Machine Learning 10-601

Tom M. Mitchell
Machine Learning Department
Carnegie Mellon University

September 20, 2017

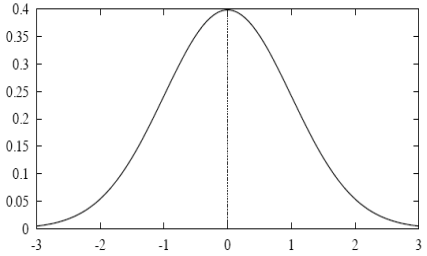
Today:
- Gradient ascent/descent
- Discriminative vs. Generrative
- Begin neural nets

Readings: (see class website)

Required:
- Mitchell: “Naïve Bayes and Logistic Regression”
- Goodfellow: Deep networks

What if we have continuous X_i ?

Gaussian Naïve Bayes (GNB): assume



$$p(X_i = x | Y = y_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} \; e^{-\frac{1}{2}\left(\frac{x-\mu_{ik}}{\sigma_{ik}}\right)^2}$$

Sometimes assume variance σ_{ik}

- is independent of Y (i.e., σ_i),
- or independent of X_i (i.e., σ_k)
- or both (i.e., σ)

Estimating Parameters: Y discrete, X_i continuous

Maximum likelihood estimates:

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$$

jth training example

ith feature

kth class

δ()=1 if (Y^j=y_k) else 0

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k)$$

What is the minimum possible error?

Best case:

- conditional independence assumption is satistied
- we know P(Y), P(X|Y) perfectly (e.g., infinite training data)

Gaussian Naïve Bayes – Big Picture

Example: Y= PlayBasketball (boolean), X1=Height, X2=MLgrade

$$Y^{new} \leftarrow \arg\max_{y \in \{0,1\}} P(Y=y) \prod_i P(X_i^{new} | Y=y)$$ assume P(Y=1) = 0.5

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, < X_1 … X_n >
  - Y is boolean
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik},\sigma_i)$
  - model P(Y) as Bernoulli (π)

- What does that imply about the form of P(Y|X)?

$$P(Y = 1|X =< X_1, ...X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

Logistic regression more generally

- Logistic regression when Y not boolean (but still discrete-valued).
- Now $y \in \{y_1 \ldots y_R\}$: 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*

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

Logistic regression more generally: Softmax

- Logistic regression when Y not boolean (but still discrete-valued).
- Now $y \in \{y_1 \ldots y_R\}$: learn R sets of weights

for each k: $$P(Y = y_k|X) = \frac{exp(w_{k0} + \sum_i w_{ki} X_i)}{\sum_{j=1}^{R} exp(w_{j0} + \sum_i w_{ji} X_i)}$$

Training Logistic Regression: MCLE

- Choose parameters W=<w_0, ... w_n> to <u>maximize conditional likelihood</u> of training data

where

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

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

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

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

Maximizing Conditional Log Likelihood

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

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

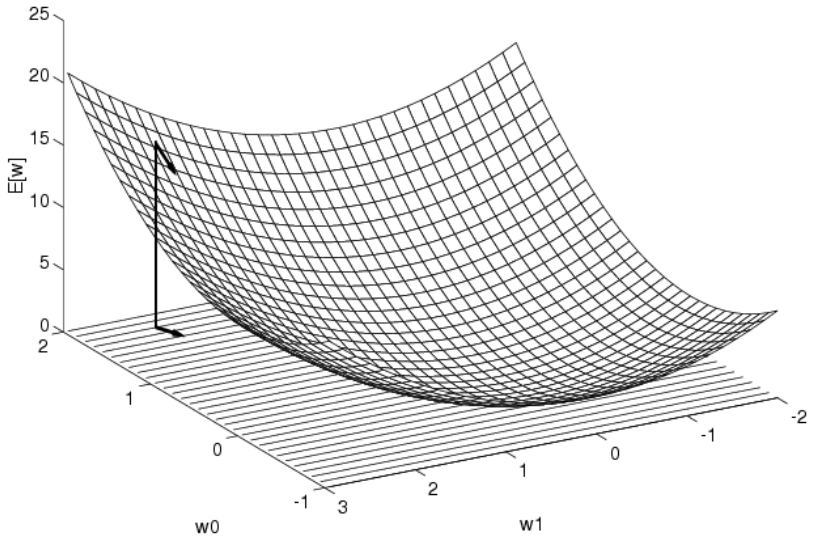
$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W) = \sum_l \ln P(Y^l|X^l, W)$$

$$= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

Good news: $l(W)$ is concave function of W

Bad news: no closed-form solution to maximize $l(W)$

Gradient Descent



Gradient

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

Training rule:

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

i.e.,

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

Gradient Descent:

Batch gradient: use error $E_D(\mathbf{w})$ over entire training set D

Do until satisfied:

1. Compute the gradient $\nabla E_D(\mathbf{w}) = \left[\frac{\partial E_D(\mathbf{w})}{\partial w_0} \ldots \frac{\partial E_D(\mathbf{w})}{\partial w_n}\right]$
2. Update the vector of parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_D(\mathbf{w})$

Stochastic gradient: use error $E_d(\mathbf{w})$ over single examples $d \in D$

Do until satisfied:

1. Choose (with replacement) a random training example $d \in D$
2. Compute the gradient just for d: $\nabla E_d(\mathbf{w}) = \left[\frac{\partial E_d(\mathbf{w})}{\partial w_0} \ldots \frac{\partial E_d(\mathbf{w})}{\partial w_n}\right]$
3. Update the vector of parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_d(\mathbf{w})$

Stochastic approximates Batch arbitrarily closely as $\eta \rightarrow 0$

Stochastic can be much faster when D is very large

Intermediate approach: use error over subsets of D

Maximize Conditional Log Likelihood: Gradient Ascent

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

$$= \sum_l Y^l(w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

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

Maximize Conditional Log Likelihood: Gradient Ascent

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

$$= \sum_l Y^l(w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

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

Gradient ascent algorithm: iterate until change < ε

For all i, repeat

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

That's all for M(C)LE. How about MAP?

- For MAP, need to define prior on W
  - given W = $\langle w_1, \ldots w_n \rangle$
    let's assume prior $P(w_i) = N(0, \sigma)$
  - i.e., assume zero mean, Gaussian prior for each w_i

- A kind of Occam's razor (simplest is best) prior

- Helps avoid very large weights and overfitting

MAP Estimates for Logistic Regression

$$W^{MAP} = \arg\max_{W} P(W|Y,X) = \frac{P(Y|W,X)P(W,X)}{P(Y,X)}$$

MAP Estimates for Logistic Regression

$$W^{MAP} = \arg\max_W P(W|Y,X) = \frac{P(Y|W,X)P(W,X)}{P(Y,X)}$$

$$= \arg\max_W P(Y|W,X)P(W,X)$$

$$= \arg\max_W P(Y|W,X)P(W)P(X)$$

$$= \arg\max_W P(Y|W,X)P(W)$$

let's assume P(W,X)=P(W) P(X)

$$W^{MAP} = \arg\max_W [\ln P(Y|W,X) + \ln P(W)]$$

zero mean Gaussian P(W)

$$P(W) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\sum_i w_i^2\right)$$

$$W^{MAP} = \arg\max_W [\ln P(Y|W,X) - \left(\frac{1}{2\sigma^2}\sum_i w_i^2\right)]$$

MLE vs MAP

- Maximum conditional likelihood estimate

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

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

- Maximum a posteriori estimate with prior W~N(0,σI)

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

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

MAP estimates and Regularization

- Maximum a posteriori estimate with prior W~N(0,σI)

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

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

called a "regularization" term

- helps reduce overfitting
- if P(W) is Gaussian, then encourages W to be near the mean of P(W) : zero here, but can easily use any mean
- used very frequently in Logistic Regression

The Bottom Line

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, < X_1 … X_n >
  - Y is boolean
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik},\sigma_i)$
  - model P(Y) as Bernoulli (π)

- Then P(Y|X) is of this form, and we can directly estimate W

$$P(Y = 1|X =< X_1, ...X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

- Furthermore, same holds if the X_i are boolean
  - trying proving that to yourself

What you should know:

- Logistic regression
  - Functional form follows from Naïve Bayes assumptions
    - For Gaussian Naïve Bayes assuming variance $\sigma_{i,k} = \sigma_i$
    - For discrete-valued Naïve Bayes too
  - But training procedure picks parameters without making conditional independence assumption
  - MLE training: pick W to maximize P(Y | X, W)
  - MAP training: pick W to maximize P(W | X,Y)
    - 'regularization'
    - helps reduce overfitting

- Gradient ascent/descent
  - General approach when closed-form solutions unavailable

Generative vs. Discriminative Classifiers

Training classifiers involves estimating f: X → Y, or P(Y|X)

Generative classifiers (e.g., Naïve Bayes)

- Assume some functional form for P(X|Y), P(X)
- Estimate parameters of P(X|Y), P(X) directly from training data
- Use Bayes rule to calculate P(Y|X= x_i)

Discriminative classifiers (e.g., Logistic regression)

- Assume some functional form for P(Y|X)
- Estimate parameters of P(Y|X) directly from training data

Use Naïve Bayes or Logisitic Regression?

Consider

- Restrictiveness of modeling assumptions

- Rate of convergence (in amount of training data) toward asymptotic hypothesis

Naïve Bayes vs Logistic Regression

Consider Y boolean, X_i continuous, X=<X_1 ... X_n>

Number of parameters to estimate:

- NB:

- LR:

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

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

Naïve Bayes vs Logistic Regression

Consider Y boolean, X_i continuous, $X=<X_1 \ldots X_n>$

Number of parameters:

- NB: 4n +1
- LR: n+1

Estimation method:

- NB parameter estimates are uncoupled
- LR parameter estimates are coupled

G.Naïve Bayes vs. Logistic Regression

[Ng & Jordan, 2002]

What if we have only finite training data?

They converge at different rates to their asymptotic (∞ data) error

Let $\epsilon_{A,n}$ refer to expected error of learning algorithm A after *n* training examples

Let *d* be the number of features: <X_1 … X_d>

$$\epsilon_{LR,n} \leq \epsilon_{LR,\infty} + O\left(\sqrt{\frac{d}{n}}\right)$$

$$\epsilon_{GNB,n} \leq \epsilon_{GNB,\infty} + O\left(\sqrt{\frac{\log d}{n}}\right)$$

So, GNB requires *n* = *O(log d)* to converge, but LR requires *n* = *O(d)*

Some experiments from UCI data sets

[Ng & Jordan, 2002]

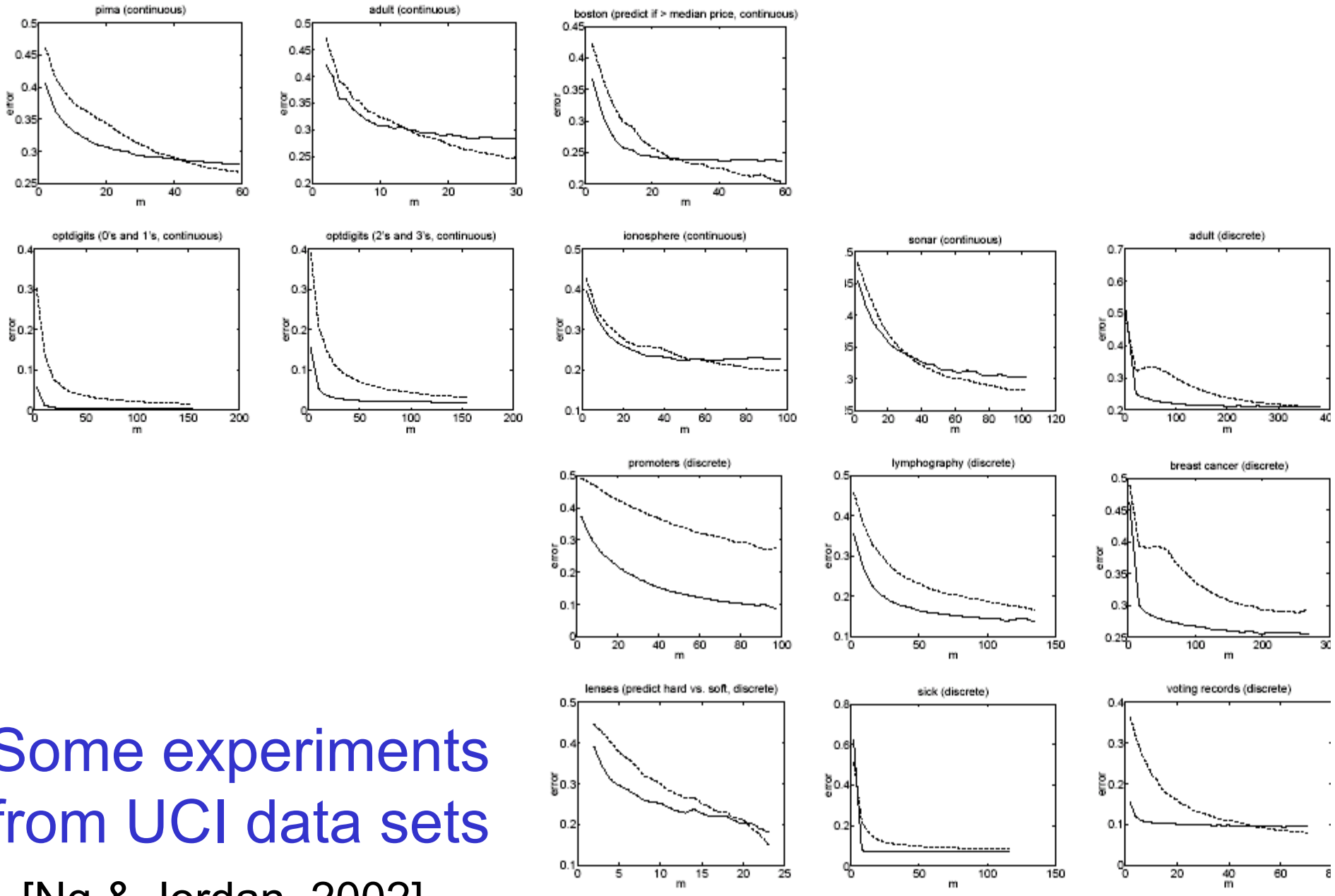


Figure 1: Results of 15 experiments on datasets from the UCI Machine Learnin repository. Plots are of generalization error vs. m (averaged over 1000 rando train/test splits). Dashed line is logistic regression; solid line is naive Bayes.

Regression

So far, we've been interested in learning P(Y|X) where Y has discrete values (called 'classification')

What if Y is continuous? (called 'regression')

- predict weight from gender, height, age, …
- predict Google stock price today from Google, Yahoo, MSFT prices yesterday
- predict each pixel intensity in robot's next camera image, from current image and current action

Regression

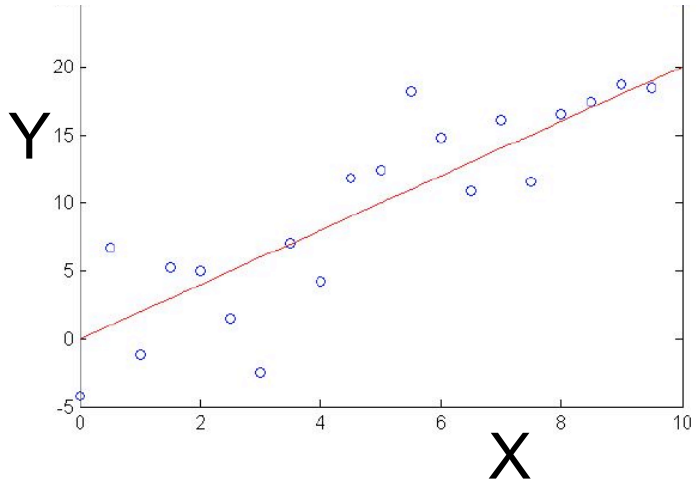
Wish to learn f:X→Y, where Y is real, given $\{<x^1,y^1>\ldots<x^n,y^n>\}$

Approach:

1. choose some parameterized form for P(Y|X; θ)
   (θ is the vector of parameters)

2. derive learning algorithm as MCLE or MAP estimate for θ

1. Choose parameterized form for P(Y|X; θ)



Assume Y is some deterministic f(X), plus random noise

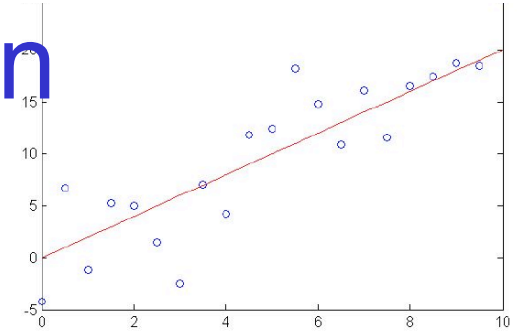
$$y = f(x) + \epsilon \qquad \text{where} \quad \epsilon \sim N(0, \sigma)$$

Therefore Y is a random variable that follows the distribution

$$p(y|x) = N(f(x), \sigma)$$

and the expected value of y for any given x is f(x)

Consider Linear Regression



$$p(y|x) = N(f(x), \sigma)$$

E.g., assume f(x) is linear function of x

$$p(y|x) = N(w_0 + w_1 x, \sigma)$$

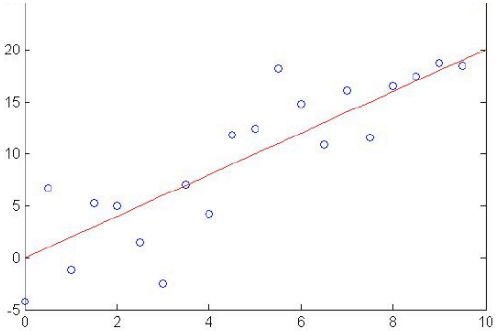
$$E[y|x] = w_0 + w_1 x$$

Notation: to make our parameters explicit, let's write

$$W = < w_0, w_1 >$$

$$p(y|x; W) = N(w_0 + w_1 x, \sigma)$$

Training Linear Regression



$$p(y|x;W) = N(w_0 + w_1 x, \sigma)$$

How can we learn W from the training data?

Learn Maximum Conditional Likelihood Estimate!

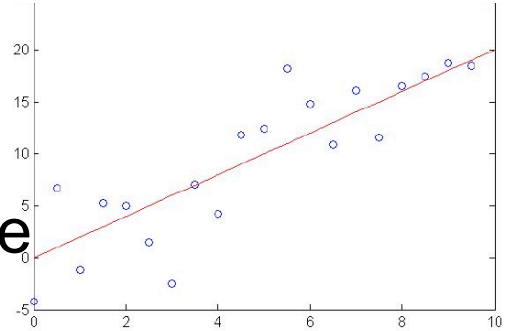
$$W_{MCLE} = \arg\max_W \prod_l p(y^l|x^l, W)$$

$$W_{MCLE} = \arg\max_W \sum_l \ln p(y^l|x^l, W)$$

where

$$p(y|x;W) = \frac{1}{\sqrt{2\pi\sigma^2}} \; e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$

Training Linear Regression

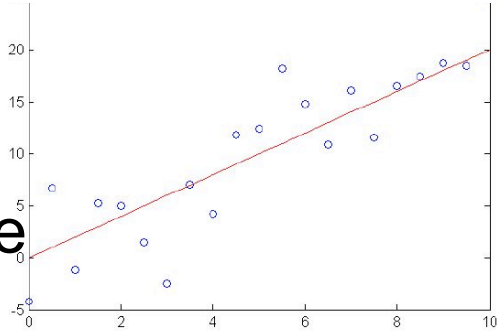


Learn Maximum Conditional Likelihood Estimate

$$W_{MCLE} = \arg\max_W \sum_l \ln p(y^l | x^l, W)$$

where $$p(y|x;W) = \frac{1}{\sqrt{2\pi\sigma^2}} \quad e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$

Training Linear Regression



Learn Maximum Conditional Likelihood Estimate

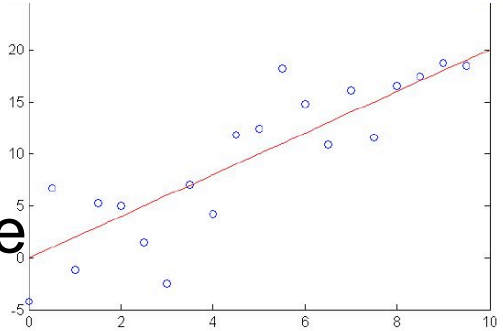
$$W_{MCLE} = \arg\max_W \sum_l \ln p(y^l | x^l, W)$$

where $$p(y|x;W) = \frac{1}{\sqrt{2\pi\sigma^2}} \; e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$

so: $$W_{MCLE} = \arg\min_W \sum_l (y^l - f(x^l;W))^2$$

l th training example

Training Linear Regression



Learn Maximum Conditional Likelihood Estimate

$$W_{MCLE} = \arg\min_W \sum_l (y^l - f(x^l; W))^2$$

Can we derive gradient descent rule for training?

$$\frac{\partial \sum_l (y^l - f(x^l; W))^2}{\partial w_i} = \sum_l 2(y^l - f(x^l; W)) \frac{\partial (y^l - f(x^l; W))}{\partial w_i}$$

$$= \sum_l -2(y^l - f(x^l; W)) \frac{\partial f(x^l; W)}{\partial w_i}$$

How about MAP instead of MLE estimate?

If we assume zero-mean Gaussian prior on each w_i

$$P(W = \langle w_1 \ldots w_n \rangle | \mu = 0, \sigma) = \frac{1}{(\sqrt{2\pi\sigma^2})^n} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n} w_i^2\right)$$

$$\ln P(W) \propto \sum_{i=1}^{n} w_i^2$$

Then:

$$W = \arg\max_W \ln P(W|\mu = 0, \sigma) + \sum_l \ln P(y^l|x^l; W)$$

$$= \arg\max_W -c\sum_i w_i^2 + \sum_l \ln P(y^l|x^l; W)$$

$$W_{MAP} = \arg\min_W c\sum_i w_i^2 + \sum_l (y^l - f(x^l; W))^2$$

A Second Common Prior: Laplace

If we assume zero-mean Laplace prior on each w_i

$$P(W = \langle w_1 \ldots w_n \rangle) | \mu = 0, b) = \frac{1}{(2b)^n} \exp\left(-\frac{1}{b}\sum_{i=1}^{n} |w_i|\right)$$

$$\ln P(W) \propto \sum_{i=1}^{n} |w_i|$$

Then:

$$W_{MAP} = \arg\max_W \ln P(W|\mu = 0, b) + \sum_l \ln P(y^l|x^l; W)$$

$$= \arg\max_W -c\sum_i |w_i| + \sum_l \ln P(y^l|x^l; W)$$

$$W_{MAP} = \arg\min_W c\sum_i |w_i| + \sum_l (y^l - f(x^l; W))^2$$

Zero mean Gaussian prior:

$$P(W = \langle w_1 \dots w_n \rangle | \mu = 0, \sigma) = \frac{1}{(\sqrt{2\pi\sigma^2})^n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} w_i^2\right)$$

$$\ln P(W) \propto \sum_{i=1}^{n} w_i^2$$

Zero mean Laplace prior:

$$P(W = \langle w_1 \dots w_n \rangle) | \mu = 0, b) = \frac{1}{(2b)^n} \exp\left(-\frac{1}{b} \sum_{i=1}^{n} |w_i|\right)$$

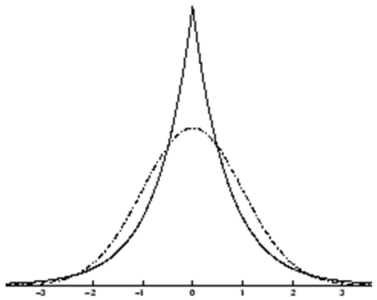
$$\ln P(W) \propto \sum_{i=1}^{n} |w_i|$$

Gaussian P(W)

→ L2 regularization

→ smaller magnitude w's

$$\ln P(W) \propto \sum_i w_i^2$$

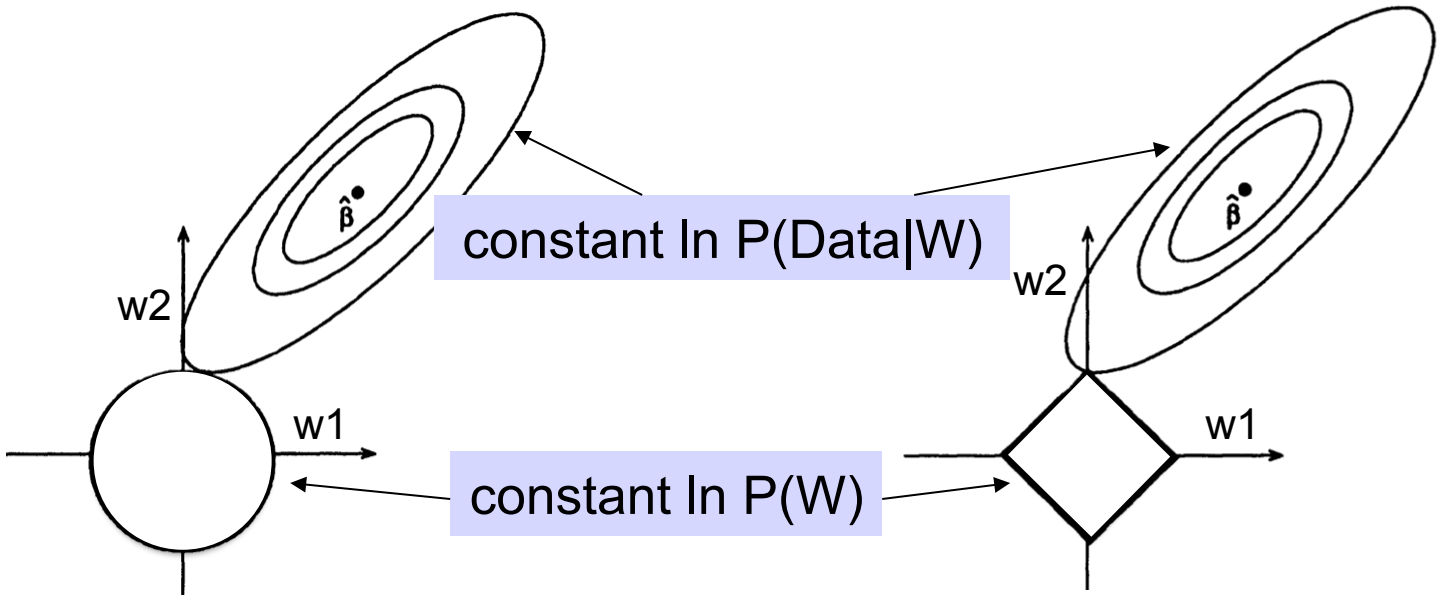


Laplace P(W)

→ L1 regularization

→ fewer non-zero w's

$$\ln P(W) \propto \sum_i |w_i|$$



Regression – What you should know

Under common assumption that $p(y|x;W) = N(f(x;W), \sigma I)$

1. MLE corresponds to minimizing sum of squared prediction errors (SSE)

2. MAP estimate minimizes SSE plus sum of squared weights

3. Again, learning is an optimization problem once we choose our objective function
   - maximize data conditional likelihood P(Y | X; W)
   - maximize posterior probability of W, P(W | Y, X) under assumed P(W)

4. Again, we can use gradient descent as a general learning algorithm
   - as long as our objective function is differentiable wrt W
   - though we might learn local optimum solution

5. Nothing we said here required that f(x) be linear in x

Artificial Neural Networks

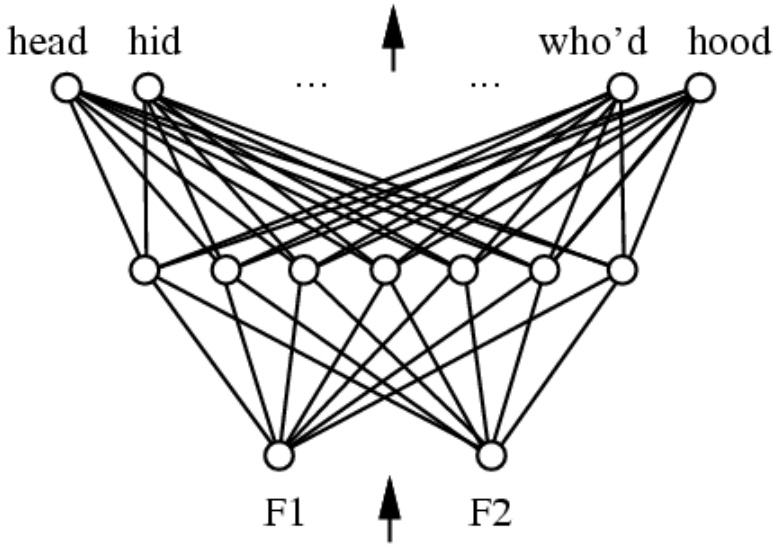
Artificial Neural Networks to learn f: X → Y

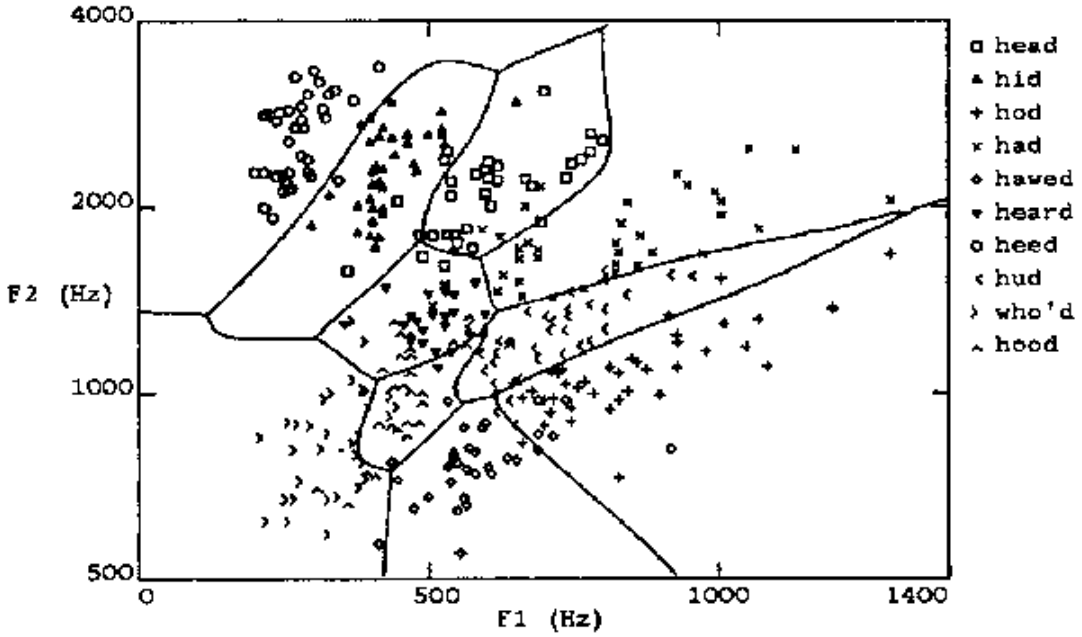
- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

- Represent f by ***network*** of computational units
- Most common type of unit is a logistic function

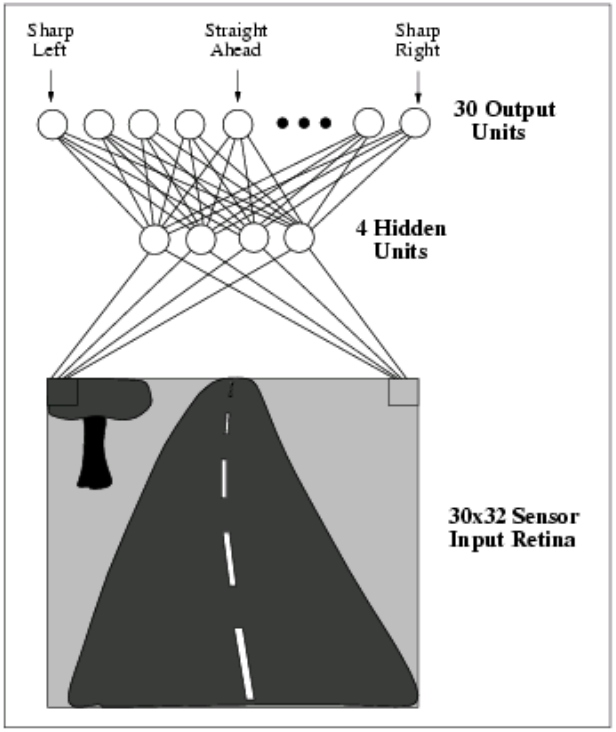
$$unit\ output = \frac{1}{1 + exp(w_0 + \sum_i w_i x_i)}$$

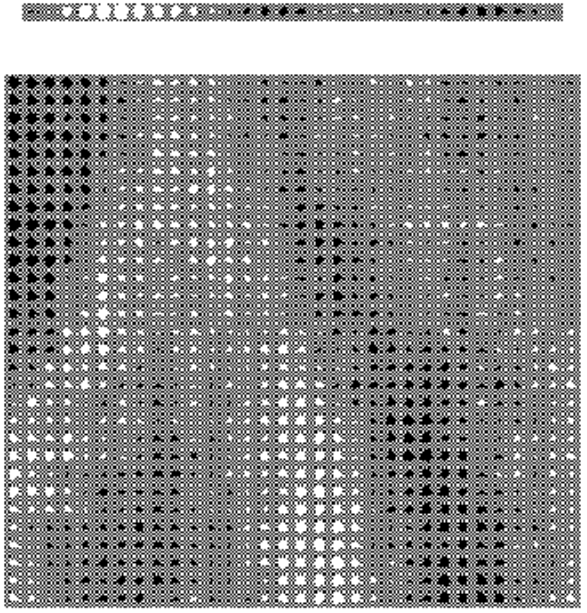
Multilayer Networks of Sigmoid Units





ALVINN
[Pomerleau 1993]





Connectionist Models

Consider humans:

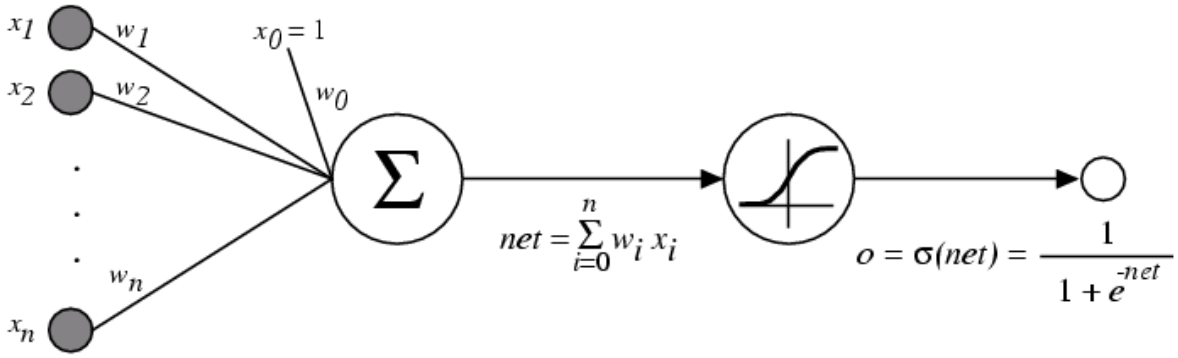
- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

$\rightarrow$ 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

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 decent rules to train

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

M(C)LE Training for Neural Networks

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

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

deterministic

- Let's maximize the conditional data likelihood

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

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

Learned neural network

MAP Training for Neural Networks

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

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

deterministic

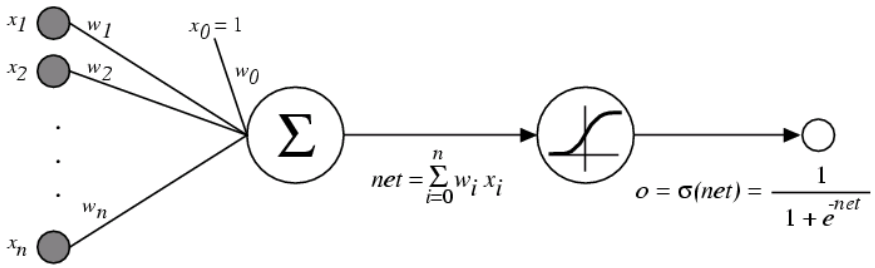
Gaussian $P(W) = N(0,\sigma I)$

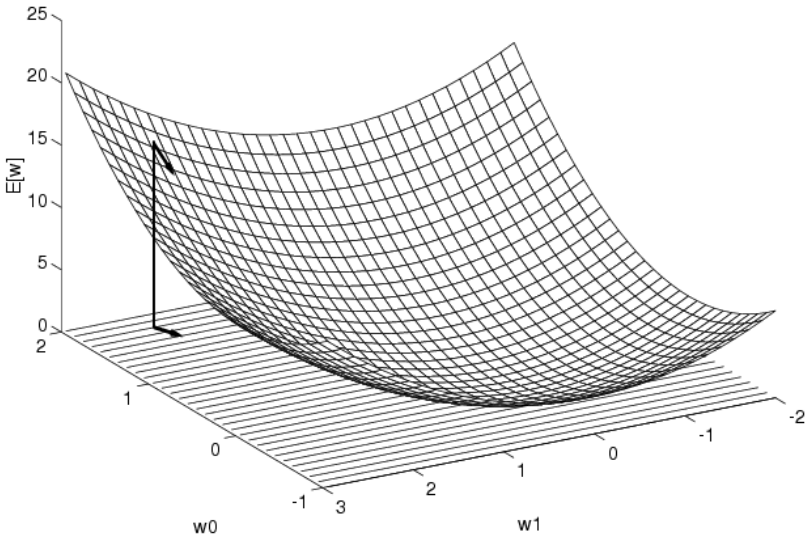
$$W \leftarrow \arg\max_W \ \ln \ P(W) \prod_l P(Y^l | X^l, W)$$

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

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

Gradient Descent





Gradient

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

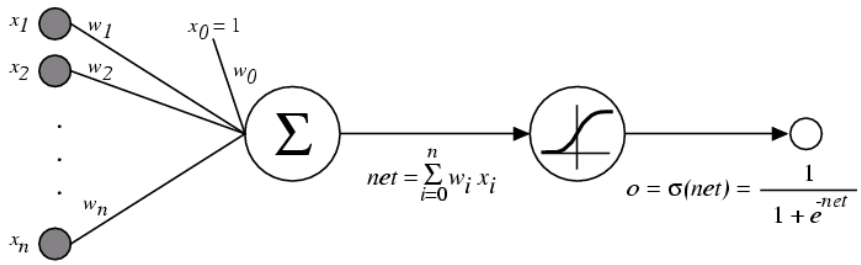
Training rule:

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

i.e.,

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

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:

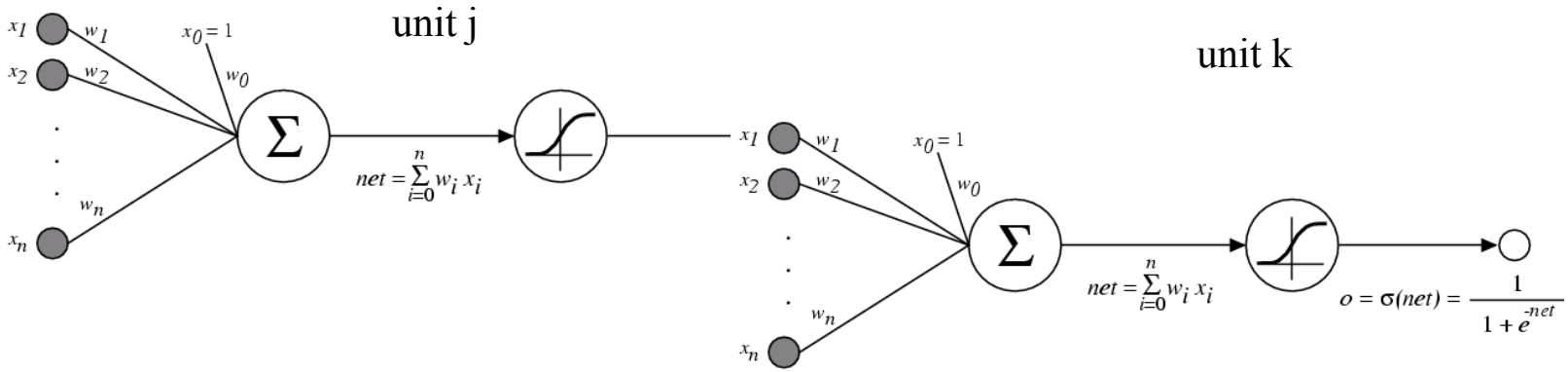
$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

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

So:

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

x_d = input
t_d = target output
o_d = observed unit output
w_i = weight I
d = training example d



x_d = input
t_d = target output
o_d = observed unit output
w_i = weight i
d = training example d

Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

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

Incremental mode Gradient Descent:
Do until satisfied

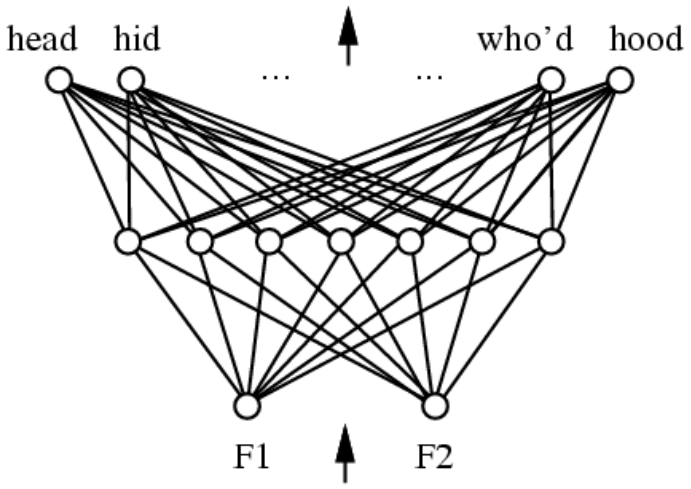
- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

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

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

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η 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

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

  3. For each hidden unit h

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

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

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

where

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

x_d = input
t_d = target output
o_d = observed unit output
w_{ij} = wt from i to j

More on Backpropagation

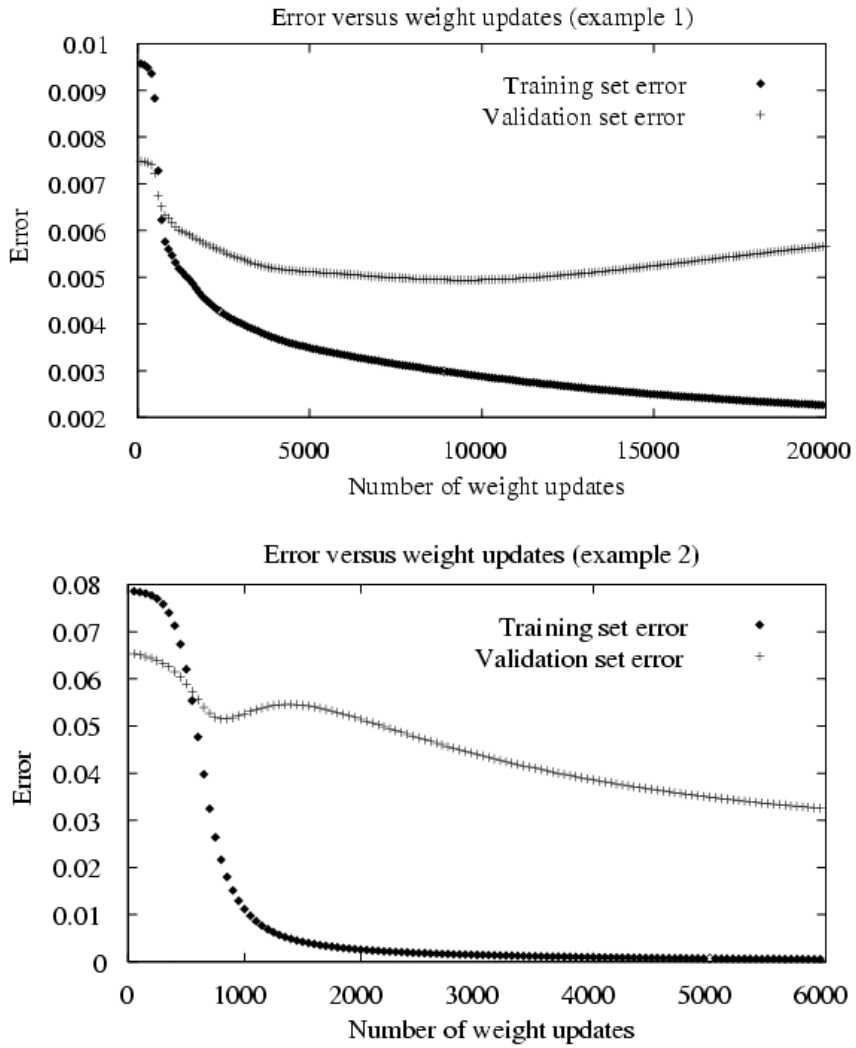
- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

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

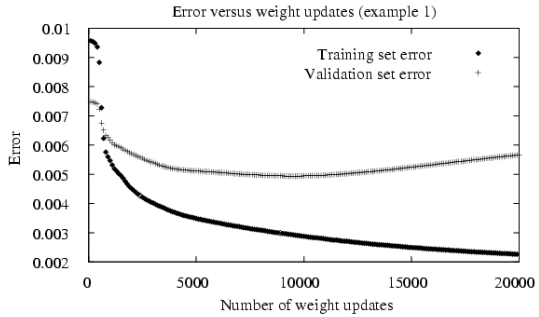
- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

+ drop out
+ …

Overfitting in ANNs



Dealing with Overfitting



Our learning algorithm involves a parameter

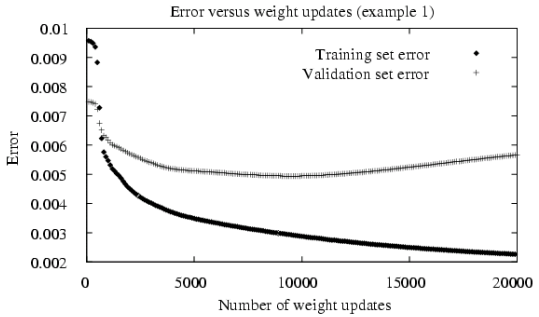
n=number of gradient descent iterations

How do we choose n to optimize future error?

(note: similar issue for logistic regression, decision trees, …)

e.g. the n that minimizes error rate of neural net over future data

Dealing with Overfitting



Our learning algorithm involves a parameter

n=number of gradient descent iterations

How do we choose n to optimize future error?

- Separate available data into <u>training</u> and <u>validation</u> set
- Use <u>training</u> to perform gradient descent
- n ← number of iterations that optimizes <u>validation</u> set error

→ *gives unbiased estimate of optimal n*

(but a <u>biased</u> estimate of true error)

K-Fold Cross Validation

Idea: train multiple times, leaving out a disjoint subset of data each time for test. Average the test set accuracies.

Partition data into K disjoint subsets

For k=1 to K

testData = kth subset

h ← classifier trained* on all data except for testData

accuracy(k) = accuracy of h on testData

end

FinalAccuracy = mean of the K recorded testset accuracies

* might withhold some of this to choose number of gradient decent steps

Leave-One-Out Cross Validation

This is just k-fold cross validation leaving out one example each iteration

---

Partition data into K disjoint subsets, *each containing one example*

For k=1 to K

testData = kth subset

h ← classifier trained* on all data except for testData

accuracy(k) = accuracy of h on testData

end

FinalAccuracy = mean of the K recorded testset accuracies

* might withhold some of this to choose number of gradient decent steps

Expressive Capabilities of ANNs

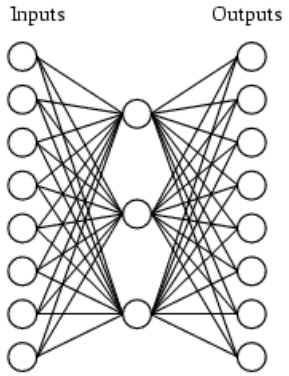
Boolean functions:

- 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].

Learning Hidden Layer Representations



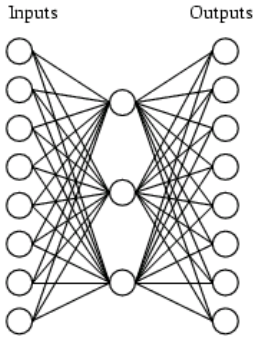
A target function:

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??

Learning Hidden Layer Representations

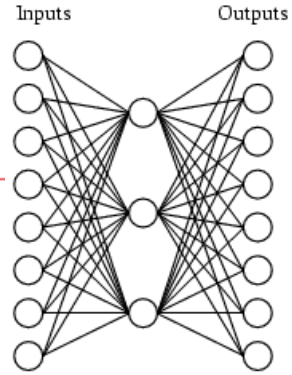
A network:

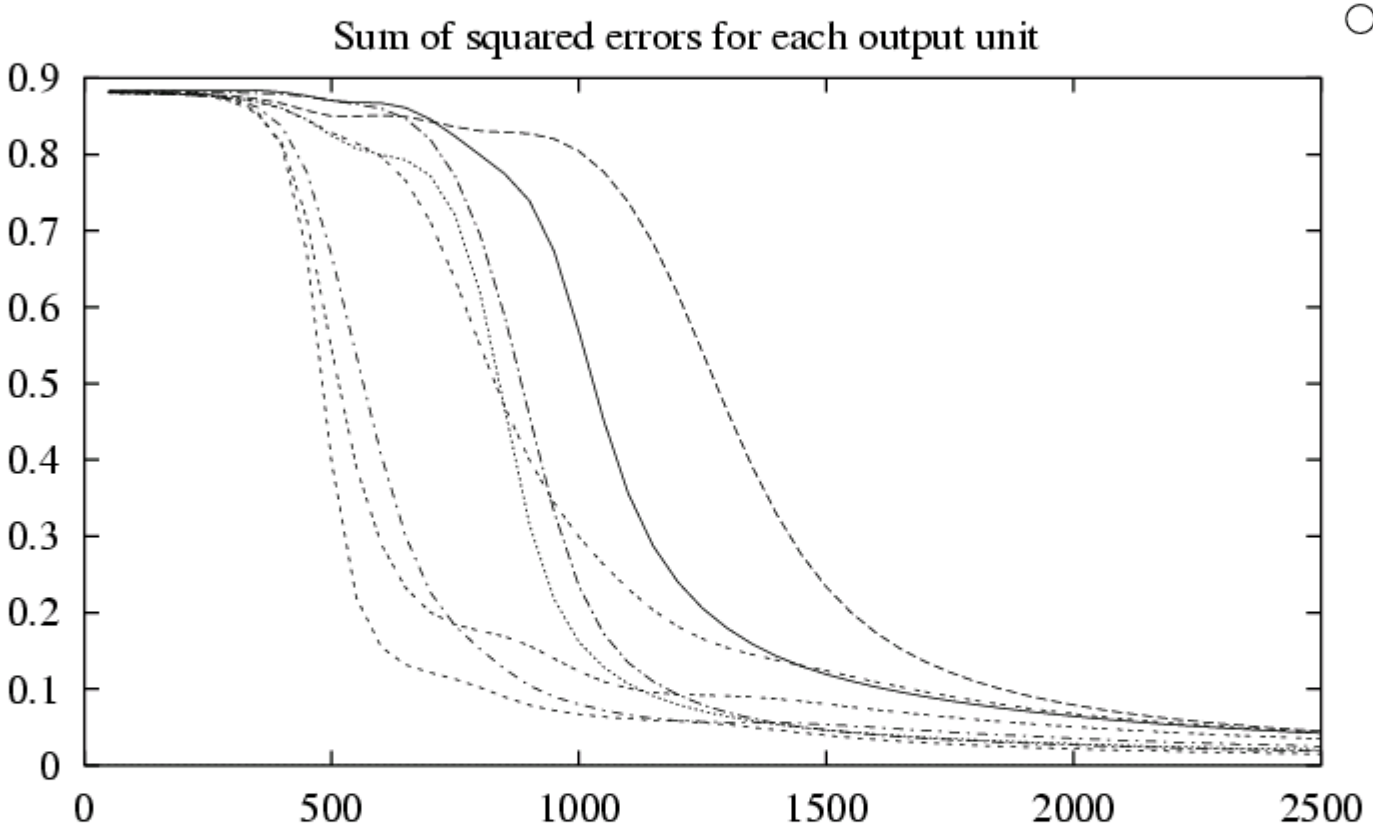


Learned hidden layer representation:

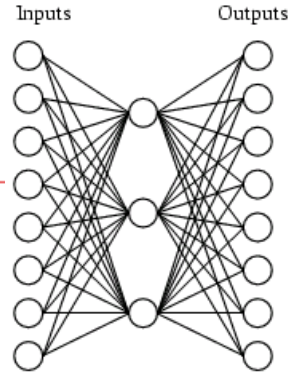
Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$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

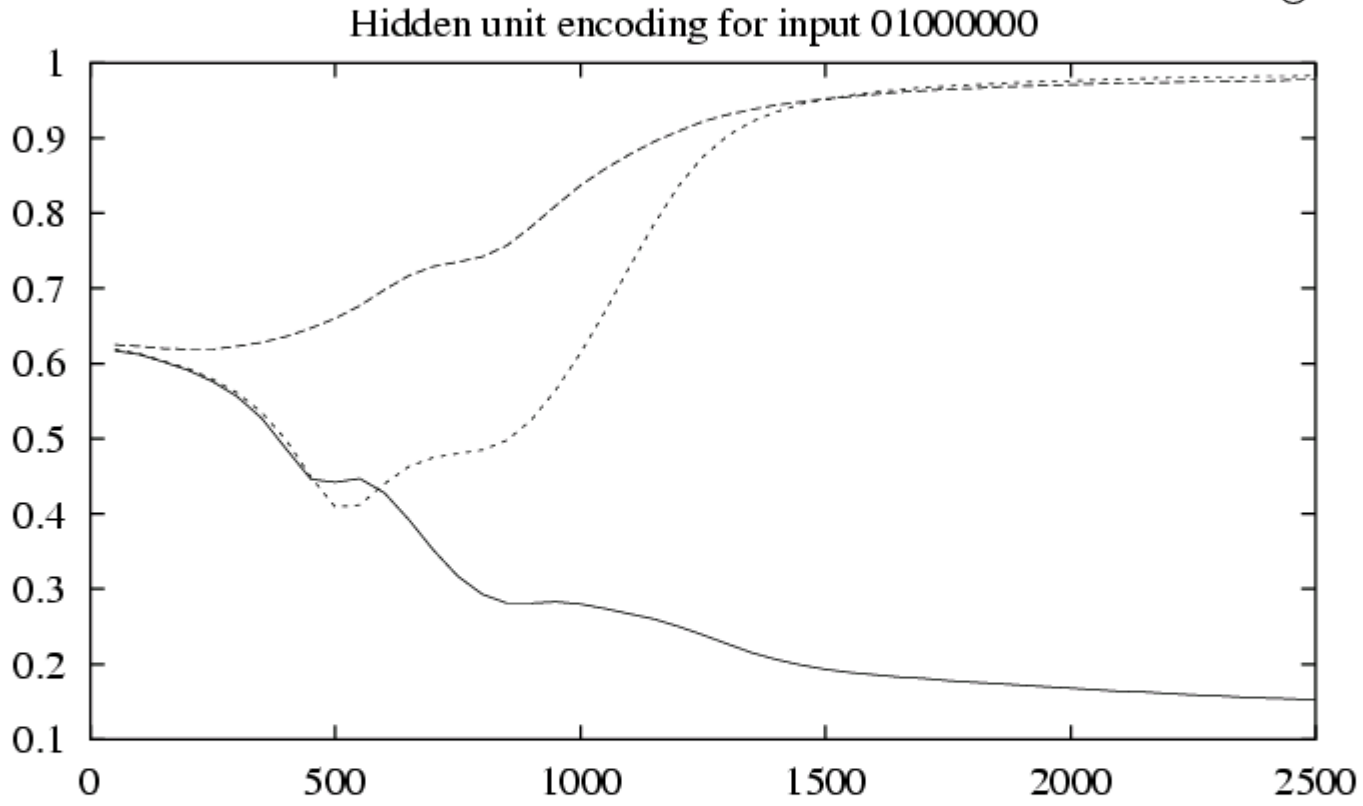
Training


Inputs
Outputs

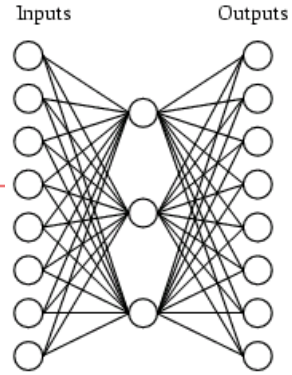

Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500

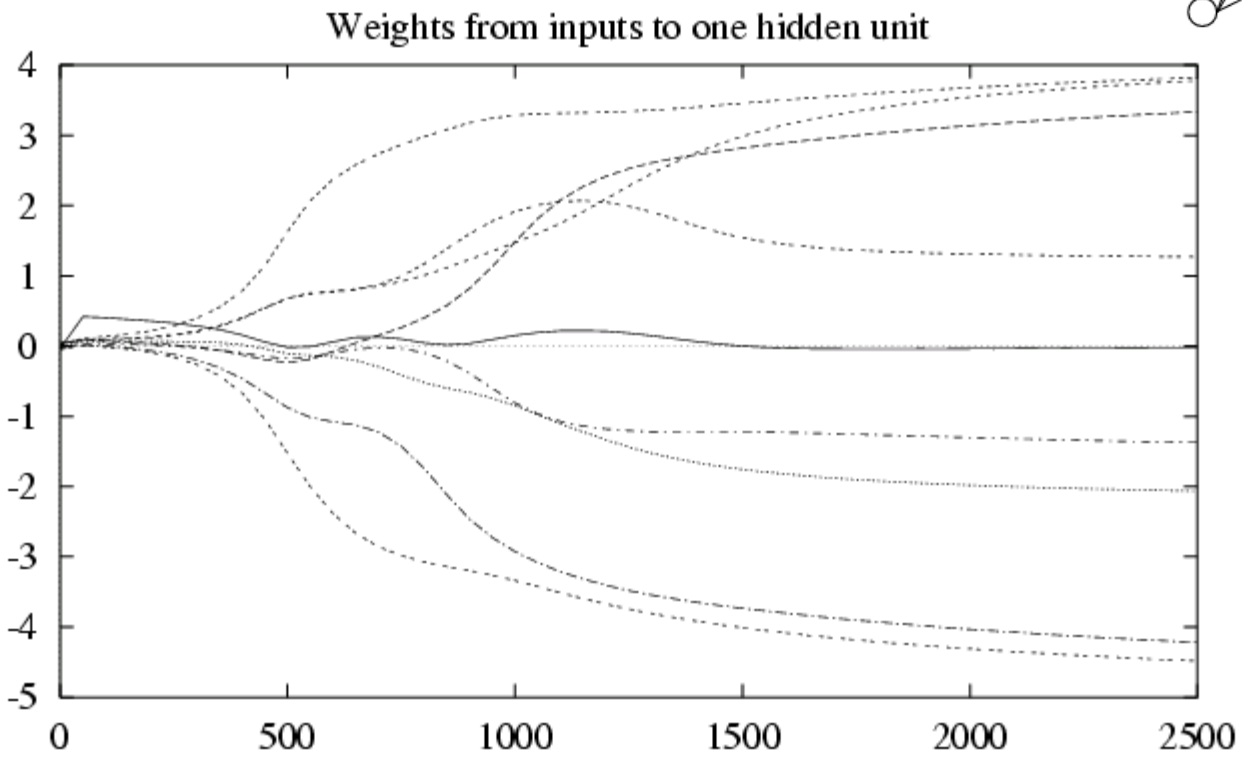
Training


Inputs
Outputs

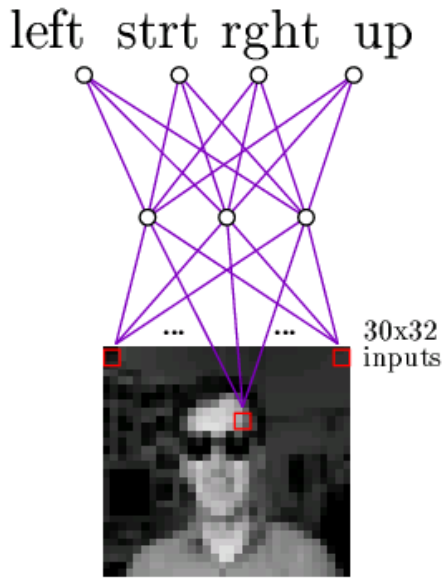

Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500

Training


Inputs
Outputs


Weights from inputs to one hidden unit
4
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

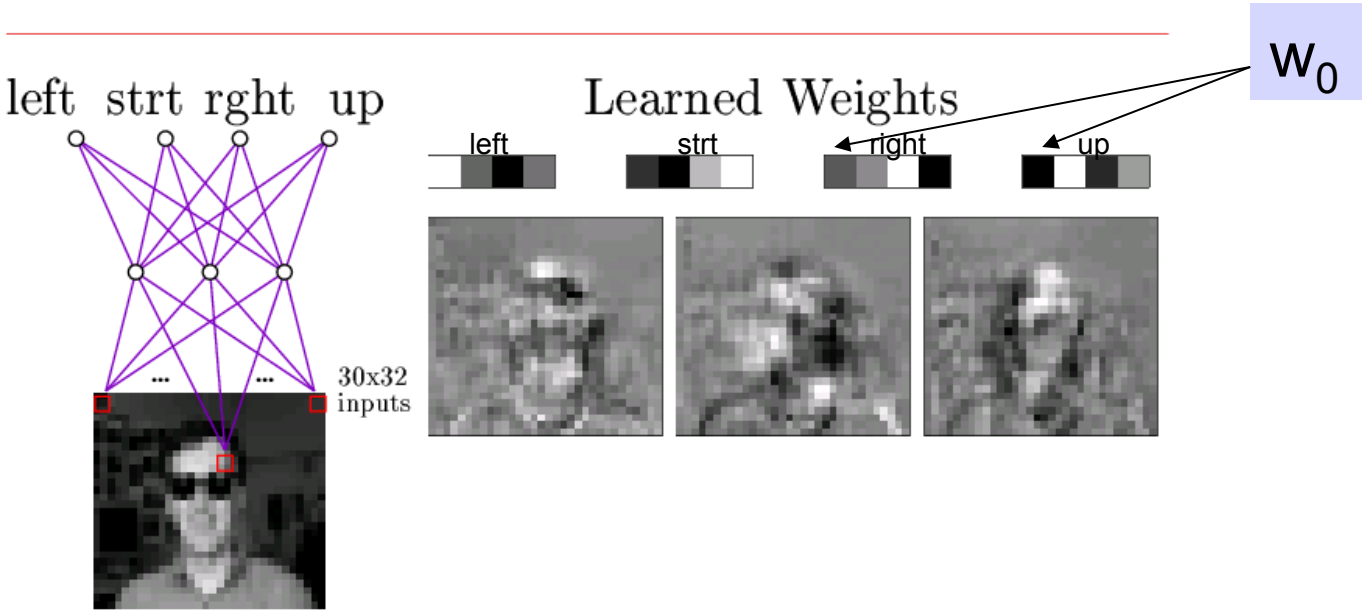
Neural Nets for Face Recognition



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