min_W E[W]$$

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

Learned neural network

Where $\hat{f}(x^l) = o(x^l)$, output of neural network for training point x^l

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



Minimize using Gradient Descent

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

Gradient

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

Training rule:

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

i.e.,

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

Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

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

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

Using all training data *D*

Stochastic

Incremental mode Gradient Descent:
Do until satisfied

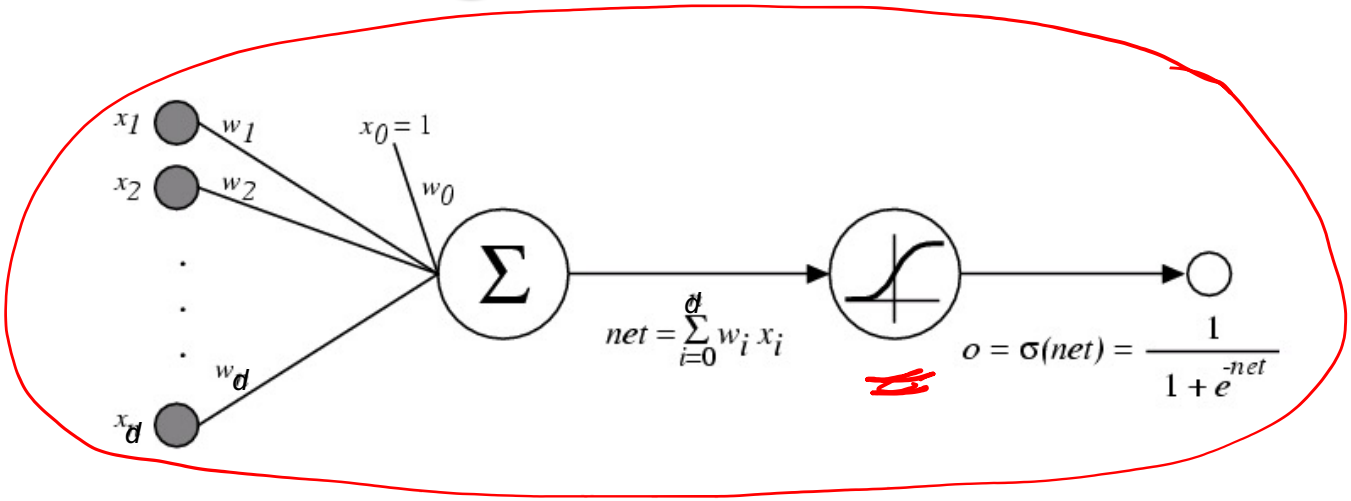
- For each training example l in D
  1. Compute the gradient $\nabla E_l[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_l[\vec{w}]$

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

randomly drawn training point (stochastic)

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Training Neural Networks



$\sigma(x)$ is the sigmoid function

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

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

Differentiable

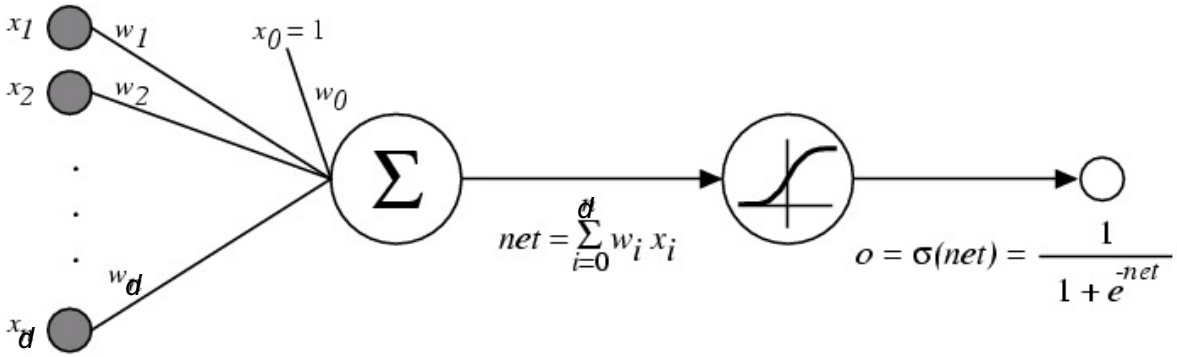
A. $\sigma(x)(1-\sigma(x))$

B. $\sigma(x)\,\sigma(-x)$

C. $-\sigma(x)$

D. $\sigma(x)^2$

Training Neural Networks



$\sigma(x)$ is the sigmoid function

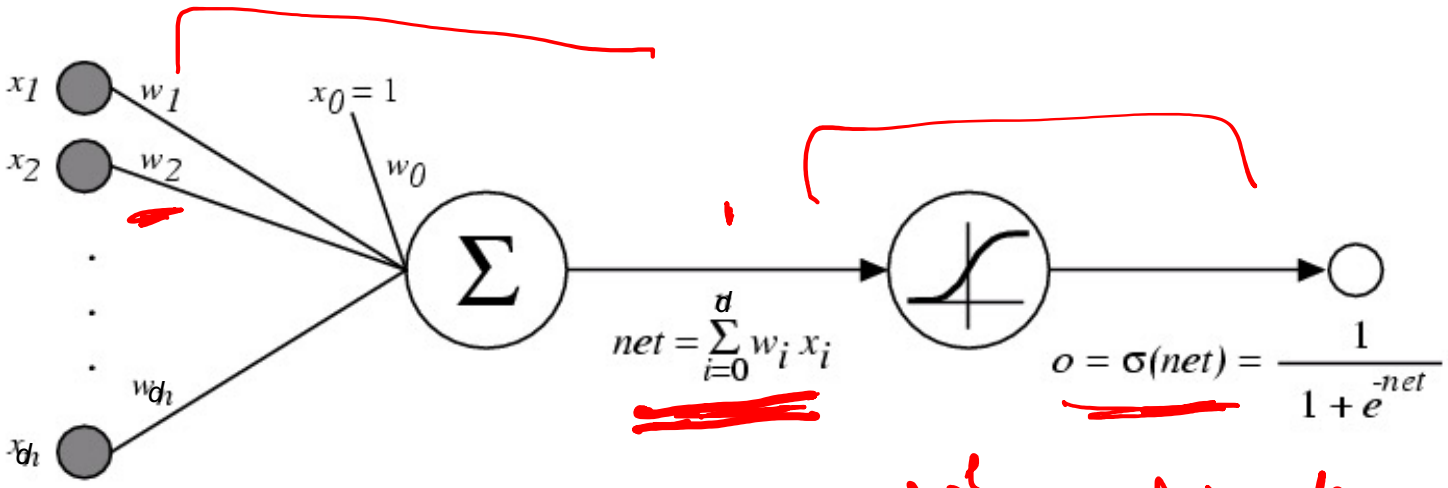
$$\frac{1}{1 + e^{-x}}$$

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 → Backpropagation

Gradient Descent for 1 sigmoid unit



$\frac{\partial E}{\partial o^l} \cdot \frac{\partial o^l}{\partial w_i}$ ← chain rule

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2 = \sum_l (y^l - o^l) \left(-\frac{\partial o^l}{\partial w_i} \right)$$

$\frac{\partial o}{\partial w_i} = \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_i}$

$o(1-o)$ x_i

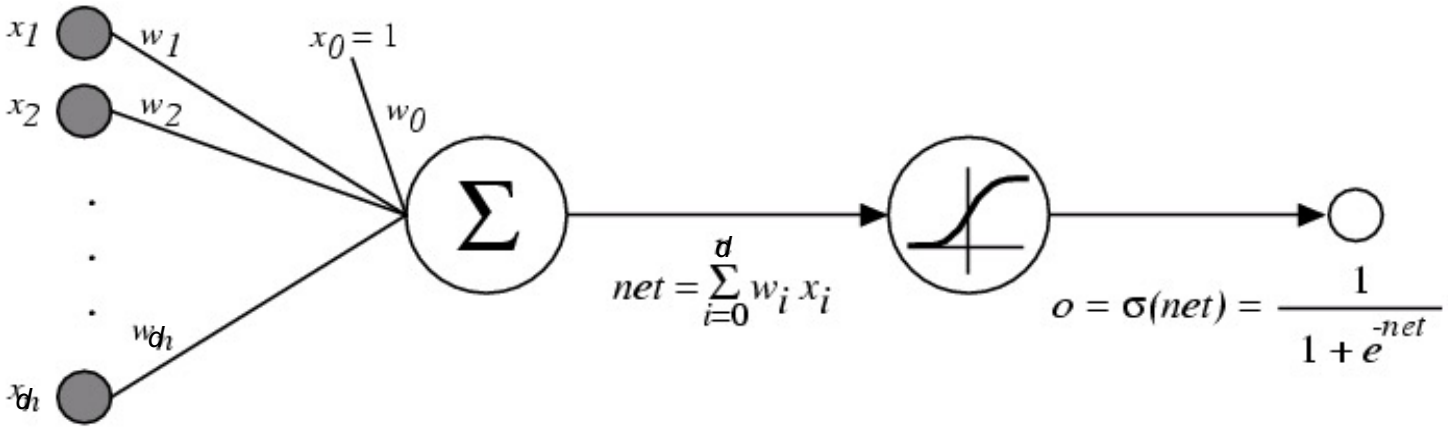
Gradient of the sigmoid function output wrt its input

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

Gradient of the sigmoid unit output wrt input weights

$$\frac{\partial o}{\partial w_i} = o(1-o)\, x_i$$

Gradient Descent for 1 sigmoid unit



$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2 = \sum_l (y^l - o^l) \left(-\frac{\partial o^l}{\partial w_i} \right)$$

Gradient of the sigmoid function output wrt its input

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

Gradient of the sigmoid unit output wrt input weights

$$\frac{\partial o}{\partial w_i} = \frac{\partial o}{\partial net} . \frac{\partial net}{\partial w_i} = o(1 - o) x_i$$

Gradient descent for training NNs

Gradient descent via Chain rule for computing gradients

= Back-propagation algorithm for training NNs

Backpropagation Algorithm (MLE)

using Stochastic gradient descent

1 final output unit

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 → Using Forward propagation
  2. $$\delta \leftarrow o(1-o)(y-o)$$
  3.
  4. Update each network weight $w_{i,j}$

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

where

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

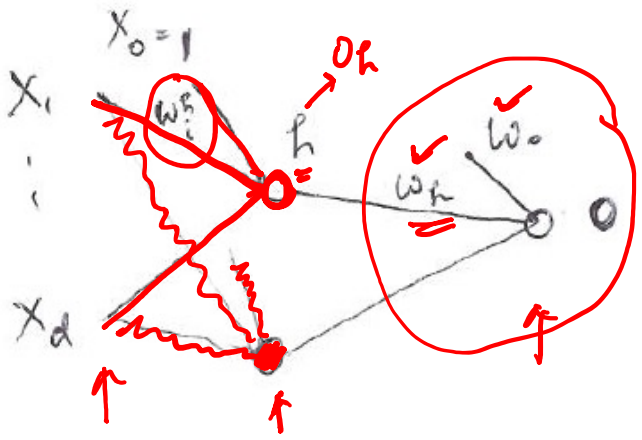
$o(1-o)\, x_i\, (y-o)$

y = label of current training example

w_{ij} = wt from i to j

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

Gradient Descent for 1 hidden layer 1 output NN



$$o = \sigma\left(w_0 + \sum_h w_h o_h\right) \equiv \sigma\left(\sum_h w_h o_h\right)$$

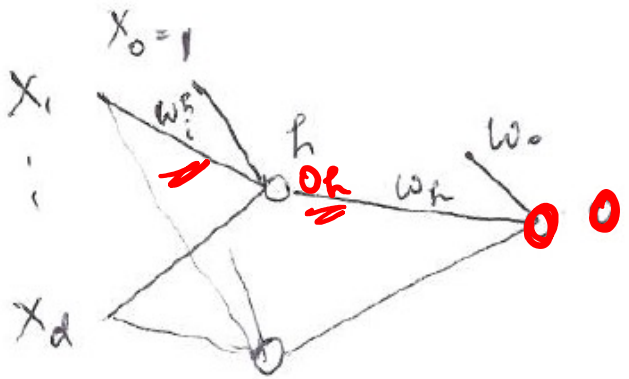
$$o_h = \sigma\left(w_0^h + \sum_i w_i^h x_i\right) \equiv \sigma\left(\sum_i w_i^h x_i\right)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2 = \sum_l (y^l - o^l)\left(-\frac{\partial o^l}{\partial w_i}\right)$$

Gradient of the output with respect to w_h

$$\frac{\partial o}{\partial w_h} = o(1-o)o_h$$

Gradient Descent for 1 hidden layer 1 output NN



$$o = \sigma(w_0 + \sum_h w_h o_h) \equiv \sigma(\sum_h w_h o_h)$$

$$o_h = \sigma(w_0^h + \sum_i w_i^h x_i) \equiv \sigma(\sum_i w_i^h x_i)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{l \in D} (y^l - o^l)^2 = \sum_l (y^l - o^l) \left(-\frac{\partial o^l}{\partial w_i}\right)$$

$\frac{\partial o_h}{\partial w_i^h}$

Gradient of the output with respect to input weights w^h_i

$$\frac{\partial o}{\partial w_i^h} = \frac{\partial o}{\partial o_h} \cdot \frac{\partial o_h}{\partial w_i^h}$$

$$= o(1-o) o_h (1-o_h) w_h x_i$$