Artificial Neural Networks

[Read Ch. 4]
[Recommended exercises 4.1, 4.2, 4.5, 4.9, 4.11]

- Threshold units

- Gradient descent

- Multilayer networks

- Backpropagation

- Hidden layer representations

- Example: Face Recognition

- Advanced topics

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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
- Emphasis on tuning weights automatically

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

When to Consider Neural Networks

- 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

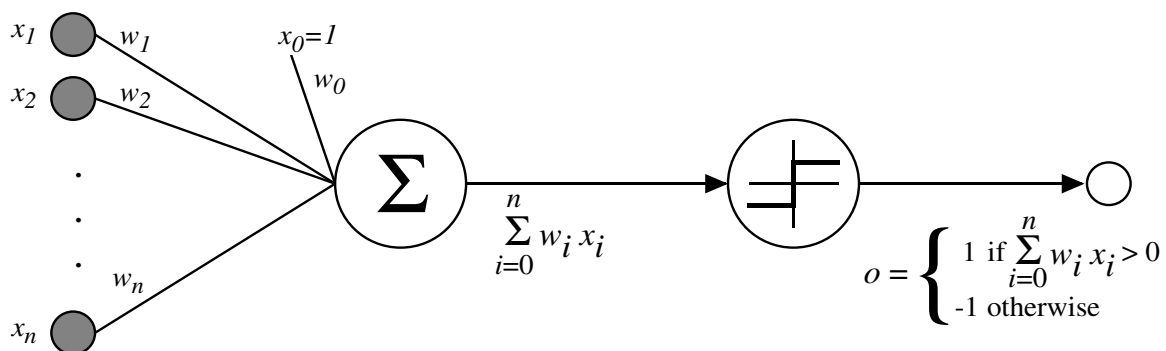- Human readability of result is unimportant

Examples:

- Speech phoneme recognition [Waibel]

- Image classification [Kanade, Baluja, Rowley]

- Financial prediction

ALVINN drives 70 mph on highways

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997
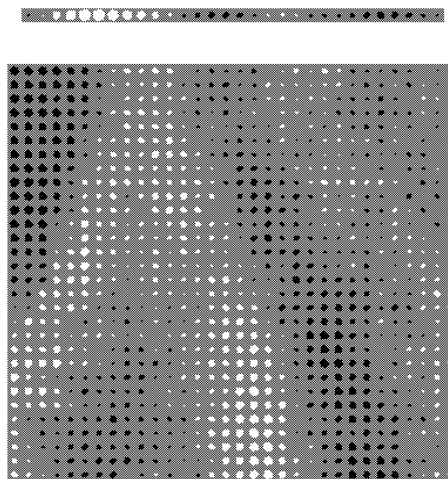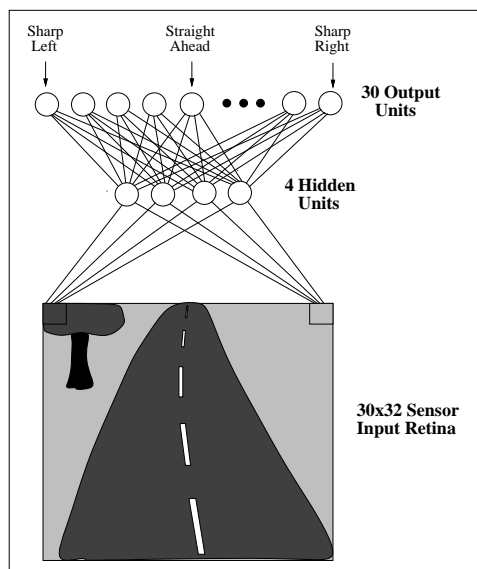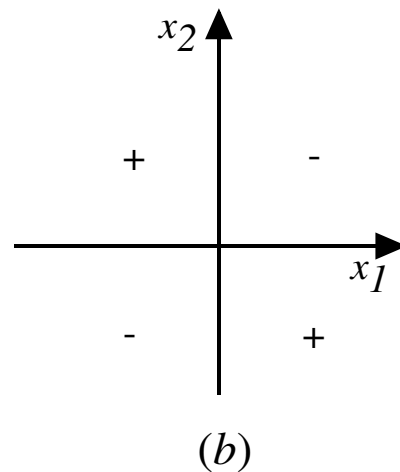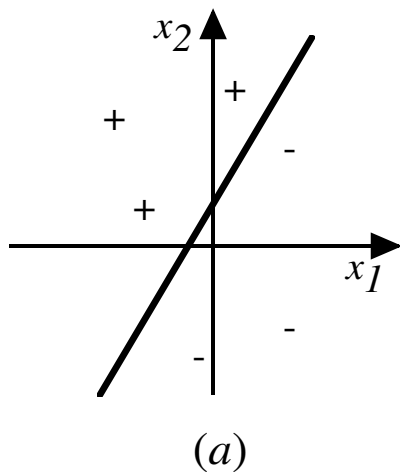
Perceptron

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

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

(a)   (b)

Represents some useful functions

- What weights represent
  $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable

- Therefore, we'll want networks of these...

     lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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*

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Perceptron training rule

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Gradient Descent

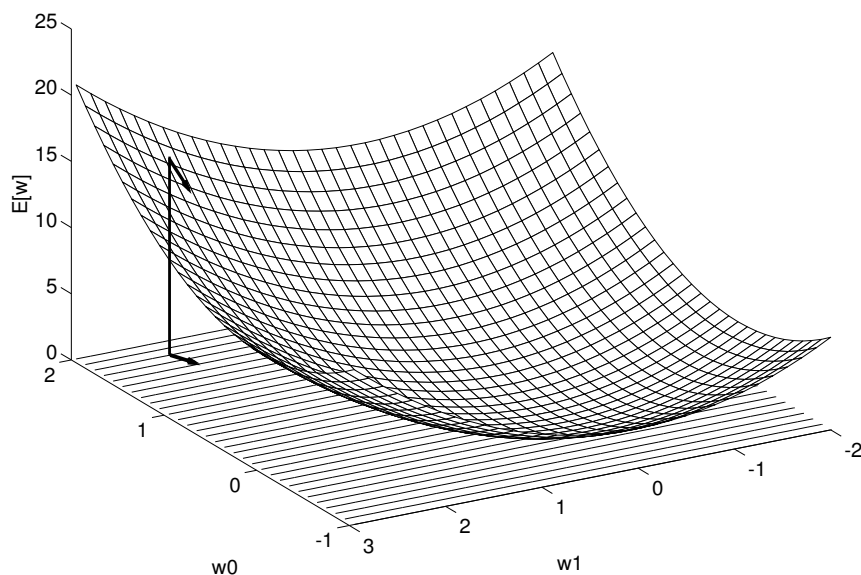To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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.,

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Gradient Descent

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d})$$

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Gradient Descent

GRADIENT-DESCENT($training\_examples, \eta$)

> *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 value. η is the learning rate (e.g., .05).*

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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:
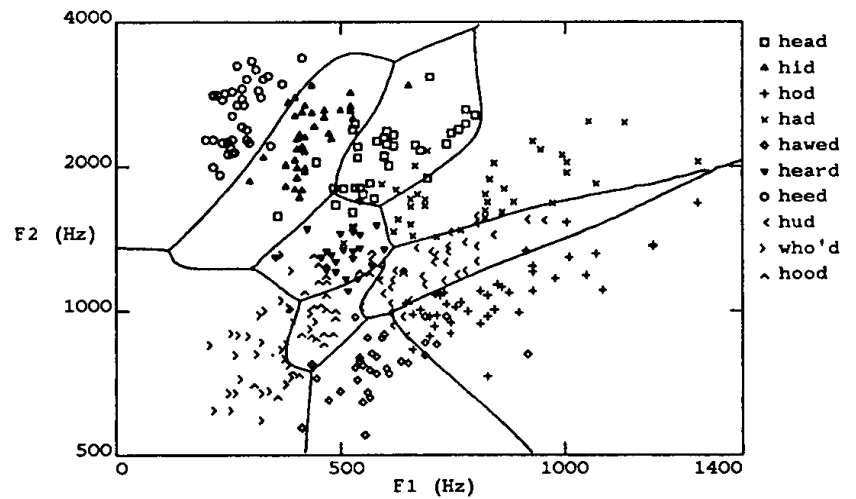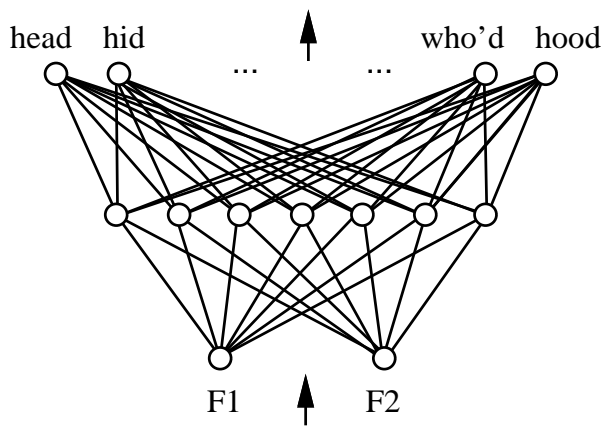Do until satisfied

- For each training example d in D

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

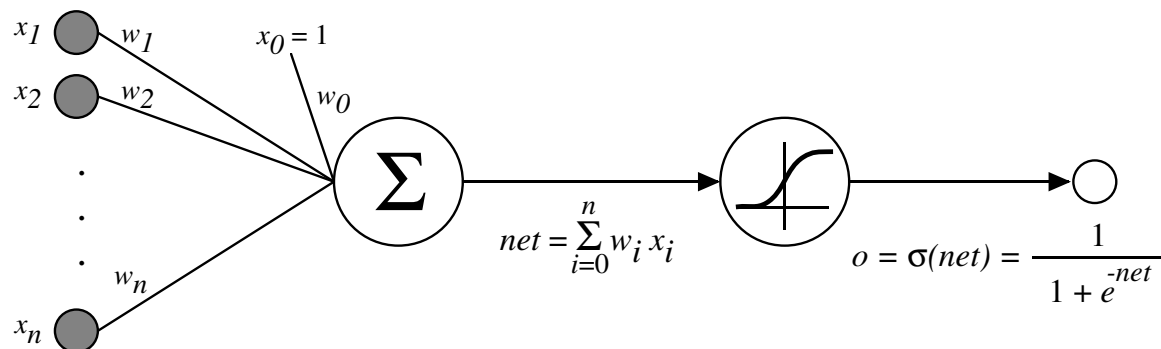$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

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

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

Multilayer Networks of Sigmoid Units

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Error Gradient for a Sigmoid Unit

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

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:

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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

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

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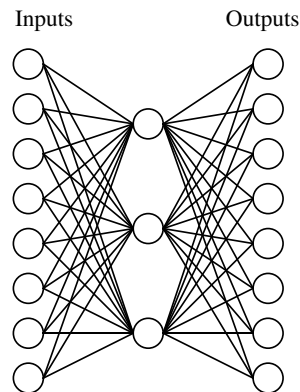
  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

  where

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

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!

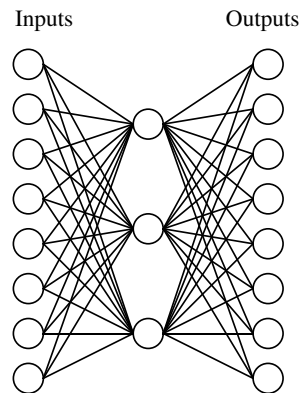- Using network after training is very fast

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Learning Hidden Layer Representations

Inputs   Outputs

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??

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Learning Hidden Layer Representations

A network:

Inputs   Outputs

Learned hidden layer representation:

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

Sum of squared errors for each output unit

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Training

Hidden unit encoding for input 01000000

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Training

Weights from inputs to one hidden unit

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997
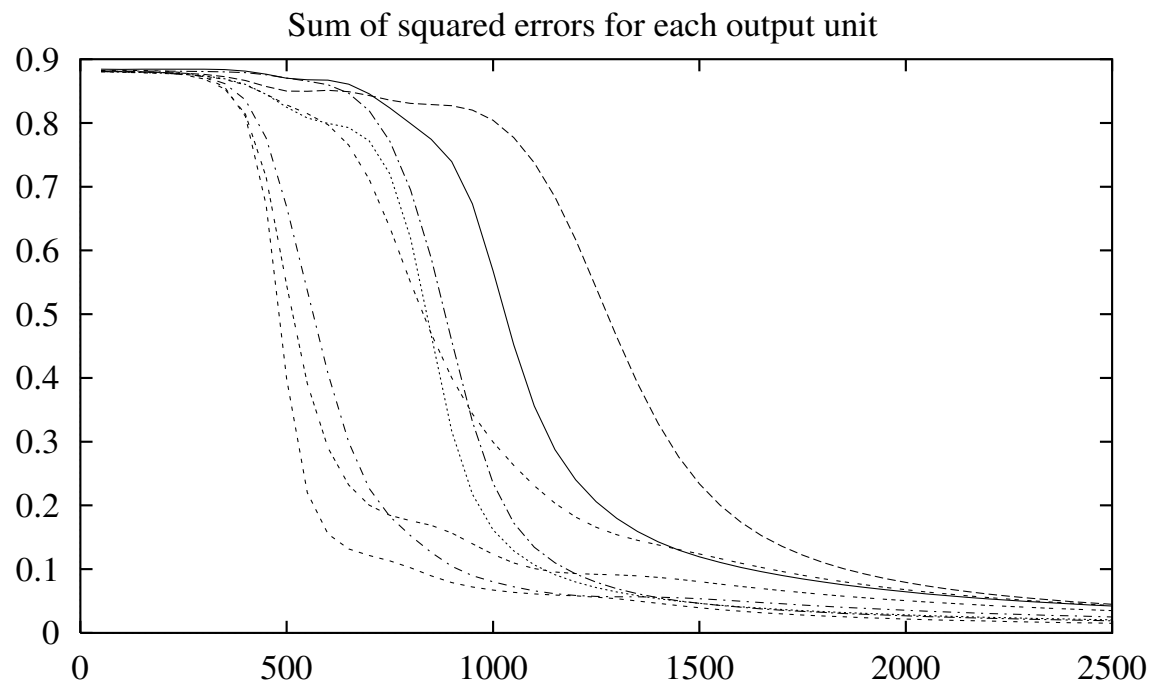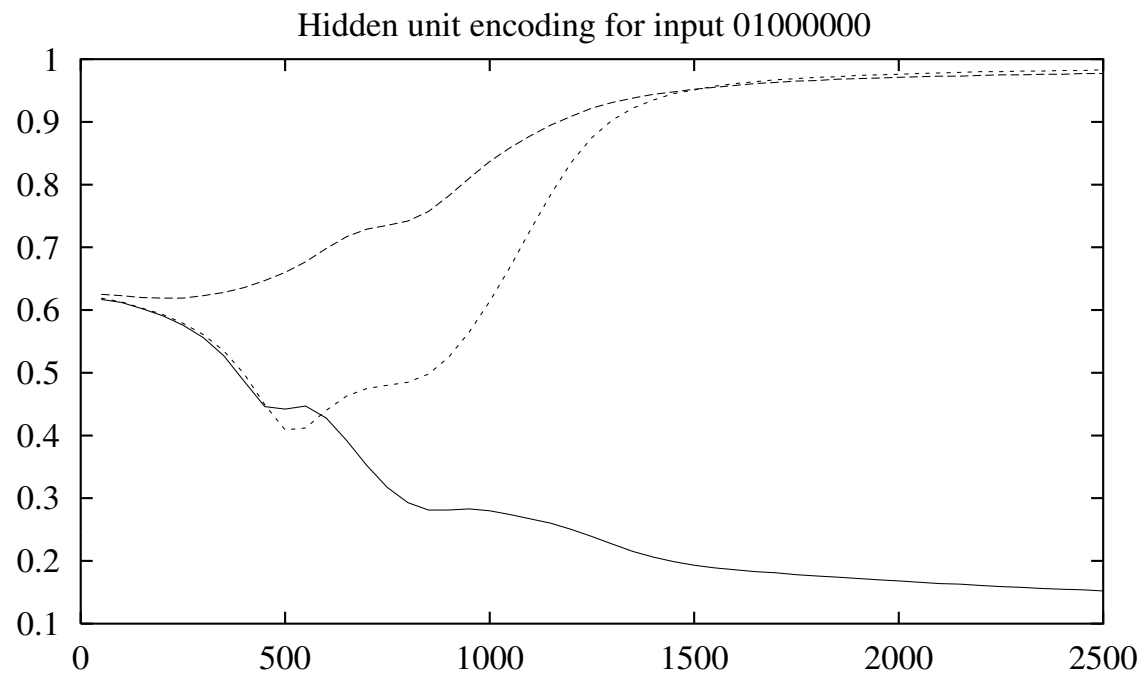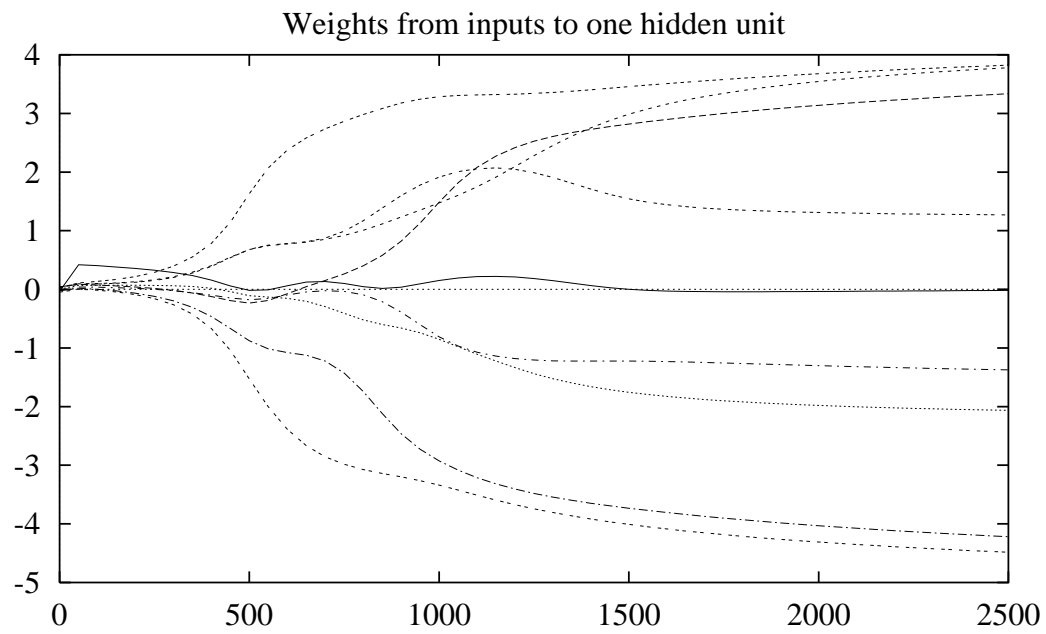
Convergence of Backpropagation

Gradient descent to some local minimum

- Perhaps not global minimum...

- Add momentum

- Stochastic gradient descent

- Train multiple nets with different inital weights

Nature of convergence

- Initialize weights near zero

- Therefore, initial networks near-linear

- Increasingly non-linear functions possible as training progresses

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997
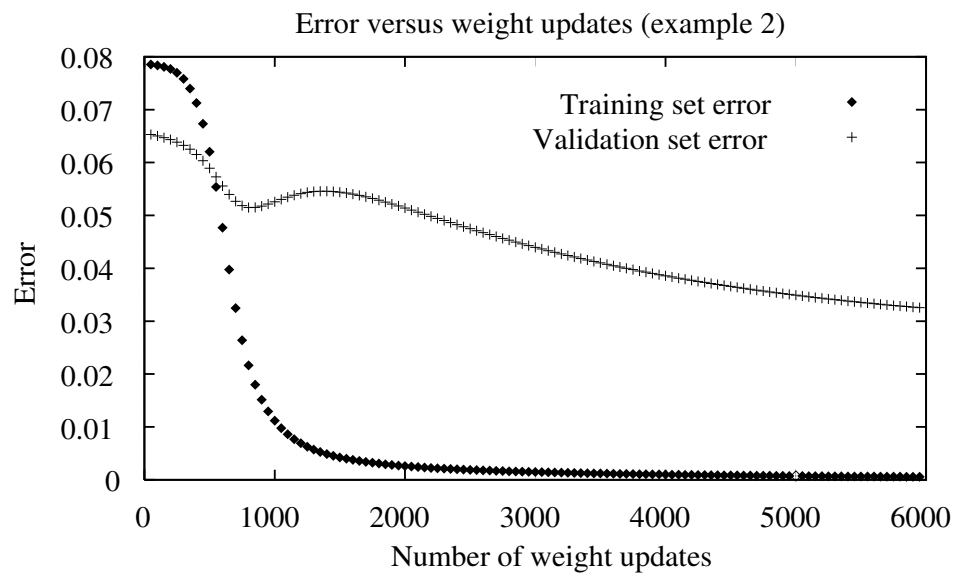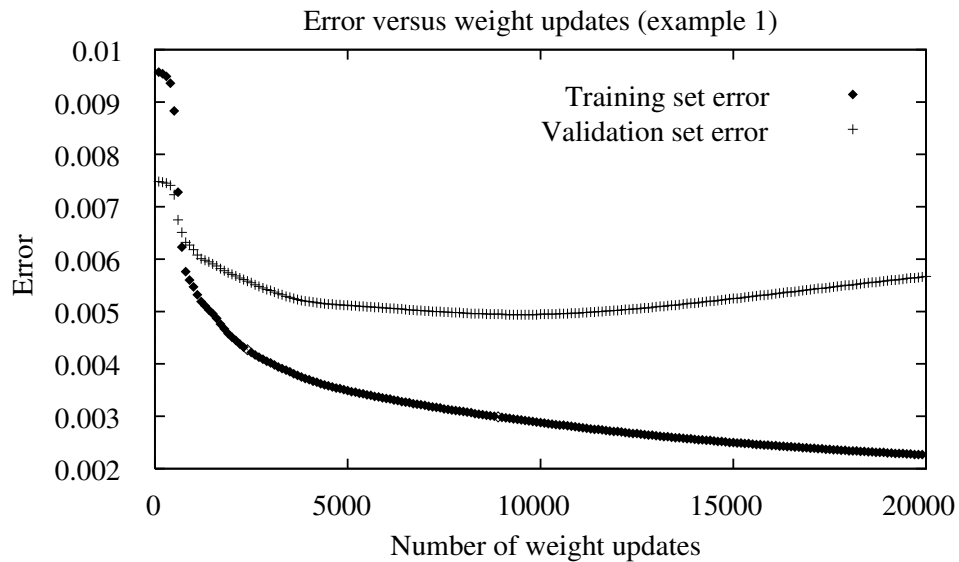
Expressive Capabilities of ANNs

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].

Overfitting in ANNs

Error versus weight updates (example 1)

Error versus weight updates (example 2)

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Neural Nets for Face Recognition

left strt rght up

30x32
inputs
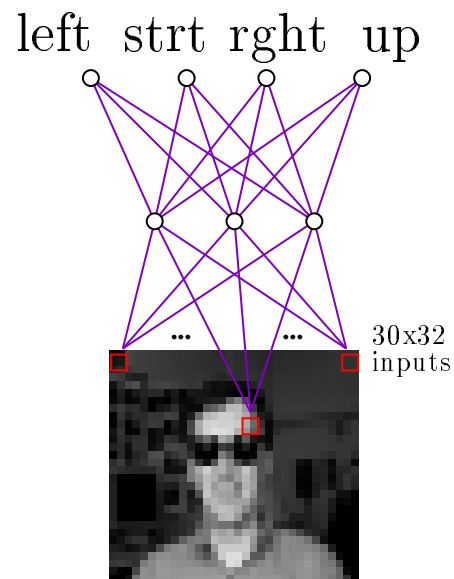
Typical input images

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

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Learned Hidden Unit Weights

left strt rght up   Learned Weights

30x32
inputs

Typical input images

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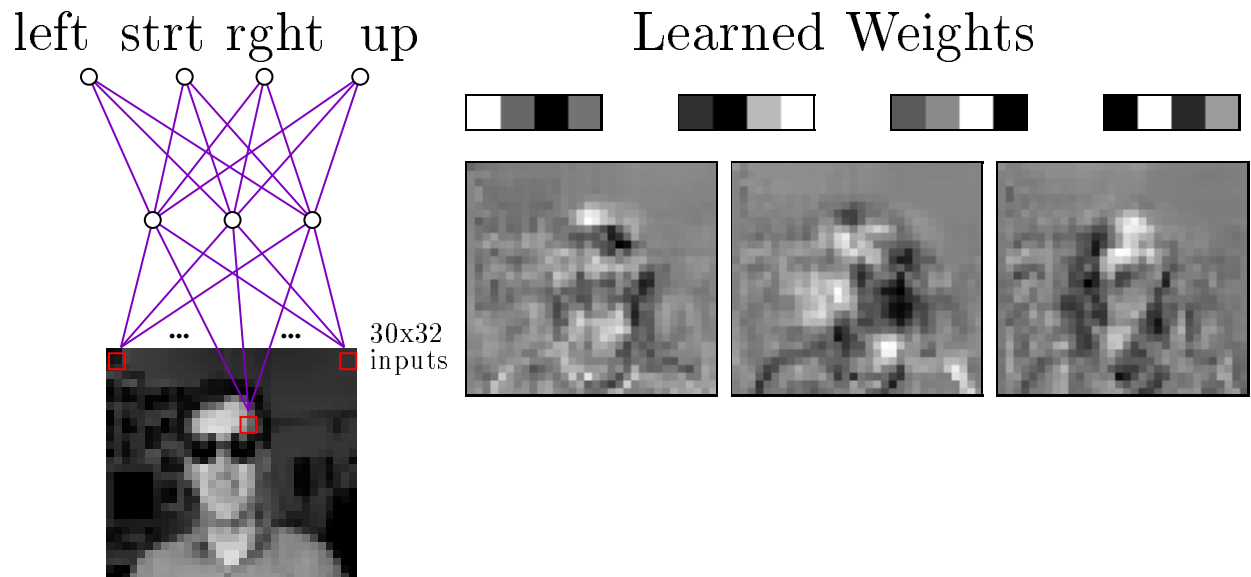
Alternative Error Functions

Penalize large weights:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

Train on target slopes as well as values:
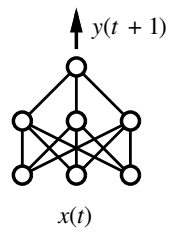
$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in inputs} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

Tie together weights:

- e.g., in phoneme recognition network

     lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Recurrent Networks

(a) Feedforward network
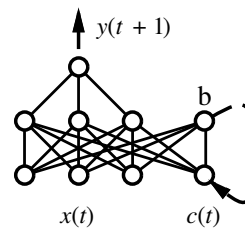
(b) Recurrent network
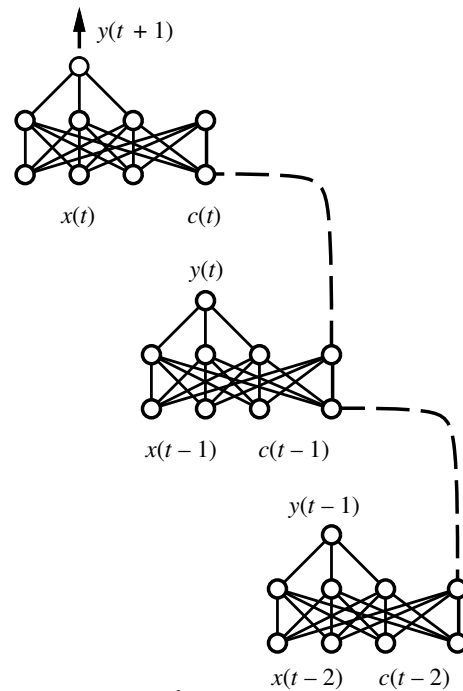
(c) Recurrent network
unfolded in time

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Evaluating Hypotheses

[Read Ch. 5]
[Recommended exercises: 5.2, 5.3, 5.4]

- Sample error, true error

- Confidence intervals for observed hypothesis error

- Estimators

- Binomial distribution, Normal distribution, Central Limit Theorem

- Paired t tests

- Comparing learning methods

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Two Definitions of Error

The **true error** of hypothesis h with respect to target function f and distribution $\mathcal{D}$ is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

The **sample error** of h with respect to target function f and data sample S is the proportion of examples h misclassifies

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))$$

Where $\delta(f(x) \neq h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

How well does $error_S(h)$ estimate $error_{\mathcal{D}}(h)$?

Problems Estimating Error

1. *Bias:* If S is training set, $error_S(h)$ is optimistically biased

$$bias \equiv E[error_S(h)] - error_{\mathcal{D}}(h)$$

For unbiased estimate, h and S must be chosen independently

2. *Variance:* Even with unbiased S, $error_S(h)$ may still *vary* from $error_{\mathcal{D}}(h)$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Example

Hypothesis h misclassifies 12 of the 40 examples in S

$$error_S(h) = \frac{12}{40} = .30$$

What is $error_\mathcal{D}(h)$?

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Estimators

Experiment:

1. choose sample S of size n according to distribution $\mathcal{D}$

2. measure $error_S(h)$

$error_S(h)$ is a random variable (i.e., result of an experiment)

$error_S(h)$ is an unbiased *estimator* for $error_{\mathcal{D}}(h)$

Given observed $error_S(h)$ what can we conclude about $error_{\mathcal{D}}(h)$?

Confidence Intervals

If

- S contains n examples, drawn independently of h and each other

- $n \geq 30$

Then

- With approximately 95% probability, $error_{\mathcal{D}}(h)$ lies in interval

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Confidence Intervals

If

- S contains n examples, drawn independently of h and each other

- $n \geq 30$

Then

- With approximately N% probability, $error_{\mathcal{D}}(h)$ lies in interval

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

where

N%:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

$error_S(h)$ is a Random Variable

Rerun the experiment with different randomly drawn S (of size n)

Probability of observing r misclassified examples:

Binomial distribution for $n = 40$, $p = 0.3$
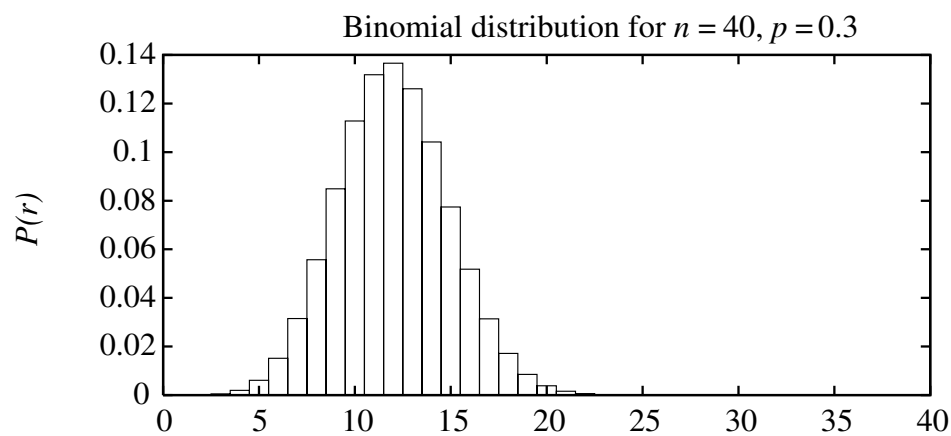
$$P(r) = \frac{n!}{r!(n-r)!}\ error_{\mathcal{D}}(h)^r (1 - error_{\mathcal{D}}(h))^{n-r}$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Binomial Probability Distribution

Binomial distribution for $n = 40$, $p = 0.3$

$P(r)$

$$P(r) = \frac{n!}{r!(n-r)!}\, p^r (1-p)^{n-r}$$
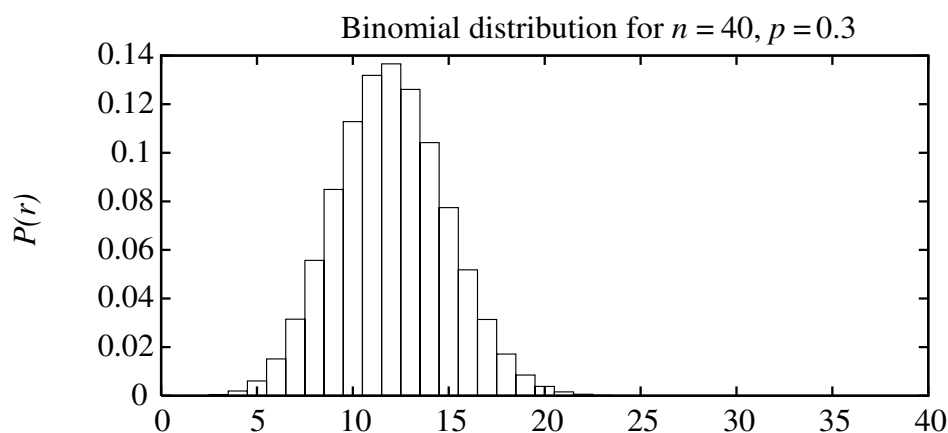
Probability $P(r)$ of r heads in n coin flips, if $p = \Pr(heads)$

- Expected, or mean value of X, $E[X]$, is

$$E[X] \equiv \sum_{i=0}^{n} iP(i) = np$$

- Variance of X is

$$Var(X) \equiv E[(X - E[X])^2] = np(1-p)$$

- Standard deviation of X, σ_X, is

$$\sigma_X \equiv \sqrt{E[(X - E[X])^2]} = \sqrt{np(1-p)}$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Normal Distribution Approximates Binomial

$error_S(h)$ follows a *Binomial* distribution, with

- mean $\mu_{error_S(h)} = error_\mathcal{D}(h)$
- standard deviation $\sigma_{error_S(h)}$
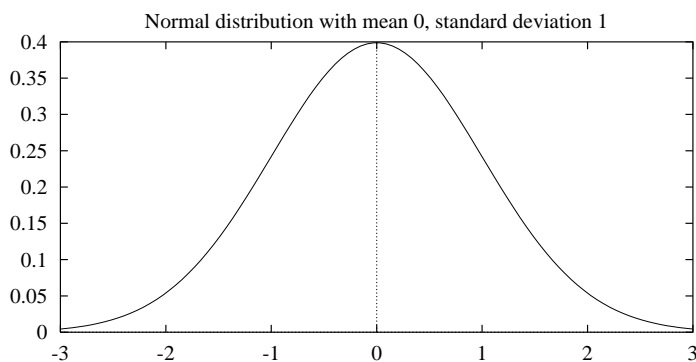
$$\sigma_{error_S(h)} = \sqrt{\frac{error_\mathcal{D}(h)(1 - error_\mathcal{D}(h))}{n}}$$

Approximate this by a *Normal* distribution with

- mean $\mu_{error_S(h)} = error_\mathcal{D}(h)$
- standard deviation $\sigma_{error_S(h)}$

$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Normal Probability Distribution

Normal distribution with mean 0, standard deviation 1

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

The probability that X will fall into the interval (a, b) is given by

$$\int_a^b p(x)dx$$

- Expected, or mean value of X, $E[X]$, is

$$E[X] = \mu$$

- Variance of X is

$$Var(X) = \sigma^2$$

- Standard deviation of X, σ_X, is

$$\sigma_X = \sigma$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Normal Probability Distribution
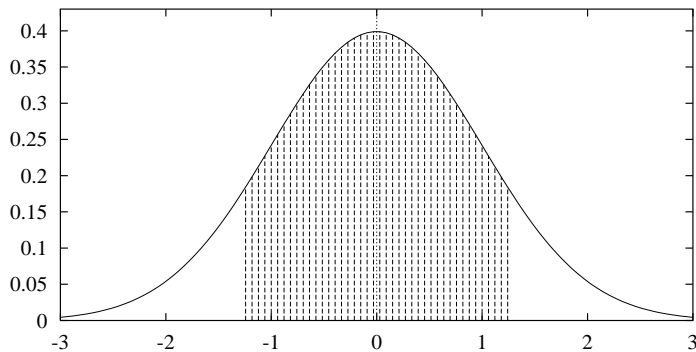
80% of area (probability) lies in $\mu \pm 1.28\sigma$

N% of area (probability) lies in $\mu \pm z_N\sigma$

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Confidence Intervals, More Correctly

If

- S contains n examples, drawn independently of h and each other

- $n \geq 30$

Then

- With approximately 95% probability, $error_S(h)$ lies in interval

$$error_\mathcal{D}(h) \pm 1.96 \sqrt{\frac{error_\mathcal{D}(h)(1 - error_\mathcal{D}(h))}{n}}$$

equivalently, $error_\mathcal{D}(h)$ lies in interval

$$error_S(h) \pm 1.96 \sqrt{\frac{error_\mathcal{D}(h)(1 - error_\mathcal{D}(h))}{n}}$$

which is approximately

$$error_S(h) \pm 1.96 \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Central Limit Theorem

Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$, all governed by an arbitrary probability distribution with mean μ and finite variance σ^2. Define the sample mean,

$$\bar{Y} \equiv \frac{1}{n} \sum_{i=1}^{n} Y_i$$

Central Limit Theorem. As $n \to \infty$, the distribution governing $\bar{Y}$ approaches a Normal distribution, with mean μ and variance $\frac{\sigma^2}{n}$.

Calculating Confidence Intervals

1. Pick parameter p to estimate

   - $error_{\mathcal{D}}(h)$

2. Choose an estimator

   - $error_S(h)$

3. Determine probability distribution that governs estimator

   - $error_S(h)$ governed by Binomial distribution, approximated by Normal when $n \geq 30$

4. Find interval (L, U) such that N% of probability mass falls in the interval

   - Use table of z_N values

Difference Between Hypotheses

Test h_1 on sample S_1, test h_2 on S_2

1. Pick parameter to estimate

$$d \equiv error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

2. Choose an estimator

$$\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$$

3. Determine probability distribution that governs estimator

$$\sigma_{\hat{d}} \approx \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}}$$

4. Find interval (L, U) such that N% of probability mass falls in the interval

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}}{n_2}}$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Paired t test to compare h_A, h_B

1. Partition data into k disjoint test sets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.

2. For i from 1 to k, do

$$\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$$

3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$$

$N\%$ confidence interval estimate for d:

$$\bar{\delta} \pm t_{N,k-1} \ s_{\bar{\delta}}$$

$$s_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2}$$

Note δ_i approximately Normally distributed

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Comparing learning algorithms L_A and L_B

What we'd like to estimate:
$$E_{S \subset \mathcal{D}}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))]$$
where $L(S)$ is the hypothesis output by learner L using training set S

i.e., the expected difference in true error between hypotheses output by learners L_A and L_B, when trained using randomly selected training sets S drawn according to distribution $\mathcal{D}$.

But, given limited data D_0, what is a good estimator?

- could partition D_0 into training set S and training set T_0, and measure
$$error_{T_0}(L_A(S_0)) - error_{T_0}(L_B(S_0))$$

- even better, repeat this many times and average the results (next slide)

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Comparing learning algorithms L_A and L_B

1. Partition data D_0 into k disjoint test sets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.

2. For i from 1 to k, do

   use T_i for the test set, and the remaining data for training set S_i

   - $S_i \leftarrow \{D_0 - T_i\}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$

3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$$

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

Comparing learning algorithms L_A and L_B

Notice we'd like to use the paired t test on $\bar{\delta}$ to obtain a confidence interval

but not really correct, because the training sets in this algorithm are not independent (they overlap!)

more correct to view algorithm as producing an estimate of

$$E_{S \subset D_0}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))]$$

instead of

$$E_{S \subset \mathcal{D}}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))]$$

but even this approximation is better than no comparison

lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997