

Machine Learning

Prof. Dr. Stefan Kramer
Johannes Gutenberg-Universität Mainz

Outline

- Logistic regression
- Artificial neural networks (ANNs)

Acknowledgements

- Andrew Moore
- Tom Mitchell
- Eibe Frank
- Ian Witten

Logistic Regression

Generative vs. Discriminative Models

- Wish to learn $f: X \rightarrow Y$, or $P(Y|X)$
- *Generative classifiers* (e.g., Naive Bayes)
 - assume some functional form for $P(X|Y)$, $P(Y)$; this is the *generative model*
 - estimate parameters of $P(X|Y)$, $P(Y)$ directly from training data
 - use Bayes rule to calculate $P(Y|X=x_i)$
- *Discriminative classifiers*
 - assume some functional form for $P(Y|X)$, the *discriminative model*
 - estimate parameters of $P(Y|X)$ directly from training data

see: <http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf> 5

Generative vs. Discriminative Models

- Consider learning $f: X \rightarrow Y$, where
 - X is a vector of real-valued features $\langle X_1, \dots, X_n \rangle$ and Y is Boolean
- We could use a so-called *Gaussian Naive Bayes* classifier
 - assume all X_i are conditionally independent given Y
 - model $P(X_i | Y=y_k)$ as Gaussian $N(\mu_{ik}, \sigma)$
 - model $P(Y)$ as Bernoulli ($\pi = P(Y=1)$)
- What does that imply about the form of $P(Y|X)$?

Convenient Form of Model

$$P(Y = 1|X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

!

implies

$$P(Y = 0|X = \langle X_1, \dots, X_n \rangle) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i)$$

linear
classification
rule!

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$

Derive $P(Y|X)$ for Continuous X_i

$$P(Y = 1|X) = \frac{P(Y = 1)P(X|Y = 1)}{P(Y = 1)P(X|Y = 1) + P(Y = 0)P(X|Y = 0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$= \frac{1}{1 + \exp((\ln \frac{1-\pi}{\pi}) + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}$$

$$P(x | y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

$$\sum_i \left(\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2} \right)$$

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

Convenient Form of Model

$$P(Y = 1|X = \langle X_1, \dots, X_n \rangle) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies


$$P(Y = 0|X = \langle X_1, \dots, X_n \rangle) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = \exp(w_0 + \sum_i w_i X_i)$$

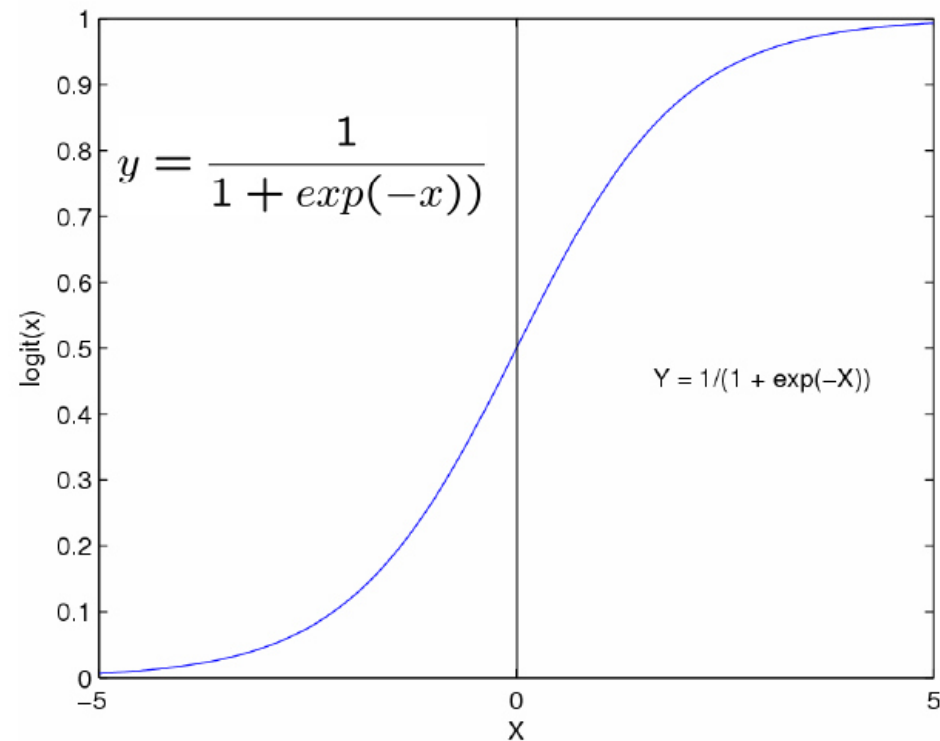
implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$



linear
classification
rule!

Logistic Function



$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

Logistic Regression More Generally

Logistic regression for generally R classes
(case of multi-class classification; classes 1 to R)

⇒ Learn $R - 1$ sets of weights!

for $k < R$

$$P(Y = y_k | X) = \frac{\exp(w_{k0} + \sum_{i=1}^n w_{ki} X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji} X_i)}$$

for $k = R$

$$P(Y = y_R | X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^n w_{ji} X_i)}$$

Training Logistic Regression: MLCE

- Choose parameters $W = \langle w_0, \dots, w_n \rangle$ to maximize conditional likelihood of training data, where

$$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

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

*flipped for
convenience
of derivation!*

- Training data $D = \{\langle X^1, Y^1 \rangle, \dots, \langle X^L, Y^L \rangle\}$
- Data likelihood $\prod_l P(X^l, Y^l|W)$
- Data conditional likelihood $\prod_l P(Y^l|X^l, W)$

$$W_{MCLE} = \arg \max_W \prod_l P(Y^l|W, X^l)$$

Expressing Conditional Log Likelihood

$$l(W) \equiv \ln \prod_l P(Y^l | X^l, W) = \sum_l \ln P(Y^l | X^l, W)$$

$$P(Y = 0 | X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

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

Only one term
can be non-zero!

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1 | X^l, W) + (1 - Y^l) \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1 | X^l, W)}{P(Y^l = 0 | X^l, W)} + \ln P(Y^l = 0 | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

Expressing Conditional Log Likelihood

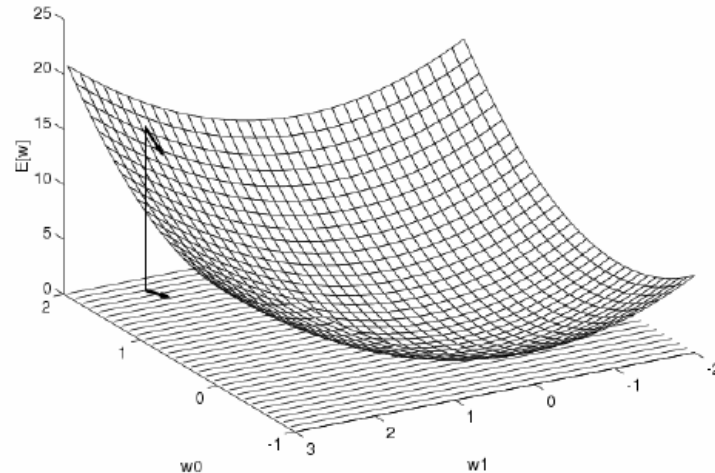
$$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

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

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i w_i X_i^l)) \end{aligned}$$

- Good news: $l(W)$ is a concave function of W
- Bad news: no closed-form solution to maximize $l(W)$

Gradient Descent



Gradient

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

Training rule:

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

i.e.,

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

Maximize Conditional Log Likelihood: Gradient Ascent

$$\begin{aligned}l(W) &\equiv \ln \prod_l P(Y^l | X^l, W) \\&= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i^n w_i X_i^l))\end{aligned}$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

Gradient ascent algorithm: iterate until change $< \varepsilon$

For all i , $w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$
repeat

Logistic Regression

- Functional form follows from Naïve Bayes assumptions, but training procedure picks parameters without the conditional independence assumption
 - MLE training: pick W to maximize $P(Y|X, W)$
 - (MAP training: pick W to maximize $P(W|X, Y)$)
- Gradient ascent/descent: general approach when closed-form solutions unavailable
- Generative vs. discriminative models: difference in data requirements
- As with linear regression: often used with forward selection or backward elimination

Artificial Neural Networks (ANNs)

Properties of Artificial Neural Nets (ANNs)

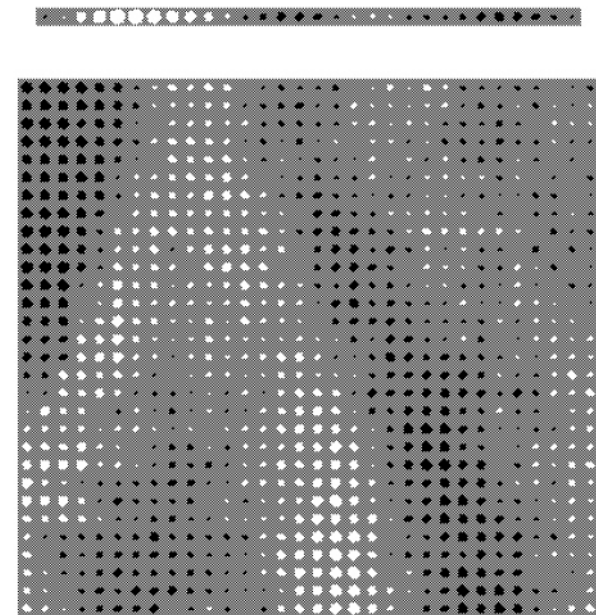
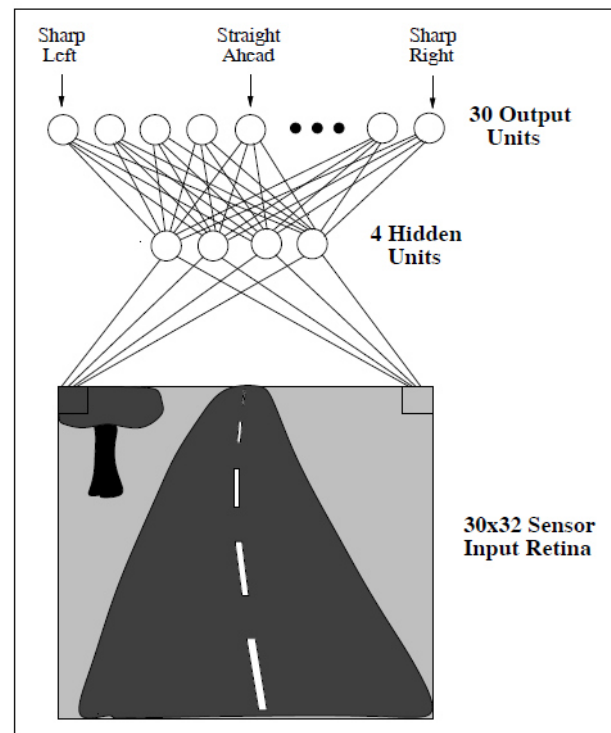
- Many neuron-*like threshold switching units*
- Many *weighted interconnections* among units
- Highly *parallel, distributed* process
- Emphasis on *tuning weights automatically*

Uses of ANNs

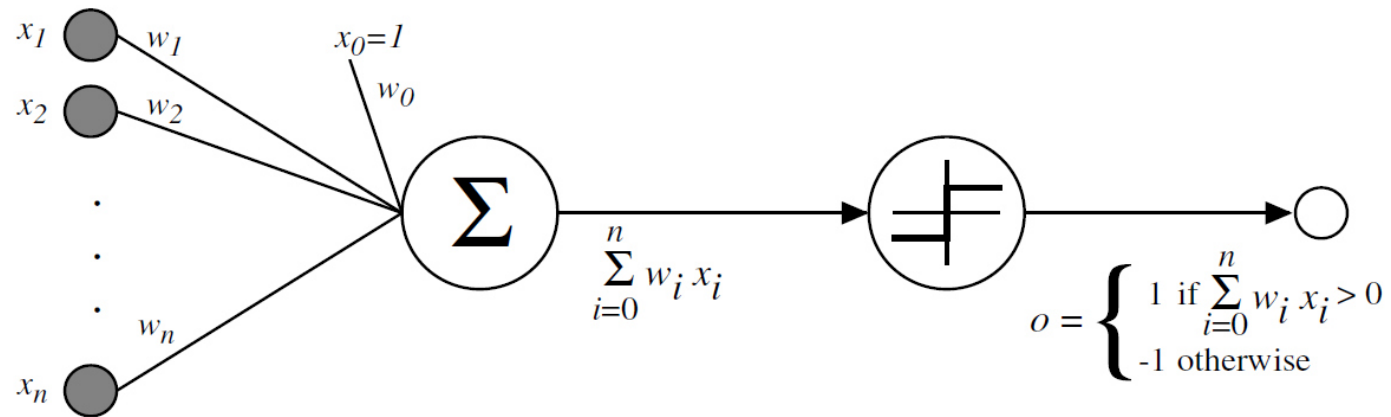
- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant
- *Examples:* speech phoneme recognition, image classification, financial prediction

ALVINN

- ... drives 70 mph on highway
- later: DARPA grand challenges 2004-2007
- weights to one hidden unit and from the same hidden unit (white is large positive, black is large negative)



Perceptron

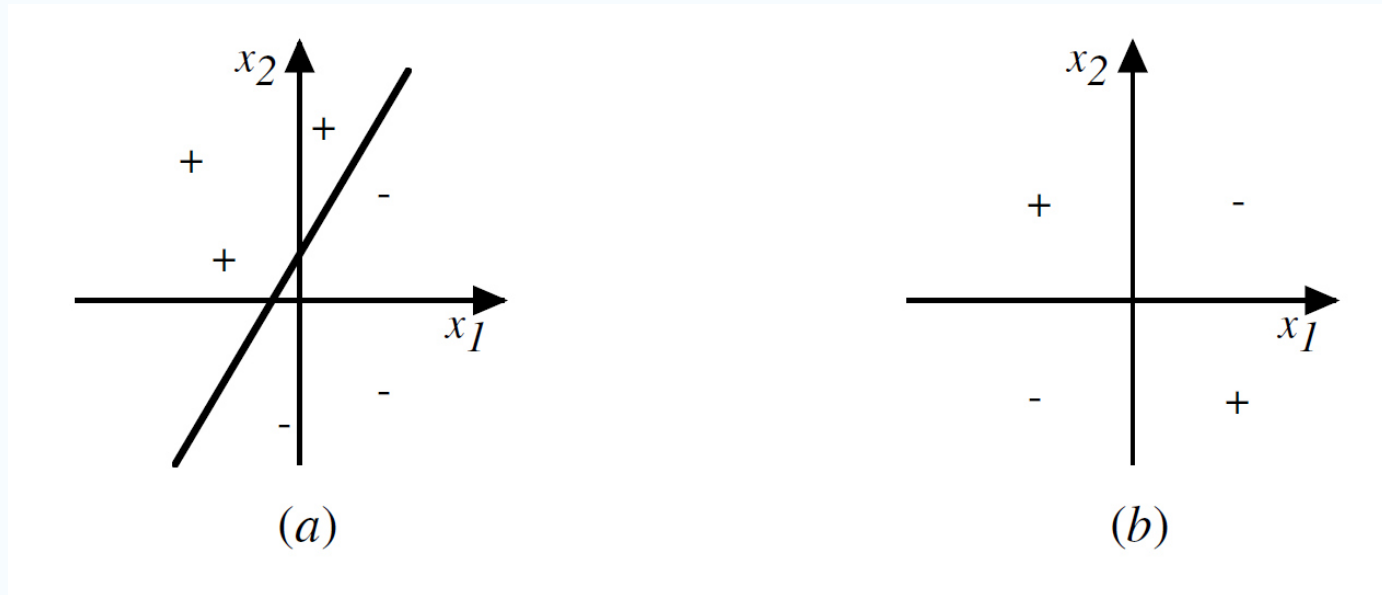


$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

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

Decision Surface of a Perceptron



Represents some useful functions (what weights represent $g(x_1, x_2) = \text{AND}(x_1, x_2)$?), but some functions are not representable, e.g., not linearly separable functions

Perceptron Training Rule

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

where

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

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Provably converges if training data linearly separable and learning rate sufficiently small

Gradient Descent

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1x_1 + \cdots + w_nx_n$$

Let's learn w_i 's that minimize the squared error

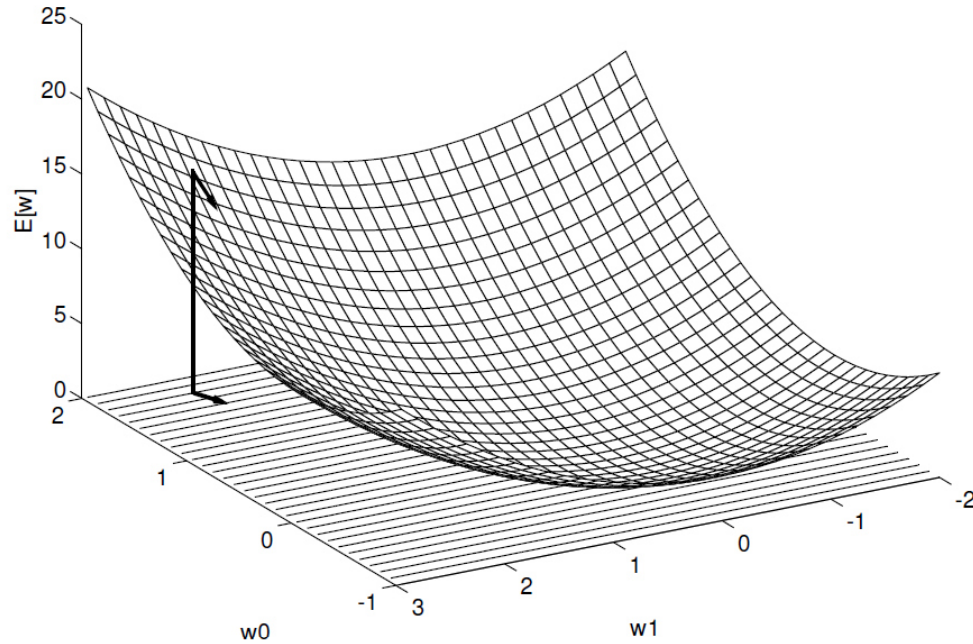
$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

where D is set of training examples.

Gradient Descent

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\ \frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d})\end{aligned}$$

Gradient Descent Again



Gradient

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

Training rule:

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

i.e.,

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

GradientDescent(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output.

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - * Input the instance \vec{x} to the unit and compute the output o
 - * For each linear unit weight w_i , Do
$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$
 - For each linear unit weight w_i , Do

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

Summary Perceptron

- Perceptron training rule guaranteed to succeed if training examples are linearly separable and learning rate sufficiently small
- Linear unit training rule uses gradient descent
- Guaranteed to converge to hypothesis with minimum squared error (a) given sufficiently small learning rate (b) even when training data contains noise and (c) even when training data not separable by H

Incremental (Stochastic) Gradient Descent

*Batch mode
gradient descent*

Do until satisfied

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

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

*Incremental gradient
descent*

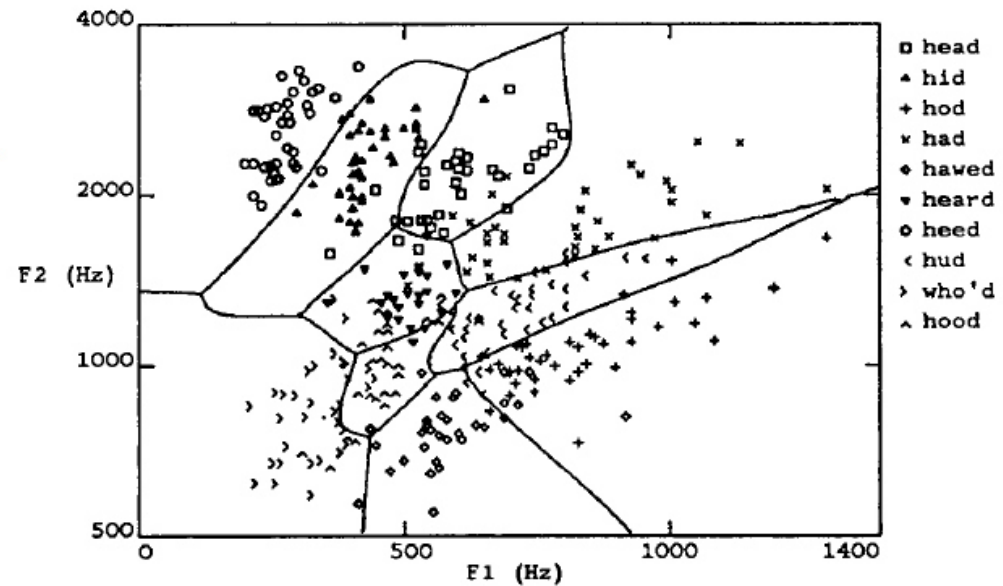
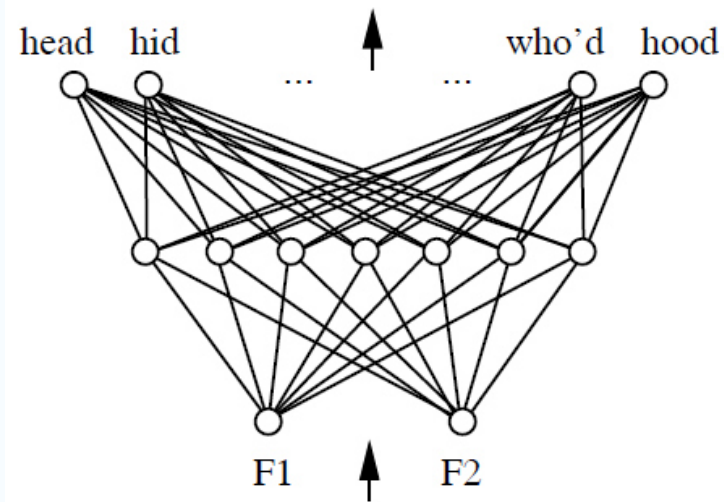
Do until satisfied

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

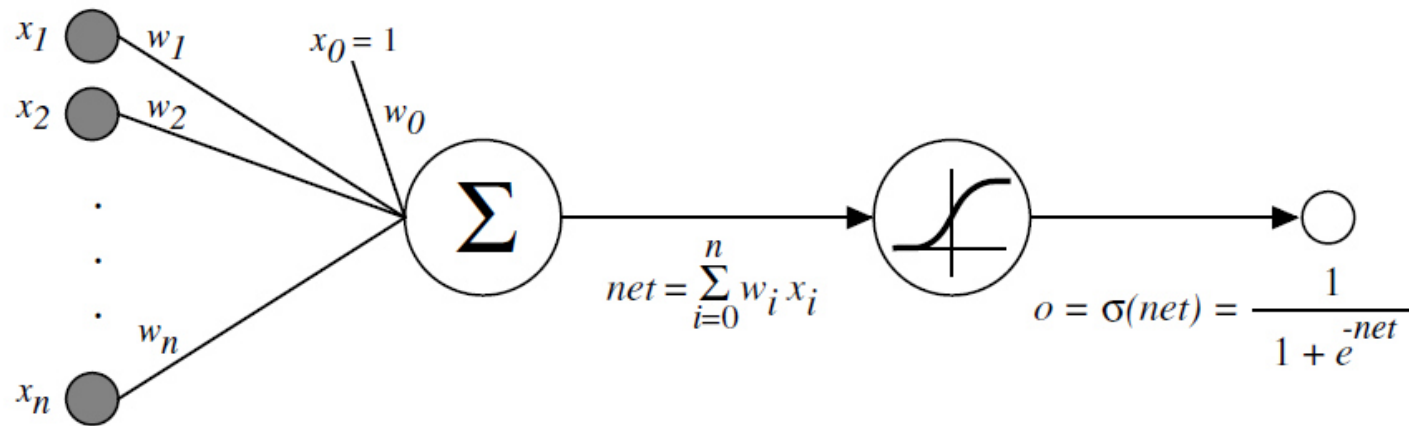
$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental gradient descent can approximate batch gradient descent arbitrarily closely if learning rate made small enough

Multilayer Networks of Sigmoid Units



Sigmoid Unit



$\sigma(x)$ is the sigmoid function

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

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

*We can derive gradient descent rules to train (a) one sigmoid unit or even (b) **multilayer networks of sigmoid units : backpropagation***³²

Error Gradient for a Sigmoid Unit

$$net_j = \sum_i w_{ji} x_{ji}$$

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\ &= - \sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}\end{aligned}$$

But we know:

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

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

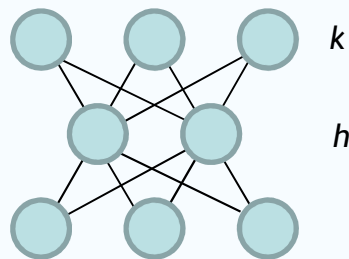
So:

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

Algorithm Back- propagation

w_{ij} denotes the weight
from unit i to unit j

x_{ij} denotes the input
from unit i to unit j
(connected via edge
with weight w_{ij})



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

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

3. For each hidden unit h

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

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

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

where

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