

331 – Intro to Intelligent Systems
Week 08
Machine Learning
Decision Trees

T.J. Borrelli

Machine Learning

“Natural Selection is the blind watchmaker, blind because it does not see ahead, does not plan consequences, has no purpose in view. Yet the living results of natural selection overwhelmingly impress us with the appearance of design as if by a master watchmaker, impress us with the illusion of design and planning.”

- Richard Dawkins, “The Blind Watchmaker”

Machine Learning

- The goal of machine learning is to develop algorithms that will automate the process of decision-making based on data
 - This will allow the machine to change its *behavior* based on the data it receives by recognizing complex patterns in the data and extrapolating to new situations

Machine Learning

- An agent is **learning** if it improves its performance on future tasks after making observations about the world
- From a collection of input/output pairs, learn a function that predicts output for new inputs
- Why do this?
 - Designers cannot anticipate all possible situations
 - Designers cannot anticipate all changes over time
 - Designers may not know how to solve the problem themselves

Machine Learning

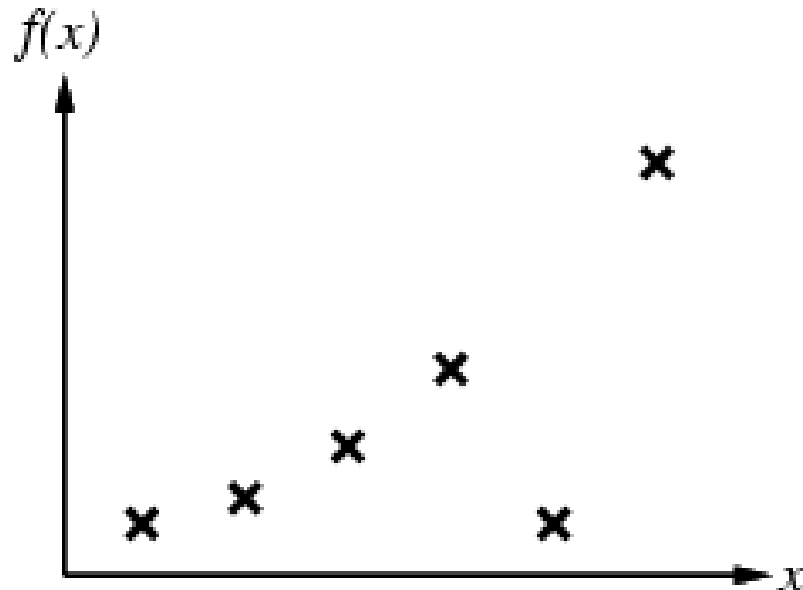
- Machine learning includes:
 - Neural networks
 - Genetic algorithms
 - Bayesian networks
 - Fuzzy Logic
 - Data mining
 - Support vector machines
 - etc.

Learning in General

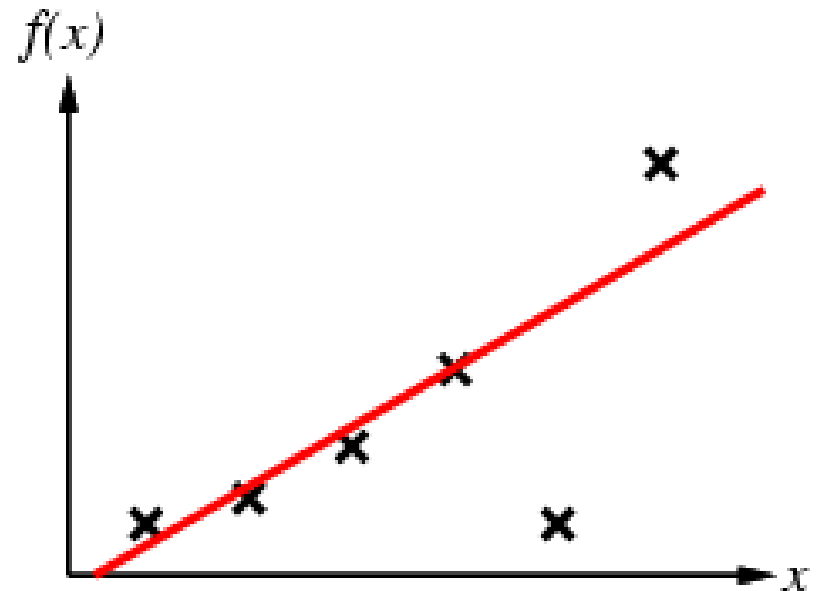
- The goal of learning is to be able to improve performance on future tasks after making observations about the world
- *Deductive learning* attempts to deduce new knowledge from rules and facts using logical inference – going from a known general rule to a new rule that is logically entailed
- *Inductive learning* attempts to learn new knowledge from examples – learning a general function or rule from specific input/output pairs

Inductive Learning Method

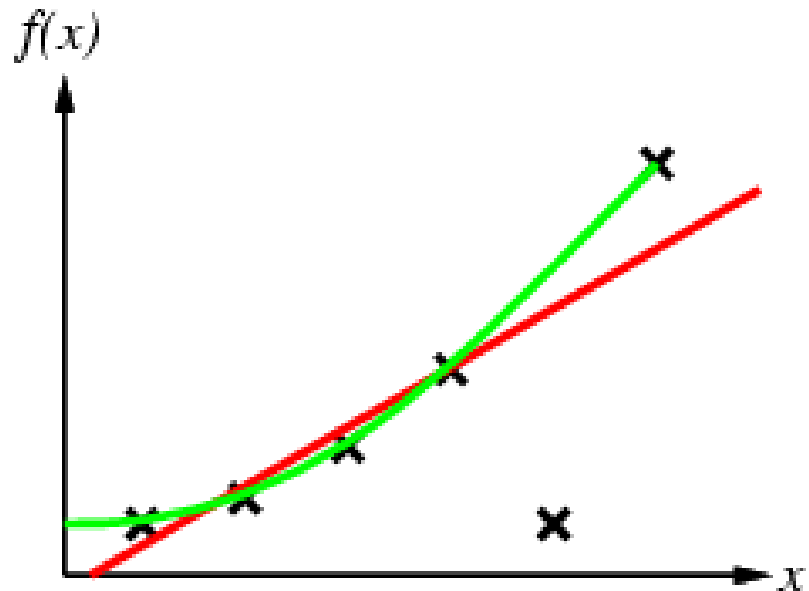
- Construct or adjust an hypothesis h to agree with a function f given a training set
- h is consistent if it agrees with f on all examples
- Consider curve fitting:



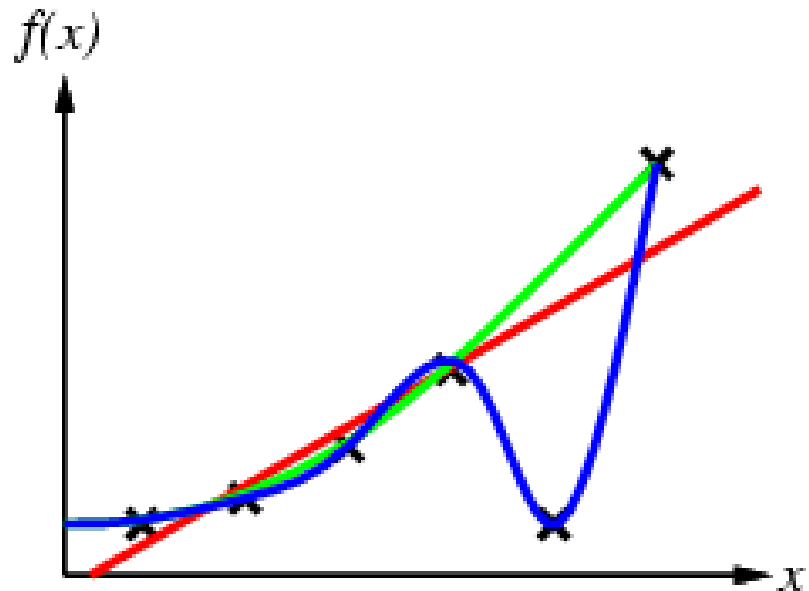
Inductive Learning Method



Inductive Learning Method



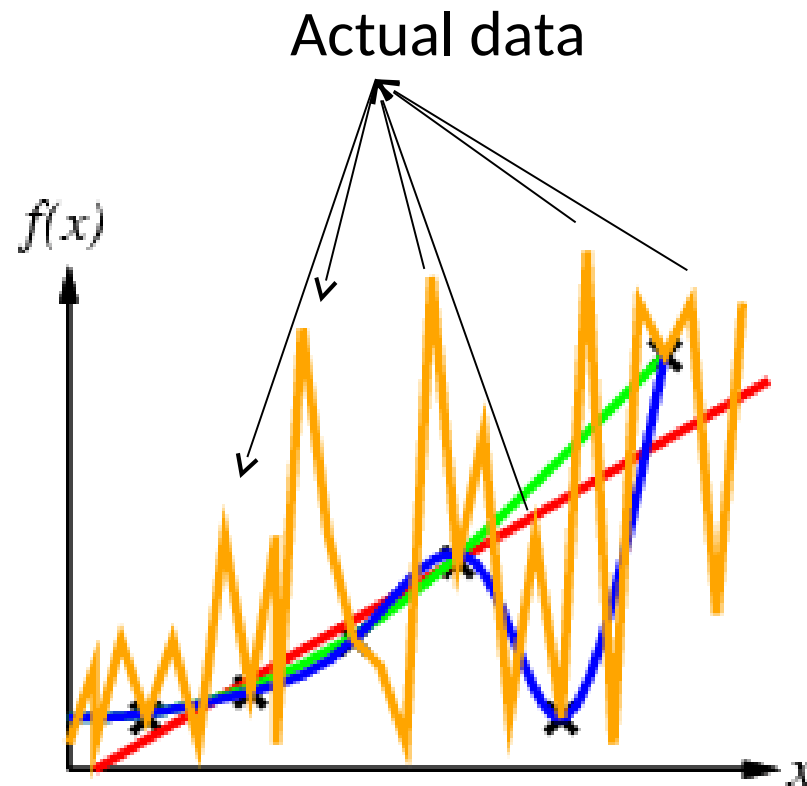
Inductive Learning Method



Inductive Learning Method

Occam's (or Ockham's) razor: prefer the simplest hypothesis that is consistent with *all* of the data. (consistent if it agrees with all data).

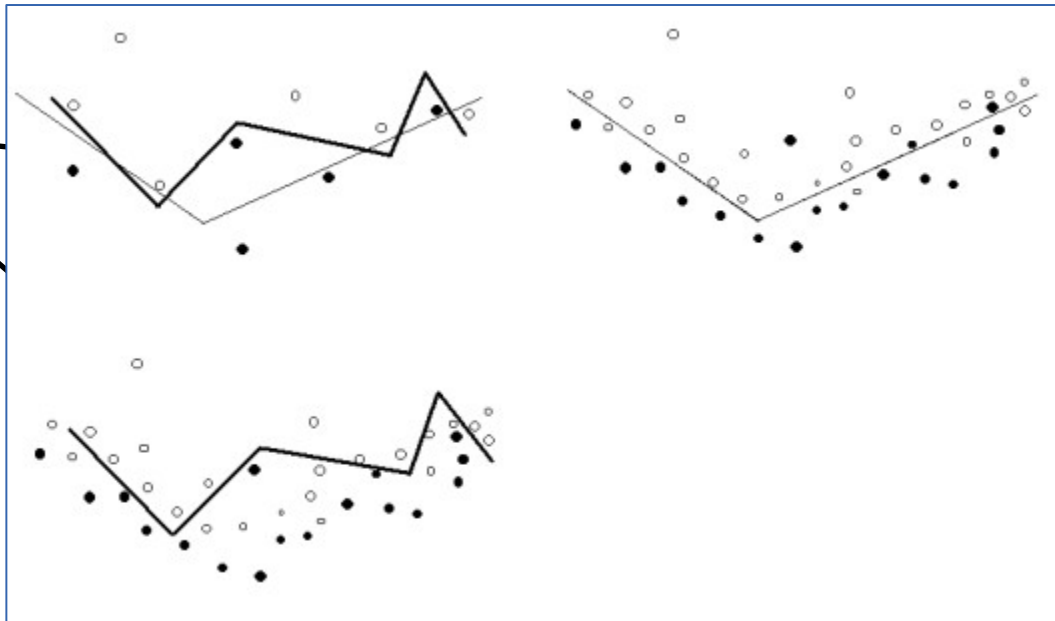
But beware of under-fitting.



The Problem of Over-fitting

The black dots represent positive examples, the gray dots represent negative examples. Goal: find the line that clearly separates the two sets of data. The two lines represent two different hypotheses.

Complex hypothesis

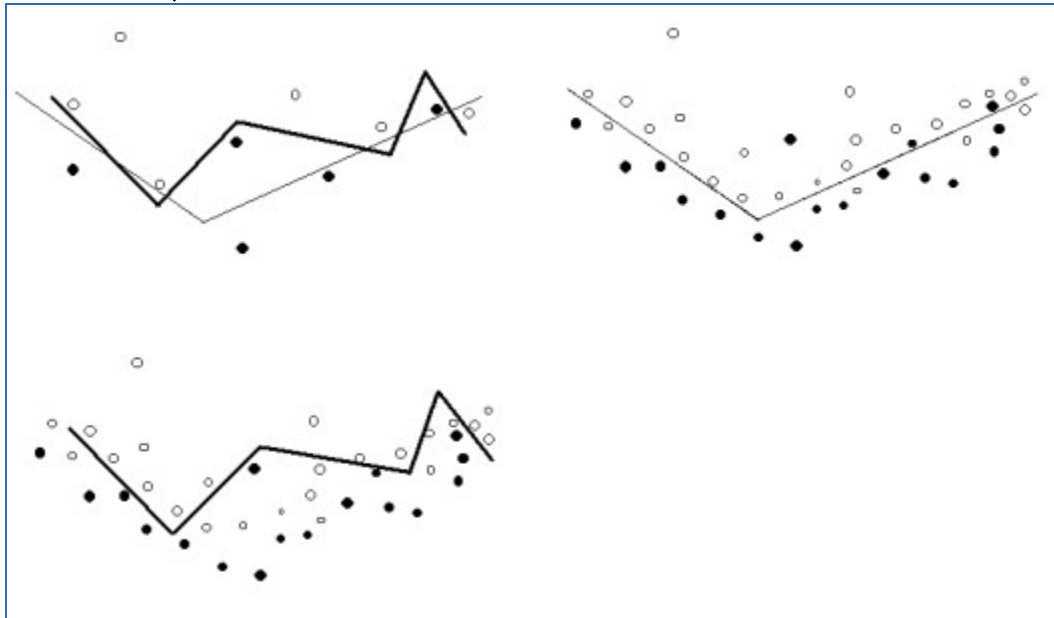


In the upper left diagram, there are just a few items of training data. These are correctly classified by the complex hypothesis. In the upper right diagram we see the complete set of data. The complex hypothesis was obviously incorrect, as shown at the lower left.

In general there is a tradeoff Between a complex hypothesis that fits the training data well and a simpler hypothesis that may generalize better.

The Problem of Over-fitting

- The simpler hypothesis, which matches the training data less well, matches the rest of the data better than the more complex hypothesis, which over-fits



Types of Inductive Learning

- Inductive learning can be *supervised* (a “teacher” is involved) or *unsupervised* (the “student” is on his or her own)
- Supervised learning is called *classification*
- Unsupervised learning is called *clustering*
- Reinforcement learning is another type of learning where there are “rewards” or “punishments” for answers that are correct or incorrect

Supervised Learning

- Also known as classification
- A classifier is designed by using a *training set* of patterns of known class for the purpose of determining the class of future sample patterns of unknown class
- A *test set* of patterns is also provided for which the true class is known, for the purpose of evaluating the effectiveness of the classifier

Non-Parametric Classification

- We do not have enough knowledge or data to be able to assume the general form of the probability model, or to estimate the relevant parameters
 - Look directly at the data instead of a summary of the data (decision trees, etc.)
 - Look at histograms, scatter plots, tables of the data (nearest-neighbor, etc.)

Nearest Neighbor Classification

- Each sample represents a “point” in feature space
- Classify unknown sample as belonging to the same class as the most similar or “nearest” sample point in the training set
- By “nearest” we usually mean the smallest distance in an n -dimensional feature space

Nearest Neighbor Classification

- Euclidean distance formula:

$$d(a,b) = \text{sqrt} \left(\sum (b_i - a_i)^2 \right)$$

- Square root is optional, to save computation time (relative distances remain the same)
- Euclidean distance emphasizes large distances
 - May want to use absolute differences in each feature instead of squared differences

Nearest Neighbor Classification

- Absolute distance formula:

$$d(a,b) = \sum |b_i - a_i|$$

- Also called “city block distance” or Manhattan distance

- Maximum distance metric

$$d(a,b) = \max |b_i - a_i|$$

- Finds only the most dissimilar pair of features

Nearest Neighbor Classification

- Minkowski distance formula:

$$d(a,b) = [\sum (b_i - a_i)^r]^{1/r}$$

- “r” is an adjustable parameter
- Generalization of the three previously defined distance metrics

Nearest Neighbor Classification

- Problem of scale:
 - An arbitrary change in the unit of measurement of one of the features could easily affect the decision
 - For example, measuring a length in millimeters rather than in meters would increase the relative contribution of this feature by a factor of 1000 compared to the other features if city block distance is used, and by 1,000,000 if Euclidean distance is used!

Nearest Neighbor Classification

- Problem of scale:
 - If one feature has a very wide range of possible values compared to the other features, it will have a very large effect on the total dissimilarity, and the decisions will be based primarily upon this single feature
 - To overcome this, it is necessary to apply scale factors to the features before computing the distances

Nearest Neighbor Classification

- Problem of scale:
 - If we want the potential influence of each of the features to be about equal, the features should be scaled so that each of them has the same standard deviation, range, or other measure of spread for the entire data set

Nearest Neighbor Classification

- Problem of scale:
 - Normalize each feature, x_i , to have a mean of 0 and a standard deviation of 1 for the entire data set:
 - i.e., replace each feature x_i with:

$$z_i = (x_i - \mu_i) / \sigma_i$$

Nearest Neighbor Classification

- Problem of scale:
 - If some prior knowledge of the relative importance of the features is available, the features could be scaled according to their importance or desired contribution to the decision making
 - Let the range, standard deviation, or weight of each normalized feature be proportional to the “accuracy” of that feature (where “accuracy” is defined as the probability of a correct decision when that feature is used alone)

K-Nearest Neighbor (k-NN)

- Base the classification of a sample on the number of “votes” of k of its nearest neighbors, rather than on only its single nearest neighbor - denoted k -NN
- Choosing k too large tends to suppress the fine structure of the underlying density, while choosing k too small puts too much emphasis on the chance locations of a few sample points
- In practice, various values of k are tried on the training set, and the one that gives the best result is chosen

K-Nearest Neighbor (k-NN)

- k-NN suffers from “the curse of dimensionality”
- In low-dimensional spaces with plenty of data nearest neighbor works well
- As the number of dimensions rises the nearest neighbors are usually not very near
 - Most neighbors are “outliers”!
- In addition, search can be difficult if there are a large number of samples – $O(N^2)$
- We would like sub-linear time complexity

Application: Pandora

- Pandora is an Internet music radio service that allows users to build customized “stations” that play music similar to a song or artist that is specified
- Pandora uses a k-NN style process called *The Music Genome Project* to locate new songs or artists that are similar to the user-specified song or artist

Application: Pandora

The Pandora process in general:

Hundreds of variables are created on which a song can be measured on a scale from 0 to 5 (e.g., acid-rock, accordion playing, etc.)

Pandora pays musicians to analyze songs and rate each one on each variable

The user specifies a song he or she likes (must be in Pandora's database)

The distance between user's choice and every song in the database is calculated

User has option to say "I like this song", "I don't like this song", or nothing

If "like" is chosen, the song and closest song are merged into a cluster

Classification Errors

- Classification errors arise when an observation that is known to belong to a certain class is classified as belonging to a different class
- A very simple rule for classification is to classify the observation as belonging to the most prevalent class (i.e., ignore any predictor results)
- This is known as the *naïve rule*, and is used as a baseline or benchmark for evaluating the performance of more complicated classifiers
 - A classifier that uses predictor results should outperform this

Estimation of Error Rate

- After developing a classification procedure, we must evaluate its performance
 - Model-based error estimation (assumes distributions are known)
 - Counting-based error estimation (assumes true classification or “ground-truth” is known for a sample test set)
 - “Confusion matrix” (also known as a contingency table)

Learning Decision Trees

- Decision tree induction is one of the simplest and most successful forms of machine learning
- A decision tree represents a function that takes as input a vector of attribute values and returns a decision (single output value)
- A decision tree reaches its decision by performing a sequence of tests on each (non-terminal) node in the tree
- Restaurant example

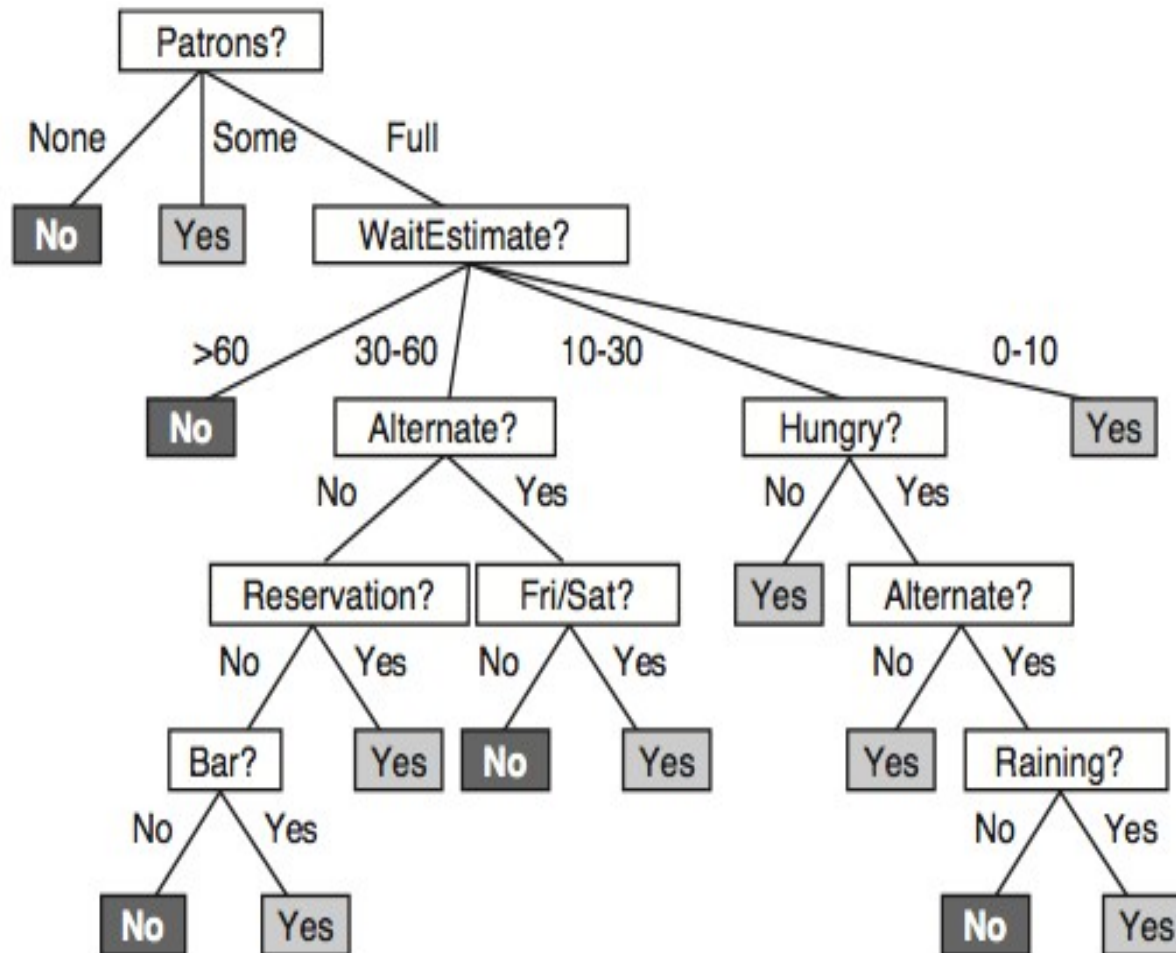
Restaurant Example

- We wish to build a decision tree to decide whether to wait for a table at a restaurant
- Goal predicate: *WillWait*
- List of attributes to consider
 - Alternate, Bar, Fri/Sat, Hungry, Patrons, Price, Raining, Reservation, Type, WaitEstimate

Restaurant Example

- List of attributes to consider
 - Alternate: is there another suitable restaurant nearby
 - Bar: does the restaurant have a comfy bar area
 - Fri/Sat: is it friday or saturday
 - Hungry: are we *really* hungry
 - Patrons: how many people are here (None, Some, Full)
 - Price: 3 categories (\$, \$\$, \$\$\$)
 - Raining: whether it is raining outside
 - Reservation: whether we made a reservation
 - Type: kind of restaurant (French, Italian, Thai, burger)
 - WaitEstimate: wait time estimated by host (<10min, 10-30min, 31-60min, >60min)

Decision Tree – deciding whether to wait for a table



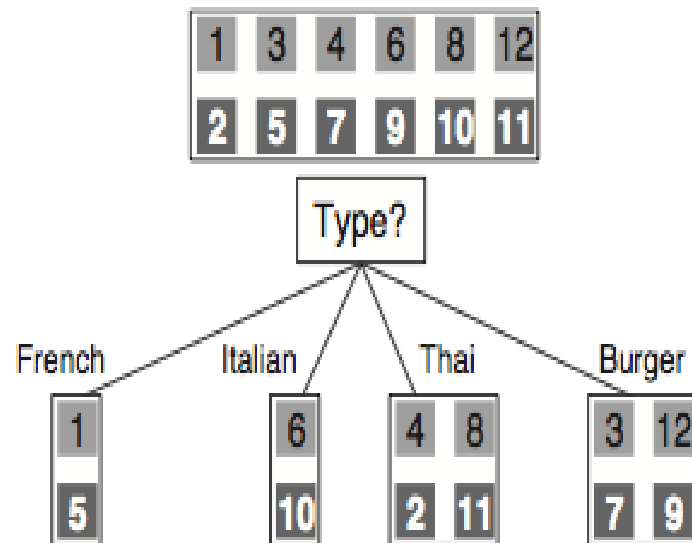
Decision Trees

- For a wide variety of problems, the decision tree format yields a nice, concise result
- Some problems cannot be represented concisely (explodes into exponentially many nodes)
- Our goal with the decision tree is to create one that is consistent with the input/output pairs and is as small as possible
- Often finding the *smallest* tree is also intractable so we may need to use heuristics

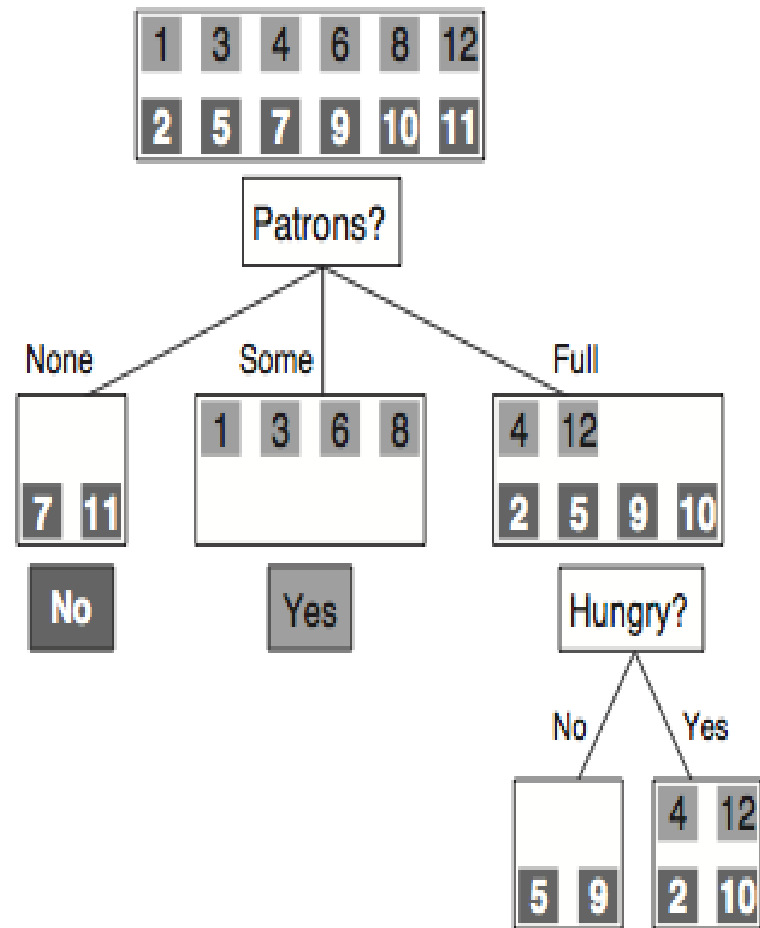
Non-exhaustive list of inputs and outputs

Num	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	Wait
x_1	yes	no	no	yes	some	\$\$\$	no	yes	French	0-10	yes
x_2	yes	no	no	yes	full	\$	no	no	Thai	30-60	no
x_3	no	yes	no	no	some	\$	no	no	Burger	0-10	yes
x_4	yes	no	yes	yes	full	\$	yes	no	Thai	10-30	yes
x_5	yes	no	yes	no	full	\$\$\$	no	yes	French	>60	no
x_6	no	yes	no	yes	some	\$\$	yes	yes	Italian	0-10	yes
x_7	no	yes	no	no	none	\$	yes	no	Burger	0-10	no
x_8	no	no	no	yes	some	\$\$	yes	yes	Thai	0-10	yes
x_9	no	yes	yes	no	full	\$	yes	no	Burger	>60	no
x_{10}	yes	yes	yes	yes	full	\$\$\$	no	yes	Italian	10-30	no
x_{11}	no	no	no	no	none	\$	no	no	Thai	0-10	no
x_{12}	yes	yes	yes	yes	full	\$	no	no	Burger	30-60	yes

Possible Decision Trees



(a)



(b)

Decision Trees

- Splitting on *Type* (on last slide) brings us no closer to being able to distinguish between “Yes” and “No” answers
- Splitting on *Patrons* however does a good job of separating “Yes” from “No” selections
- We still have some work to do, but we found some leaf nodes early on

Decision Trees

- When creating our decision tree the goal is to approximately minimize its depth
- Therefore we should pick attributes in such a way that we can classify inputs as soon as possible
- A “perfect” attribute is one that divides the examples into sets, each of which are all positive or all negative (thus terminal)
- A really useless attribute such as Type on the previous slide, leaves the example sets with roughly the same proportion of positive and negatives examples as in the original set

Decision Trees and Entropy

- How do we know which attributes to pick so that we pick good ones?
- (Shannon) Entropy – from information theory – a measure of the uncertainty of a random variable
- As information goes up (i.e. we make new observations) – entropy goes down
- Consider a (fair) coin flip – equally likely to come up heads or tails (0 or 1) – this corresponds to “1 bit” of entropy
- Roll of a 4-sided die has 2 bits of entropy – takes 2 bits to describe each of the 4 outcomes

331 – Intro to Intelligent Systems

Week09

Bayesian Belief Networks

T.J. Borrelli



Bayesian Network

- Causality – if *this*, then *that*
- But we do not know the whole picture
- Describe events in terms of probabilities
- Allows us to reverse direction of probabilistic statement



Uses of BBNs

- Predict what will happen
- Or to infer causes from observed events
- Allow calculation of conditional probabilities of the nodes in the network, given that the values of some of the nodes have been observed



Types of Inference for BBNs

- Causal Inference (abduction, prediction) - Given that A implies B, and we know the value of A, we can compute the value of B
- Diagnostic Inference (induction) - Given that A implies B, and we know the value of B, we can compute the probability of A



Types of Inference

- Intercausal Inference (“explaining away”) -
Given that both A and B imply C, and we know C, we can compute how a change in the probability of A will affect the probability of B, even though A and B were originally independent

Bayesian Belief Network (BBN)



- Directed Acyclic Graph
- Causal relations between variables
- Models cause-and-effect relations
- Allows us to make inferences based on limited knowledge of environment and known probabilities
- Allows us to reverse direction of probabilistic statement



BBN

- Certain Independence assumptions must hold between random variables
- To specify the probability distribution of a Bayesian network
 - Need to know the *a priori* probabilities of all root nodes (nodes with no predecessors)
 - Conditional probabilities of all non-root nodes



Logically

- Only part of the story is considered in the causal (cause-and-effect) relationship between phenomena
- It is $P \rightarrow Q$ not $P \leftrightarrow Q$
- Meaning there could be other potential causes of Q



Directed and Acyclic Graph

- The directions show the causal relations between events
 - thus somewhat important
- If cycles are present we have a Causal Loop



Bayes' paper

- Bayes described in his famous paper (in 1763) “degree-of-belief” as opposed to classical probability (or fuzzy logic which has partial set membership)
- As new information is added to our defined system of beliefs, the probabilities of events occurring should be updated
- Also known as “Bayesian Inference”



Bayes says a lot of things ...

- However in his original paper, didn't explicitly state the rule attributed to him (conveniently called Bayes' Rule or Bayes' Theorem). So I will state it here:
- $P(A | B) = (P(B | A) * P(A)) / P(B)$



Explanation of Bayes' Rule

- $P(A | B) = (P(B | A) * P(A)) / P(B)$
- Means that the Probability of event A occurring given that event B has occurred is equal to the probability of event B occurring given that A has occurred times the independent probability of A divided by the independent probability of B

Derivation of Bayes' Rule



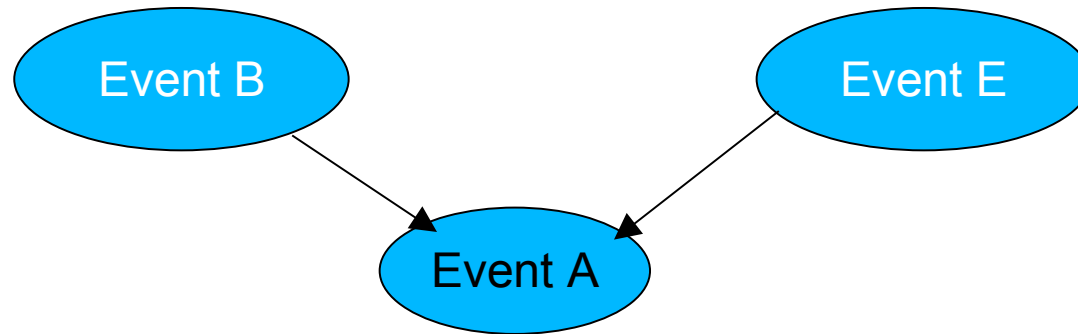
- By definition of Conditional Probabilities:
- $P(A | B) = P(A \cap B) / P(B)$
- By the same note: $P(B | A) = P(B \cap A) / P(A)$
- By the commutative law of probability and set theory: $P(A \cap B) \equiv P(B \cap A)$
- $P(A \cap B) = P(A | B) * P(B) = P(B | A) * P(A)$
- Thus: $P(A | B) = (P(B | A) * P(A)) / P(B)$



Definitions

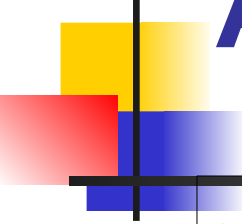
- Bayes' Rule: $P(A | B) = (P(B | A) * P(A)) / P(B)$
- $P(A)$ is “prior probability” or “marginal probability” of A
- $P(A|B)$ is “conditional probability of A, given B. Also called “posterior probability”
- $P(B)$ is also a “prior” or “marginal” probability and acts as a normalizing constant so event probabilities add to 1

Belief Network



- Event A is “Alarm”
- Event B is “Burglary”
- Event E is “Earthquake”

Alarm Example



t	.001
f	.999

Event B

Event E

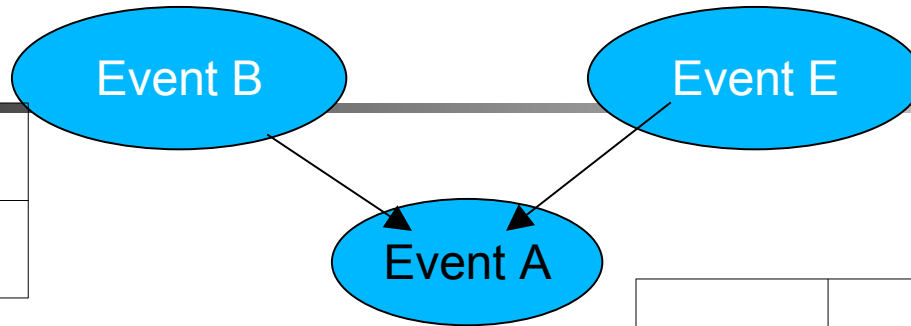
Event A

t	.002
f	.998

- **Given**
 - Initial Truth Tables
 - Causal relationships

Alarm Example-2

T	.001
F	.999



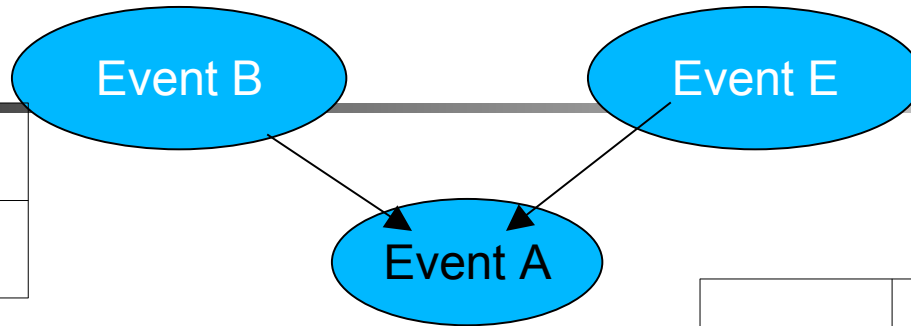
T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

- **Given**
 - Initial Truth Tables
 - Causal relationships
 - Conditional Probabilities
- **Can calculate conditional probabilities based on new information**

Alarm Example-3

T	.001
F	.999



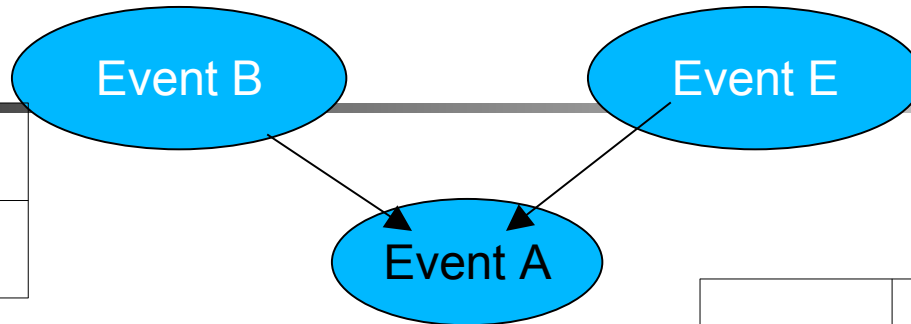
T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

$$\begin{aligned}
 P(A) &= P(A|B,E) \cdot P(B) \cdot P(E) + \\
 &P(A|\neg B,E) \cdot P(\neg B) \cdot P(E) + \\
 &P(A|B,\neg E) \cdot P(B) \cdot P(\neg E) + \\
 &P(A|\neg B,\neg E) \cdot P(\neg B) \cdot P(\neg E) \\
 &= .002516
 \end{aligned}$$

Alarm Example-4

T	.001
F	.999



T	.002
F	.998

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001

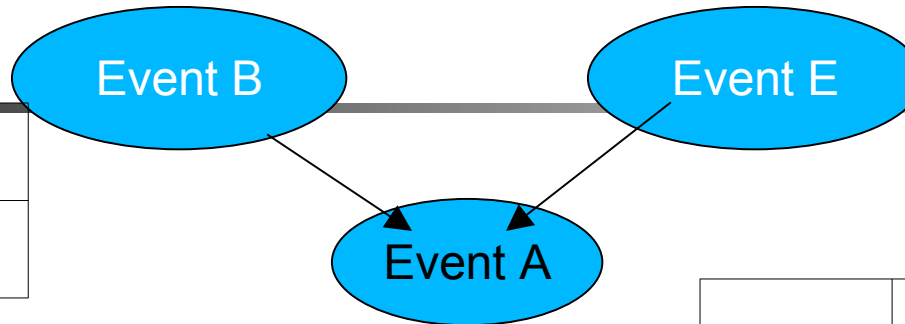
- Now Suppose We Observe That E is True!

....

What does this change in the previous equation?

Alarm Example-5

T	.001
F	.999



T	.002
F	.998

- E is True!
- What does this change in the previous equation?
- $P(A) = P(A|B,E)*P(B)*P(E) +$
 $P(A|\neg B,E)*P(\neg B)*P(E) +$
 $P(A|B,\neg E)*P(B)*P(\neg E) +$
 $P(A|\neg B,\neg E)*P(\neg B)*P(\neg E) =$
 $P(A|B,E)*P(B) + P(A|\neg B,E)*P(\neg B) = .95$
 $* .001 + .29 * .999 = .291$

B	E	A
T	T	.95
T	F	.94
F	T	.29
F	F	.001



Example

- Suppose we want to know the chance that it rained yesterday if we suddenly find out that the lawn is wet
- By Bayes' rule we can calculate this probability from it's inverse: the probability that the lawn would be wet if it *had* rained yesterday


$$P(A|B) = (P(B|A) * P(A)) / P(B)$$

- $P(A|B)$ = “the probability of event A, given that we know B” - for example, the probability that it rained yesterday given that the lawn is wet”
- $P(A)$ = the probability of rain, all other things being equal
- $P(B)$ = the probability of the lawn being wet, all other things being equal

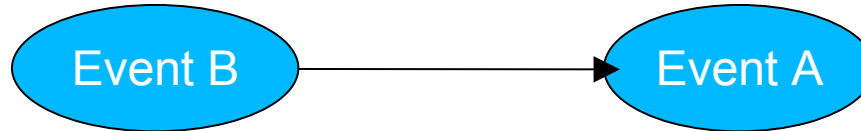


What did we just do?

- We were able to rephrase the probability in terms of the probability of B given A
($P(B|A)$) and independent probabilities $P(A)$ and $P(B)$

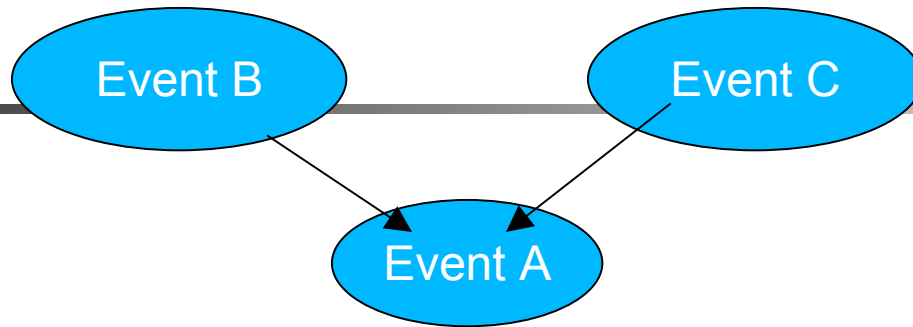


More Probability



- As an extension of Bayes' Theorem:
- Given that event A is dependent upon event B
- $P(A) = P(A|B) * P(B) +$
 $P(A|\neg B) * P(\neg B)$

Similarly



- Event A is dependent upon event B and C
- $$P(A) = P(A|B,C) * P(B) * P(C) +$$
$$P(A|B, \neg C) * P(B) * P(\neg C) +$$
$$P(A|\neg B, C) * P(\neg B) * P(C) +$$
$$P(A|\neg B, \neg C) * P(\neg B) * P(\neg C)$$



Problems of Inconsistency

- If we look at the following “reasonable-looking” situation:
 - $P(A|B) = .8$
 - $P(B|A) = .2$
 - $P(B) = .6$
- Bayes' Rule: $P(A|B) = (P(B|A)*P(A))/ P(B)$



Inconsistency

- $P(A|B) = (P(B|A) * P(A)) / P(B)$
- $.8 = .2 * P(A) / .6$
- $.48 = .2 * P(A)$
- Thus, $P(A) = 2.4$

- So, What's Wrong With This?



Skunk in the Woodpile

- You can't see it, but you know it's there
- The probability for an event cannot exceed 1
- Problem: failure to keep probabilities consistent
- This is sometimes difficult to see at first
- Need to check for this



Some Uses

- Used in medical problem diagnosis (PATHFINDER) when only limited information and events is known
- Used in map learning and language processing and understanding
- Used in Games to provide human-like reasoning based on incomplete information. (First Person Sneaker)



A Good Paper to Read

- Charniak, E. *Bayesian Networks without Tears*. American Association for Artificial Intelligence. AI Magazine. 1991.

331 – Intro to Intelligent Systems

Week10b

Genetic Algorithms

T.J. Borrelli

Genetic Algorithms

- Genetic techniques can be applied with a range of representations
- The best representation depends on the problem to be solved
- Usually, we start with a hypothesis consisting of a population of chromosomes (problem states)
- Each chromosome consists of a number of genes (the features that make up a state)

Basic Genetic Algorithm

Generate a random population of chromosomes (the first generation).

**If termination criteria are satisfied, stop.
Otherwise, continue with step 3.**

Determine the fitness of each chromosome.

Apply crossover and mutation to selected genes from the current generation of chromosomes to generate a new population of chromosomes (the next generation).

Return to step 2.

Fitness

- Fitness is an important concept in genetic algorithms
- The fitness of a chromosome determines how likely it is that it will reproduce
- Fitness is usually measured in terms of how well the chromosome solves some goal problem
 - For example, if the genetic algorithm is to be used to sort numbers, then the fitness of a chromosome will be determined by how close to a correct sorting it produces
- Fitness can also be subjective (aesthetic)

Crossover

Crossover is applied as follows:

- Select a random crossover point.

- Break each chromosome into two parts, splitting at the crossover point.

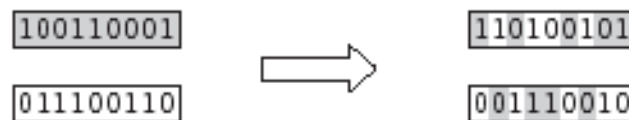
- Recombine the broken chromosomes by combining the front of one with the back of the other, and vice versa, to produce two new chromosomes.

Crossover

- Usually, crossover is applied with one crossover point, but can be applied with more, such as in the following case which has two crossover points:



- Uniform crossover involves using a probability to select which genes to crossover



Mutation

- A mutation is a unary operator (it applies to only one gene)
- A mutation randomly selects some genes (bits) to be “flipped”

010101110001001

↓

010101110**1**01001

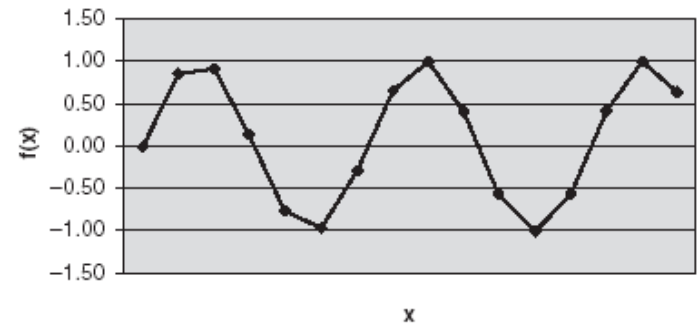
- Mutation is usually applied with a low probability, such as 1 in 1000

Termination Criteria

- A genetic algorithm is run over a number of generations until the termination criteria are reached
- Typical termination criteria are:
 - Stop after a fixed number of generations
 - Stop when a chromosome reaches a specified fitness level
 - Stop when a chromosome succeeds in solving the problem, within a specified tolerance
- Human judgment can also be used in subjective cases

Optimizing a Mathematical Function Using a Genetic Algorithm

A genetic algorithm can be used to find the maximum value for x in $f(x) = \sin(x)$, over the range of x from 0 to 15, where x is in radians.



We will use a population size of 4 chromosomes. The first step is to generate a random population, which is our first generation:

$c1 = 1001$

$c2 = 0011$

$c3 = 1010$

$c4 = 0101$

Optimizing a Mathematical Function Using a Genetic Algorithm

To calculate the fitness of a chromosome, we need to first convert it to a decimal integer, and then calculate $f(x)$ for this integer.

We will assign fitness as a numeric value from 0 to 100, where 0 is the least fit and 100 is the most fit.

$f(x)$ generates real numbers between -1 and 1. We will assign a fitness of 100 to $f(x) = 1$ and a fitness of 0 to $f(x) = -1$. Fitness of 50 will be assigned to $f(x) = 0$. Hence, the fitness of x , $f'(x)$, is defined as follows:

$$f'(x) = 50 (f(x) + 1) = 50 (\sin(x) + 1)$$

Optimizing a Mathematical Function Using a Genetic Algorithm

The fitness ratio of a chromosome is that chromosome's fitness as a percentage of the total fitness of the population. This table shows the calculations that are used to find the fitness values for the first generation:

Chromosome	Genes	Integer Value	$f(x)$	Fitness	Fitness Ratio
c1	1001	9	0.41	70.61	46.30%
c2	0011	3	0.14	57.06	37.40%
c3	1010	10	-0.54	22.80	14.90%
c4	0101	5	-0.96	2.05	1.34%

Optimizing a Mathematical Function Using a Genetic Algorithm

Now we need to run a single step of the genetic algorithm to produce the next generation. The first step is to select which chromosomes will reproduce.

Roulette-wheel selection involves using the fitness ratio to randomly select chromosomes to reproduce. This is done as follows: The range of real numbers from 0 to 100 is divided up between the chromosomes proportionally to each chromosome's fitness. Hence, in the first generation, c1 has 46.3% of the range (i.e., from 0 to 46.3), c2 will have 37.4% of the range (i.e., from 46.4 to 83.7), and so on.

Optimizing a Mathematical Function Using a Genetic Algorithm

A random number is now generated between 0 and 100. This number will fall in the range of one of the chromosomes, and this chromosome will be selected for reproduction. The next random number is used to select this chromosome's mate.

This technique helps to ensure that populations do not stagnate, due to constantly breeding from the same parents.

We will generate 4 random numbers to find the four parents that will produce the next generation. Suppose the first random number is 56.7 (c2 is chosen), and next number is 38.2 (c1 is chosen).

Optimizing a Mathematical Function Using a Genetic Algorithm

We will now combine c1 and c2 to produce 2 new offspring. First, we randomly select the crossover point to be between the second and third genes (bits):

10 | 01

00 | 11

Crossover is now applied to produce 2 offspring, c5 and c6:

c5 = 1011

c6 = 0001

In a similar way, c1 and c3 are chosen to produce offspring c7 and c8, using a crossover point between the third and fourth bits:

c7 = 1000

c8 = 1011

Optimizing a Mathematical Function Using a Genetic Algorithm

The population c1 to c4 is now replaced by the second generation, c5 to c8 (c4 did not have a chance to reproduce, so its genes will be lost “survival of the fittest”). The fitness values for the second generation are below:

Chromosome	Genes	Integer Value	$f(x)$	Fitness	Fitness Ratio
c5	1011	11	-1	0.00	0.00%
c6	0001	1	0.84	92.07	48.10%
c7	1000	8	0.99	99.47	51.90%
c8	1011	11	-1	0.00	0.00%

At this point, c7 has been found to be the optimal solution, and the termination criteria would probably determine that the run could stop. (We could set a threshold value for fitness that must be exceeded.)

The Building Block Hypothesis

- Genetic algorithms manipulate short, low-order, high fitness schemata in order to find optimal solutions to problems
- These short, low-order, high fitness schemata are known as “building blocks”
- Hence genetic algorithms work well when small groups of genes represent useful features in the chromosomes
- This tells us that it is important to choose a correct representation!

Representations for GAs

- The representation selected must support the genetic operators
- Sometimes the bit-level representation is most natural
 - Crossover and mutation can be used to directly produce potential solutions
- What about using a GA to find potential solutions for the Traveling Salesperson problem?
 - Mutations and crossovers must preserve the property that the offspring have to be *legal paths* through all the cities, visiting each *only once*
 - How can a “good path” be passed on to a future generation?

Representations for GAs

- What do we mean by the “naturalness” of a representation?
- Example: Suppose we want our genetic operators to be able to order the numbers 6, 7, 8, and 9
- An integer representation gives a very natural and evenly spaced ordering
 - Within base 10 integers, the next item is simply one more than the previous
 - What if we used binary numbers as the representation?

Representations for GAs

Consider the bit pattern for 6, 7, 8, and 9:

0110 0111 1000 1001

Between 6 and 7, and between 8 and 9 there is a 1 bit change. Between 7 and 8 all four bits change! This representational anomaly can be a huge problem when trying to generate a solution that requires organizing these four bit patterns.

A number of techniques, usually under the general heading of *gray coding*, address this problem of non-uniform representation. In a gray code, each number is exactly one bit different from its neighbors. Using gray coding instead of standard binary numbers, the genetic operator's transitions between states of near neighbors is natural and smooth.

Gray Codes

The gray-coded bit patterns for the binary numbers 0 – 15

Binary	Gray
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Properties of Genetic Algorithms

- An important strength of GAs is the parallel nature of search
 - GAs implement a powerful form of hill-climbing that maintains multiple solutions, eliminates the unpromising, and improves good solutions
 - Initially, the solutions are scattered through the space of possible solutions; after several generations they tend to cluster around areas of higher solution quality

Properties of Genetic Algorithms

- GAs, unlike sequential forms of hill-climbing, do not immediately discard unpromising solutions
 - Even weak solutions may contribute to future candidates
- There is no strict ordering of states on an open list, as we saw with A^* , DFS, or BFS
 - Rather, there is simply a population (family) of fit solutions to a problem, each potentially available to help produce new possible solutions within a paradigm of parallel search

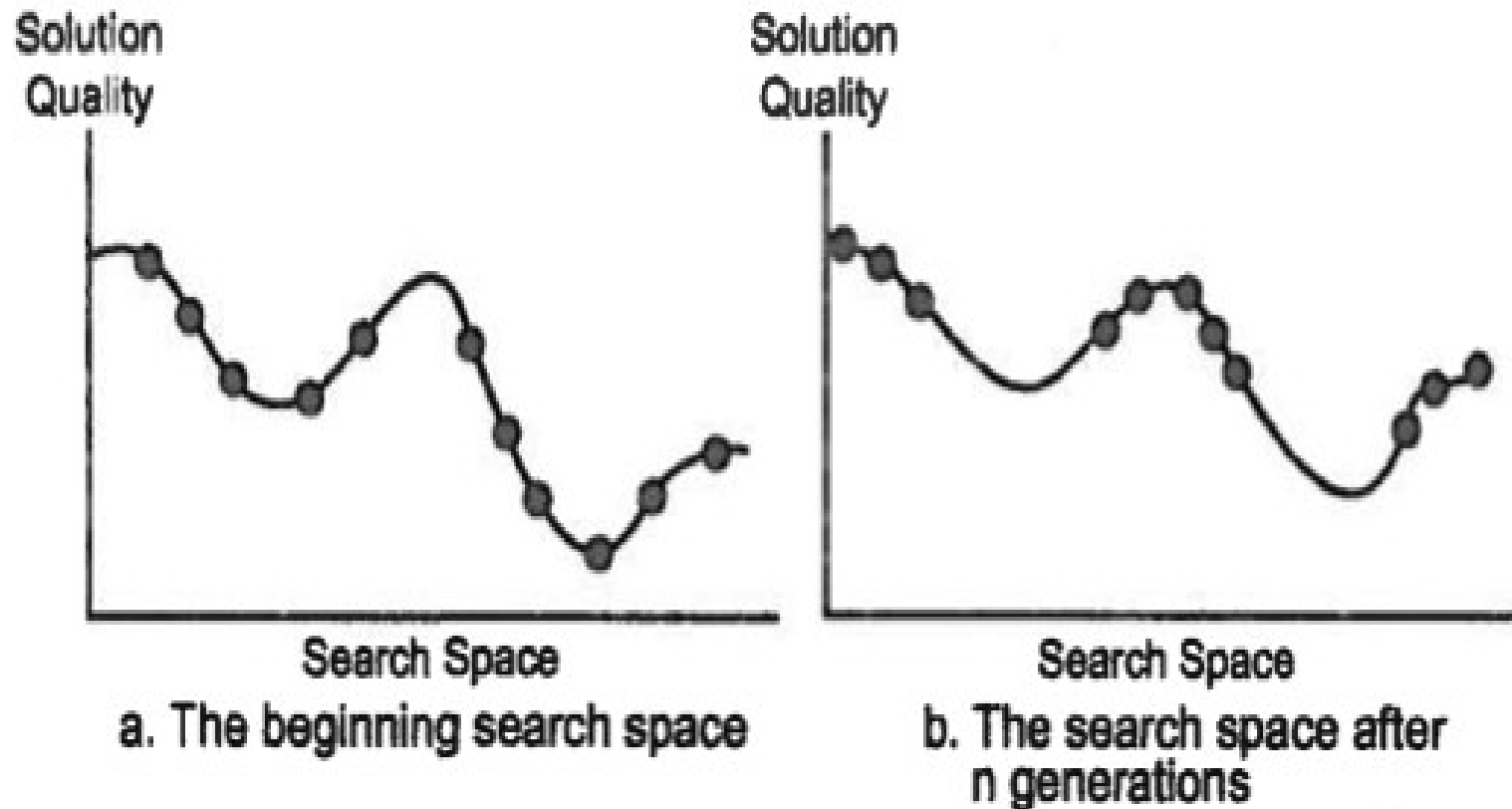


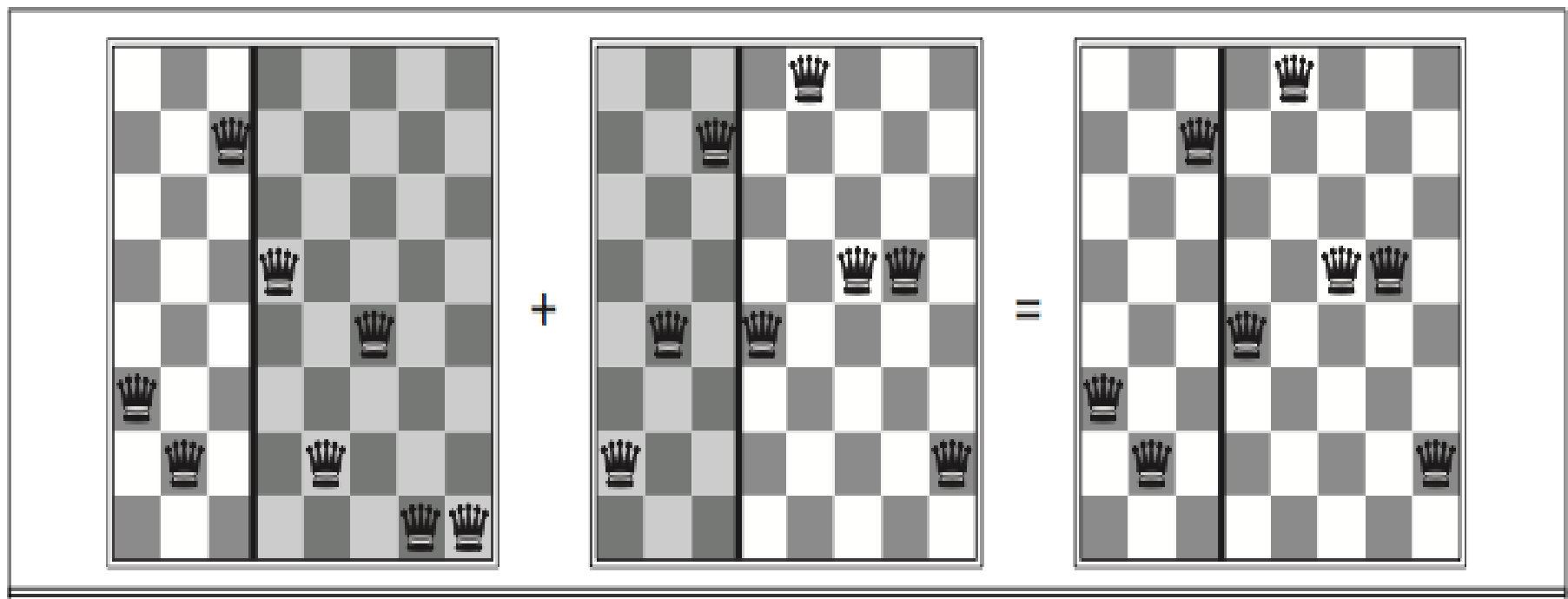
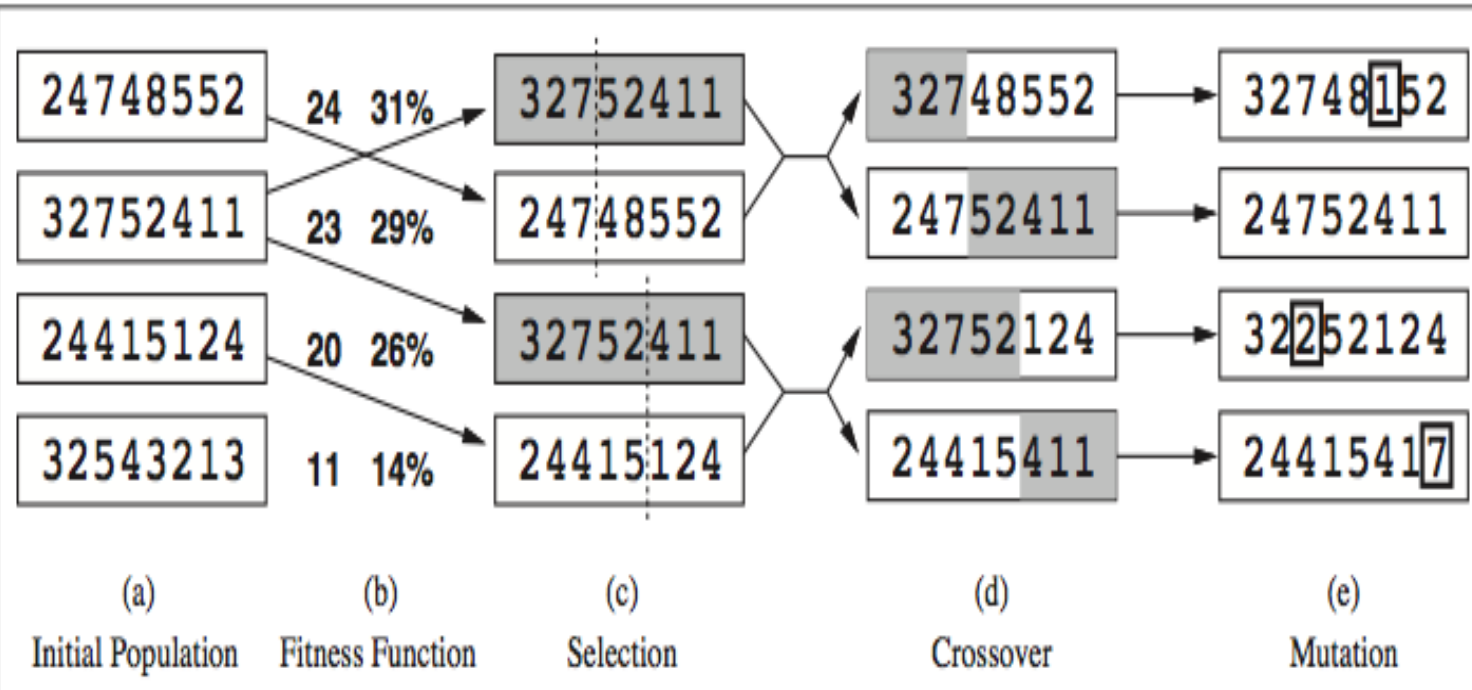
Figure 12.2 Genetic algorithms visualized as parallel hill climbing, adapted from Holland (1986).

Properties of Genetic Algorithms

- An important strength of GAs is in the parallel nature of its search
- GAs implement a powerful form of hill climbing that maintains multiple solutions, eliminates the unpromising, and improves on good solutions

Schema

- Often used by GAs to solve problems
- A **schema** is a substring in which some of the positions can be left unspecified
- For example, considering the 8-queens problem, the string $246*****$ describes all 8-queens states in which the first three queens are in positions 2, 4 and 6, respectively
- Strings that match the scheme (e.g. 24613578) are called *instances* of the schema

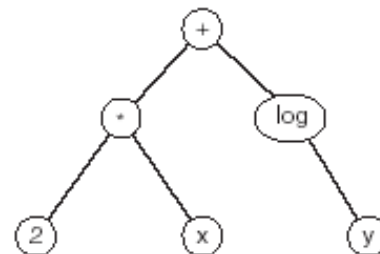


Schema

- It can be shown that if the average fitness of the instances of a schema is above the mean fitness, then the number of instances of the schema within a population will grow over time
- GAs work best when schemata correspond to meaningful components of a solution

Genetic Programming

- Genetic programming is a method used to evolve LISP S-expressions
- The S-expressions are represented as trees
- This diagram below shows an example of a tree representation of the S-expression $(+(*2x)(\log y))$
- A random set of expressions is generated, and the “fittest” individuals reproduce to produce the next generation

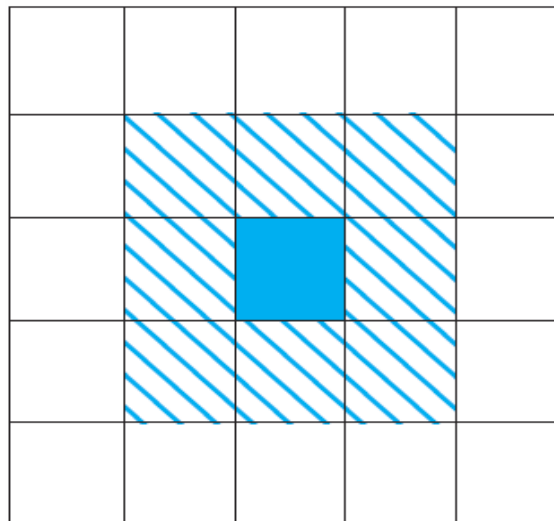


What is Life?

- What are the defining features of life?
 - Self-reproduction
 - Ability to evolve by Darwinian natural selection
 - Response to stimuli
 - Ability to die
 - Growth or expansion
- Not all living things obey these rules, and some things that are not alive do!
- Defining life is very difficult!

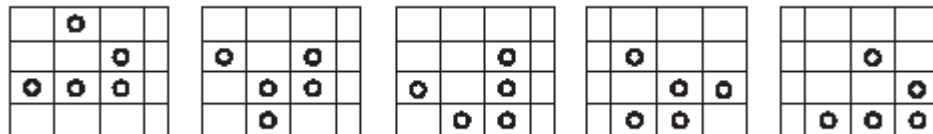
Conway's Life

- Conway's Life is a game modeled on a two dimensional cellular automaton (grid of cells), where each cell can be alive or dead
- The cross-hatched cells indicate the set of neighbors for the center shaded cell (8-neighborhood)



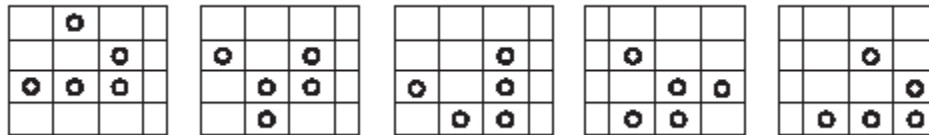
Conway's Life

- A set of rules determines how the cells will change from one generation to the next:
 1. A dead cell will come to life if it has three living neighbors
 2. A living cell with two or three living neighbors, will stay alive
 3. A living cell with fewer than two living neighbors will die of loneliness
 4. A living cell with more than three living neighbors will die of overcrowding



Conway's Life

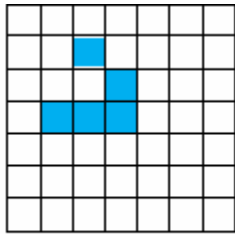
- Surprisingly complex “life-like” behavior can emerge from these simple rules
- This diagram shows a successive sequence of generations of Conway's Life



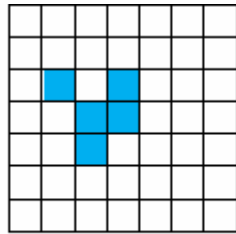
- This pattern is known as a “glider”
- There is also a pattern known as a “glider gun” which constantly fires out gliders

Conway's Life

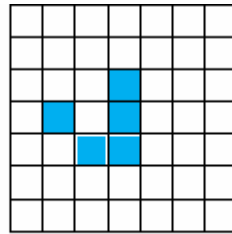
A glider moves across the display:



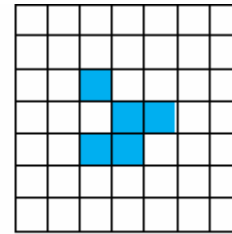
time 0



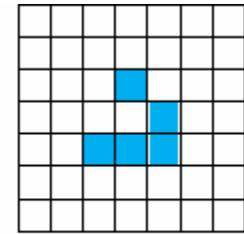
time 1



time 2

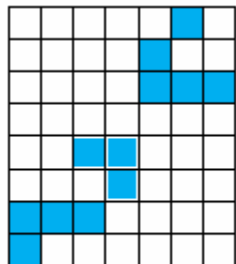


time 3

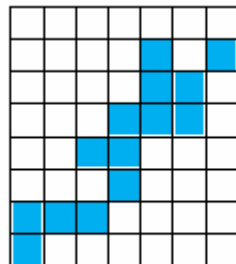


time 4

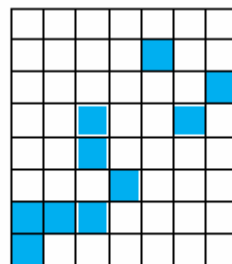
A glider is consumed by another entity:



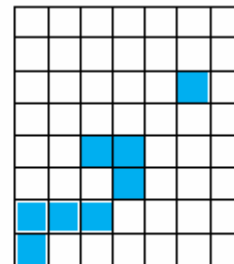
time 0



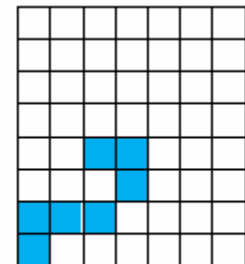
time 1



time 2



time 3



time 4

Emergent Behavior

- Emergent behavior is the idea that complex behavior can emerge from simple rules
- This is particularly prevalent in systems based on evolutionary methods, such as genetic algorithms
- Example:
 - Boids – simulation of flocking behavior of birds given very simple rules about how birds fly
 - The simulated flock learns to fly in such a way as to avoid large obstacles without being taught explicitly how to do so

Self-Reproducing Systems

- Von Neumann proposed a self-reproducing system based on cellular automata
- Ultimately we may have robots that can obtain the raw materials necessary to produce new versions of themselves
- This would be useful for exploring other planets, among other things
- Consider an “intelligence explosion” and “The Singularity”

331 – Intro to Intelligent Systems

Week 10

Neural Networks

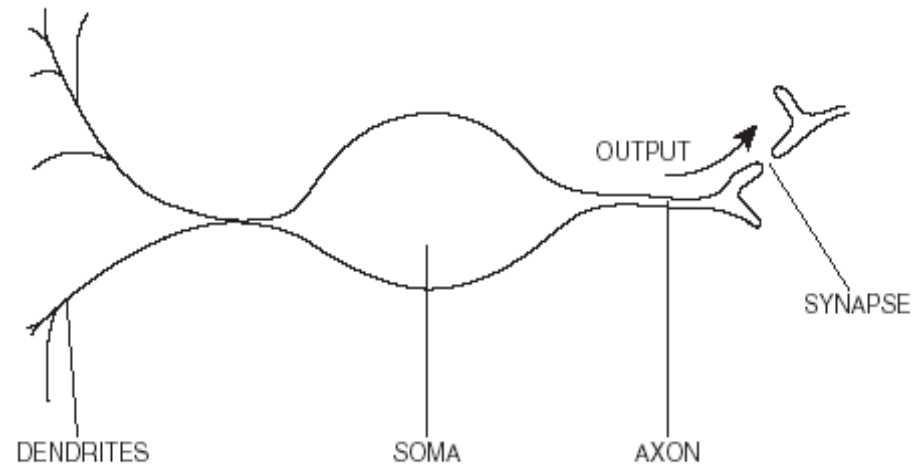
T.J. Borrelli

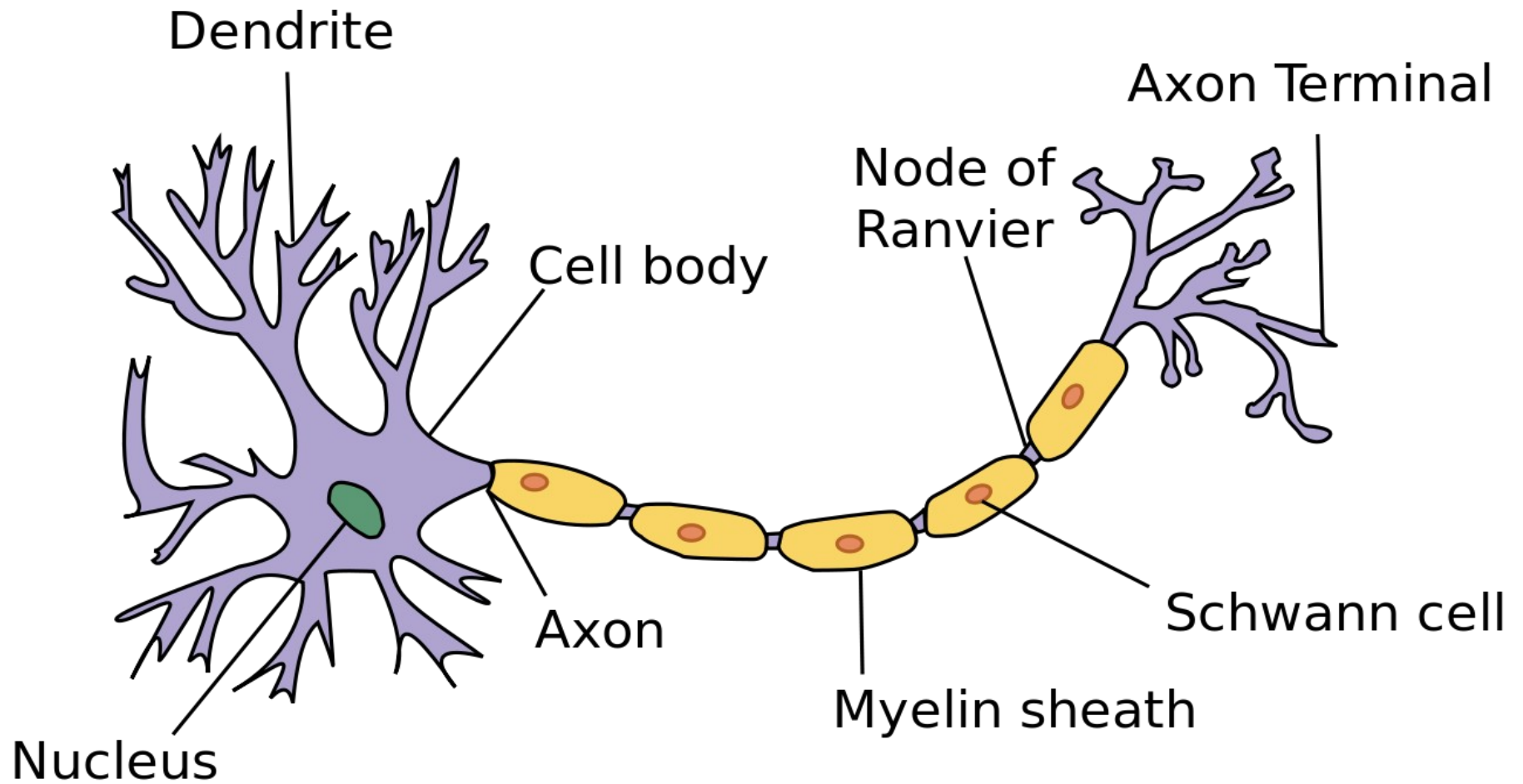
Neural Networks

- A neural network is a network of artificial neurons, which is based on the operation of the human brain
- Neural networks usually have their nodes arranged in layers
- One layer is the input layer and another is the output layer
- There are one or more hidden layers between these two

Biological Neurons

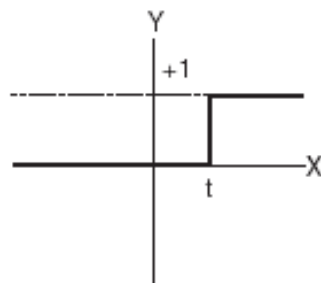
- The human brain is made up of billions of simple processing units called neurons
- Inputs are received on dendrites, and if the input levels are over a threshold, the neuron fires, passing a signal through the axon to the synapse which then connects to another neuron



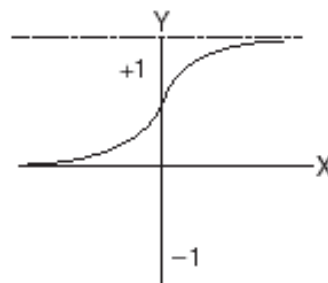


Artificial Neurons

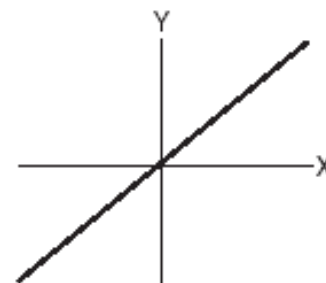
- Each neuron in the network receives one or more inputs
- An activation function is applied to the inputs which determines the output of the neuron (the activation level)
 - Three typical activation functions:



(a) Step function



(b) Sigmoid function



(c) Linear function

Artificial Neurons

- A step activation function works as follows:

$$X = \sum_{i=1}^n w_i x_i \quad Y = \begin{cases} +1 & \text{for } X > t \\ 0 & \text{for } X \leq t \end{cases}$$

- Each node i has a weight, w_i associated with it
 - The input to node i is x_i
 - t is the threshold
- If the weighted sum of the inputs to the neuron is above the threshold, the neuron will fire

Artificial Neurons

Example: Consider a simple neuron that has just two inputs. Each input has a weight associated with it:

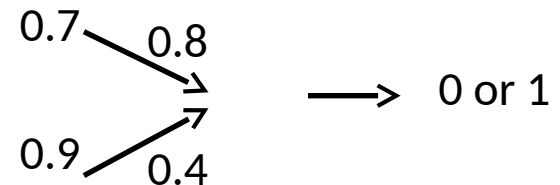
$$w_1 = 0.8 \text{ and } w_2 = 0.4$$

The inputs to the neurons are:

$$x_1 = 0.7 \text{ and } x_2 = 0.9$$

The summed weight of these inputs is:

$$(0.8 \times 0.7) + (0.4 \times 0.9) = 0.92$$



Inhibited and strengthened connections

X_1	X_2	$X_1 * X_2$ (Output)
+	+	+
+	-	-
-	+	-
-	-	+

Artificial Neurons

The activation level Y , is defined for this neuron as:

$$Y = +1 \text{ for } X > t$$
$$0 \text{ for } X \leq t$$

Hence, if $t < 0.92$, then this neuron will fire (output = 1) with this particular set of inputs. Otherwise it will not fire (output = 0).

For a neural network to learn, the weights associated with each connection can be **changed** in response to particular sets of inputs and events.

Perceptrons

- A perceptron (Rosenblatt, 1958) is a single neuron that classifies a set of inputs into one of two categories (usually $\{1, -1\}$, or $\{1, 0\}$)
- The perceptron usually uses a step function, which returns 1 if the weighted sum of inputs exceeds a threshold, and -1 (or 0) otherwise
- A perceptron can learn to represent a Boolean operator, such as AND or OR, where the result is classified as *true* if the output is 1 and *false* if the output is 0

Perceptrons

- The perceptron is trained as follows:
 - First, inputs are given random weights (usually between -0.5 and 0.5)
 - A sample training data is presented. If the perceptron misclassifies it, the weights are modified according to the following formula:
$$W_i \leftarrow W_i + (\alpha \times x_i \times e)$$
 - where α is the learning rate (between 0 and 1) and e is the size of the error

Perceptrons

- In other words, if the perceptron incorrectly classifies a positive piece of training data as negative, then the weights need to be modified to increase the output for that set of inputs
 - Error = expected - actual
- This can be done by adding a *positive* value to the weight of an input that had a *negative* output value, and vice versa
 - $W_i \leftarrow W_i + (\alpha \times x_i \times e)$
 - e is 0 if the output is correct, otherwise it is positive if the output is too low and negative if the output is too high
 - This is known as the ***perceptron training rule***

Perceptrons

- Once this modification to the weights has taken place, the next piece of training data is used in the same way
- Once all of the training data have been applied, the process starts again, until all of the weights are correct and all errors are 0
- Each iteration of this process is known as an *epoch*

Perceptrons

Example: Use the perceptron training rule to show how a perceptron can learn the logical OR function. Use a threshold of 0 and a learning rate of $\alpha = 0.2$.

First, the weight associated with each of the two inputs is initialized to a random value between -1 and 1:

$$w_1 = -0.2 \text{ and } w_2 = 0.4$$

Next, the first epoch is run. The training data will consist of the four combinations of 1's and 0's possible with the 2 inputs.

Hence, the first piece of training data is:

$$x_1 = 0 \text{ and } x_2 = 0$$

The expected output is $x_1 \vee x_2 = 0$

Perceptrons

The output is:

$$Y = \text{Step}((-0.2 \times 0) + (0.4 \times 0)) = 0$$

Hence the output Y is as expected and the error is 0. So the weights do not change.

Now for $x_1 = 0$ and $x_2 = 1$ the expected output is 1.

$$Y = \text{Step}((-0.2 \times 0) + (0.4 \times 1)) = \text{Step}(0.4) = 1$$

Again, this is correct, and so the weights do not change.

For $x_1 = 1$ and $x_2 = 0$ the expected output is also 1.

$$Y = \text{Step}((-0.2 \times 1) + (0.4 \times 0)) = \text{Step}(-0.2) = 0$$

This is incorrect, so the weights must be adjusted.

Perceptrons

Use the perceptron training rule $W_i \leftarrow W_i + (\alpha \times x_i \times e)$ to assign new values to the weights.

The error is 1 and the learning rate is 0.2, so assign the following value to w_1 :

$$w_1 = -0.2 + (0.2 \times 1 \times 1) = -0.2 + 0.2 = 0$$

Use the same formula to assign a new value to w_2 :

$$w_2 = 0.4 + (0.2 \times 0 \times 1) = 0.4$$

Because w_2 did not contribute to this error, it is not adjusted.

Perceptrons

The final piece of training data is now used, $x_1 = 1$ and $x_2 = 1$ and the new weights:

$$Y = \text{Step}((0 \times 1) + (0.4 \times 1)) = \text{Step}(0.4) = 1$$

This is correct, so the weights are not adjusted.

This is the end of the first epoch, and at this point the method runs again and continues to repeat until all four pieces of training data are classified correctly.

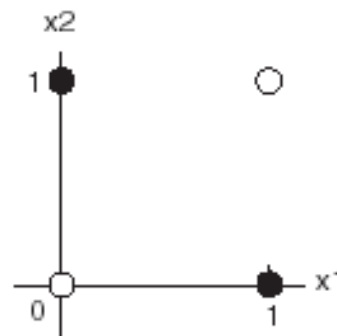
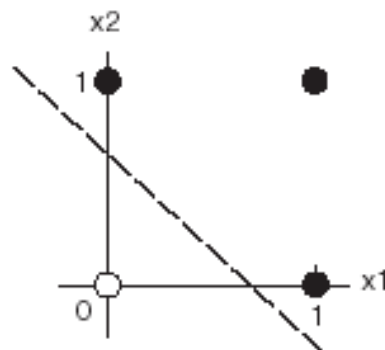
Perceptrons

This table shows the complete sequence – it takes just 3 epochs for the perceptron to correctly learn to classify input values. Lines in which an error was made are highlighted.

Epoch	X1	X2	Expected Y	Actual Y	Error	w1	w2
1	0	0	0	0	0	-0.2	0.4
1	0	1	1	1	0	-0.2	0.4
1	1	0	1	0	1	0	0.4
1	1	1	1	1	0	0	0.4
2	0	0	0	0	0	0	0.4
2	0	1	1	1	0	0	0.4
2	1	0	1	0	1	0.2	0.4
2	1	1	1	1	0	0.2	0.4
3	0	0	0	0	0	0.2	0.4
3	0	1	1	1	0	0.2	0.4
3	1	0	1	1	0	0.2	0.4
3	1	1	1	1	0	0.2	0.4

Perceptrons

- Perceptrons can only classify linearly separable functions
- The left graph shows a linearly separable function (OR)
- The right graph shows a non-linearly separable function (X-OR)



Perceptrons

- The reason that a single perceptron can only model functions that are linearly separable can be seen by examining the following functions:

$$X = \sum_{i=1}^n w_i x_i \quad Y = \begin{cases} +1 & \text{for } X > t \\ 0 & \text{for } X \leq t \end{cases}$$

- Using these functions, we are effectively dividing the search space using a *line for which* $X = t$. Hence in a perceptron with two inputs, the line that divides one class from the other is defined as:

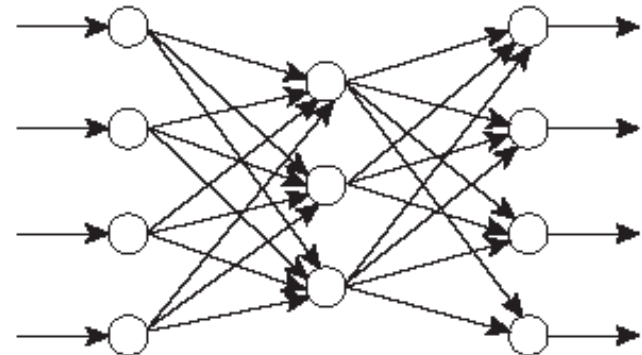
$$w_1 x_1 + w_2 x_2 = t$$

- The perceptron works by identifying a set of values for w_i which generates a suitable linear function. In cases where a linear function does not exist, the perceptron cannot succeed.

Multilayer Neural Networks

- Multilayer neural networks can classify a range of functions, including non-linearly separable ones
- Each input layer neuron connects to all neurons in the hidden layer
- The neurons in the hidden layer connect to all neurons in the output layer

A feed-forward network



Backpropagation

- Multilayer neural networks learn in the same way as perceptrons
- However, there are many more weights, and it is important to assign credit (or blame) correctly when changing weights
- Backpropagation networks use the *sigmoid* activation function, as it is easy to differentiate

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

Backpropagation

- After values are fed forward through the network, errors are fed back to modify the weights in order to train the network
- For each node, we calculate an error gradient
- For a node k in the output layer, the error e_k is the difference between the desired output and the actual output

Backpropagation

For node j , X_j is the input,

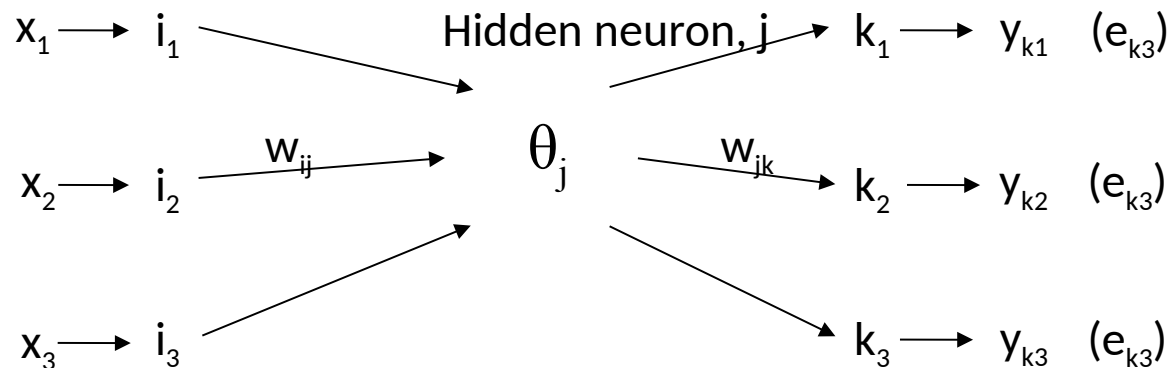
$$X_j = \sum_{i=1}^n w_{ij} - \theta_j$$

N is the number of inputs to j

θ_j is the threshold for node j

$$Y_j = \frac{1}{1 + e^{-X_j}}$$

Y_j is the output



Backpropagation

The error gradient for k is: $\delta_k = y_k \cdot (1 - y_k) \cdot e_k$

Similarly, for a node j in the

hidden layer: $\delta_j = y_j \cdot (1 - y_j) \sum_{k=1 \text{ to } n} w_{jk} \delta_k$

Now the weights are updated as follows:

$$W_{ij} \leftarrow W_{ij} + (\alpha \times x_i \times \delta_j)$$

$$W_{jk} \leftarrow W_{jk} + (\alpha \times y_j \times \delta_k)$$

α is the learning rate, (a positive number below 1)

Recurrent Networks

- Feed forward networks do not have memory
- Recurrent networks can have connections between nodes in any layer, which enables them to store data, i.e., have a memory
- Recurrent networks can be used to solve problems where the solution depends on previous inputs as well as current inputs (e.g. predicting stock market movements)

Evolving Neural Networks

- Neural networks can be susceptible to local maxima
- Evolutionary methods (genetic algorithms) can be used to determine the starting weights for a neural network, thus avoiding these kinds of problems
- Genetic algorithms can also be used to determine the connections between nodes, and how many input nodes should be used

Hebbian Learning

- Hebb's theory of learning is based on the observation that in biological systems when one neuron contributes to the firing of another neuron, the connection between the two is strengthened
- We can show how a network can use Hebbian learning to transfer its response from a primary or unconditioned stimulus to a conditioned stimulus
- This allows us to model the type of learning studied in Pavlov's experiments

Inhibited and strengthened connections

X_1	X_2	$X_1 * X_2$ (Output)
+	+	+
+	-	-
-	+	-
-	-	+

331 – Intro to Intelligent Systems

Week 11

Fuzzy Logic

T.J. Borrelli

Bivariate logic

- Bivalent/bivariate (Aristotelian) logic uses two logical values – true and false
- Two major assumptions:
 - For any element and a set, the element is either a member of the set or it is a member of the complement of that set
 - The law of the excluded middle: an element cannot belong to both a set and also its complement
- Fuzzy logic disregards these assumptions

Fuzzy Reasoning

- Multivalent logics use many logical values – often in a range of real numbers from 0 to 1
- It is important to note the difference between multivalent logic and probability
 - $P(A) = 0.5$ means that A may be true or may be false with a probability of 0.5
 - A logical value of 0.5 means both true and false at the same time (50% true and 50% false)

Why Fuzzy Logic

- Although probability theory is good for measuring randomness of information, it is not good for measuring the *meaning* of information
- Much of the confusion around the use of English (natural) language is related to lack of clarity (vagueness) rather than randomness
- Fuzzy logic then presents a *possibility theory* as a measure of this vagueness as probability theory is measure of randomness

Fuzzy Reasoning

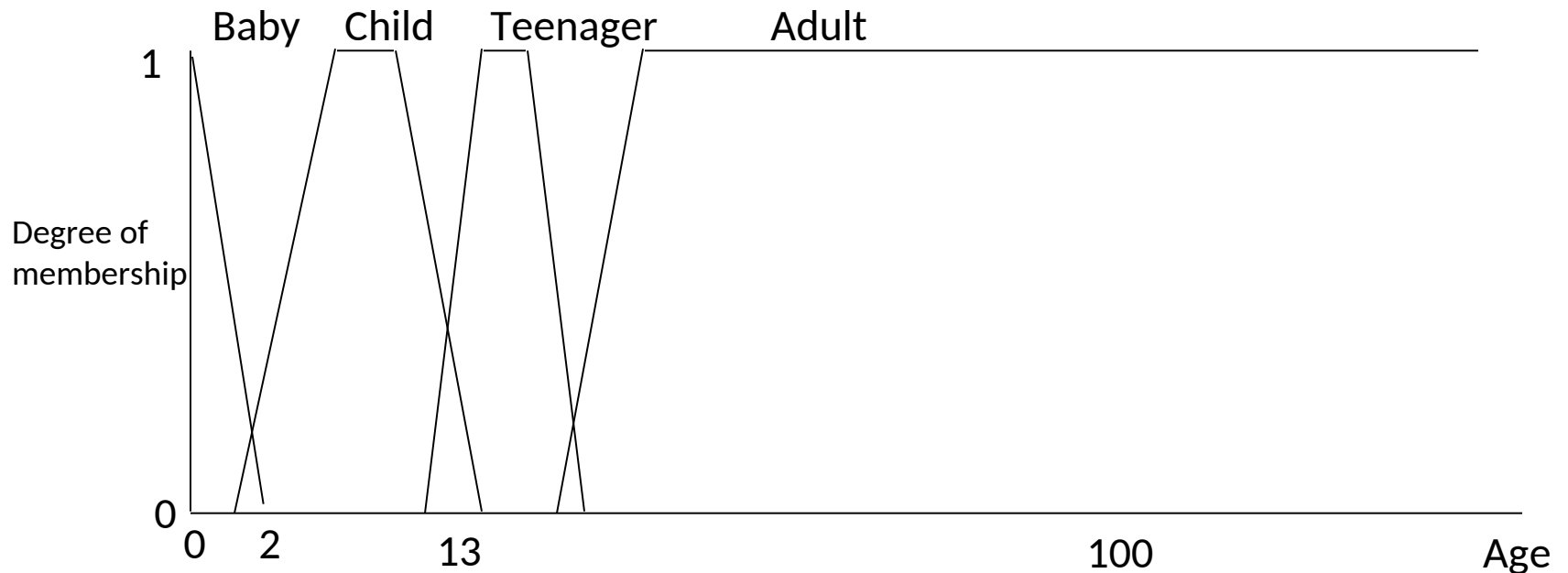
- Variables used in fuzzy systems can express qualities such as height, which can take values such as “tall”, “short” or “very tall”
- These values define subsets of the universe of discourse

Fuzzy Sets

- A *crisp* set is a set for which each value either is or is not contained in the set
- A *fuzzy* set is a set for which each value has a membership value, and so is a member to some extent
- The membership value defines the extent to which a variable is a member of a fuzzy set
- The membership value ranges from 0 (not at all a member of the set) to 1 (absolutely a member of the set)

Fuzzy Set Membership Example

Four sets: Baby, Child, Teenager, and Adult



Note that at age 13 a person might be defined as both a child and a teenager.

Fuzzy Set Membership Functions

We can define a membership function for the fuzzy set Baby (B) and the fuzzy set Child (C) as follows:

$$M_B(x) = 1 - (x/2) \text{ for } x < 2, 0 \text{ for } x \geq 2$$

$$M_C(x) = (x-1)/6 \text{ for } x \geq 1 \text{ and } x \leq 7, 1 \text{ for } x > 7 \text{ and } x \leq 8, \\ (14 - x)/6 \text{ for } x > 8 \text{ and } x \leq 14, 0 \text{ for } x > 14 \text{ or } x < 1$$

Similarly, we could define membership functions for fuzzy sets Teenager (T) and Adult (A). Note that there is nothing special about these functions – they have been chosen entirely arbitrarily and reflect a subjective view.

Fuzzy Set Membership Functions

To represent a fuzzy set in a computer, we use a list of pairs, where each pair represents a number and the fuzzy membership value for that number:

$$A = \{ (x_1, M_A(x_1)), \dots, (x_n, M_A(x_n)) \}$$

For example, we could define B, the fuzzy set of Babies as follows:

$$B = \{ (0, 1), (2, 0) \}$$

This can also be thought of as representing the x and y coordinates of two points on the line that represents the set membership function. Similarly, we could define the fuzzy set of Children, C, as follows:

$$C = \{ (1, 0), (7, 1), (8, 1), (14, 0) \}$$

Crisp Set Operators

- Not A – the complement of A , which contains the elements which are not contained in A
- $A \cap B$ – the intersection of A and B , which contains those elements which are contained in both A and B
- $A \cup B$ – the union of A and B which contains all the elements of A and all the elements of B
- Fuzzy sets use the same operators, but the operators have different meanings

Fuzzy Set Operators

- Fuzzy set operators can be defined by their membership functions:

$$M_{\neg A}(x) = 1 - M_A(x) \quad (\text{complement})$$

$$M_{A \cap B}(x) = \text{MIN}(M_A(x), M_B(x)) \quad (\text{intersection})$$

$$M_{A \cup B}(x) = \text{MAX}(M_A(x), M_B(x)) \quad (\text{union})$$

- We can also define containment (subset operator): $B \subseteq A$ iff $\forall x (M_B(x) \leq M_A(x))$

Fuzzy Set Operators

For example, The set of not-Babies, $\neg B$, is defined as follows:

$$\neg B = \{ (0, 0), (2, 1) \}$$

Similarly, we can define $\neg C = \{ (1, 1), (7, 0), (8, 0), (14, 1) \}$.

To determine the intersection of B and C (Babies and Children), we need to have the sets defined over the same values.

Hence, we must *augment* sets B and C:

$$B = \{ (0, 1), (1, 0.5), (2, 0), (7, 0), (8, 0), (14, 0) \}$$

$$C = \{ (0, 0), (1, 0), (2, 0.166), (7, 1), (8, 1), (14, 0) \}$$

Now find the intersection by using $M_{A \cap B}(x) = \text{MIN}(M_A(x), M_B(x))$:

$$B \cap C = \{ (0, 0), (1, 0), (2, 0), (7, 0), (8, 0), (14, 0) \}$$

Fuzzy Set Operators

But this has not worked! We need to define the set using values that will correctly define the ranges. In other words, define the intersection as follows:

$$B \cap C = \{ (1, 0), (1.75, 0.125), (2, 0) \}$$

where 1.75 was used as the value for x . This was determined by calculating the value of x for which $M_B(x) = M_C(x)$. If a person is in the set $B \cap C$, then she is *both* a Baby *and* a Child.

The union of the fuzzy sets of babies and children is as follows:

$$B \cup C = \{ (0, 1), (1.75, 0.125), (7, 1), (8, 1), (14, 0) \}$$

A person who belongs to the set $B \cup C$ is *either* a Baby *or* a Child.

Fuzzy Set Operators

In traditional set theory, if set A contains set B, then all the elements of set B are also elements of set A. In other words, the union $A \cup B = A$ and the intersection $A \cap B = B$. In this case, B is said to be a subset of A, which is written $B \subset A$.

For example, consider a fuzzy set R which is the fuzzy set of retirees. We will define this set by the following membership function:

$$M_R(x) = 0 \text{ for } x \leq 55, (x - 55)/45 \text{ for } x > 55 \text{ and } x \leq 100, 0 \text{ for } x > 100$$

Suppose the universe of people range in age from 0 to 100. Then R is a subset of Adult. In fuzzy terms, B is a fuzzy subset of A if B's membership function is always less than (or equal to) the membership function for A, when defined over the same range.

Fuzzy Logic

- Fuzzy logic is a *non-monotonic* logical system that applies to fuzzy variables
 - Non-monotonic: if a new fuzzy fact is added to the database, this fact may contradict conclusions that were already derived
- Each fuzzy variable can take a value from 0 (not at all true) to 1 (entirely true) but can also take on real values in between
 - 0.5 might indicate “as true as it is false”

Fuzzy Logic

- We use logical connectives defined as:

$$A \vee B \equiv \text{MAX} (A, B)$$

$$A \wedge B \equiv \text{MIN} (A, B)$$

$$\neg A \equiv 1 - A$$

Note that if $A = 0.5$, then $A = \neg A$ (?)

We cannot write a complete truth table for a fuzzy logical connective because it would have an infinite number of entries.

Consider a *trivalent* logical system with three logical values: $\{0, 0.5, 1\}$:

A	B	$A \vee B$
0	0	0
0	0.5	0.5
0	1	1
0.5	0	0.5
0.5	0.5	0.5
0.5	1	1
1	0	1
1	0.5	1
1	1	1

Fuzzy Logic

- Recall:
 $A \rightarrow B \equiv \neg A \vee B$
- We might define fuzzy implication as:
 $A \rightarrow B \equiv \text{MAX}((1 - A), B)$
- This gives the unintuitive truth table shown on the right
- $0.5 \rightarrow 0 = 0.5$, where we would expect $0.5 \rightarrow 0 = 0$

A	B	$A \rightarrow B$
0	0	1
0	0.5	1
0	1	1
0.5	0	0.5
0.5	0.5	0.5
0.5	1	1
1	0	0
1	0.5	0.5
1	1	1

Fuzzy Logic

- An alternative is Gödel implication, which is defined as:
 $A \rightarrow B \equiv (A \leq B) \vee B$
- This gives a more intuitive truth table

A	B	$A \rightarrow B$
0	0	1
0	0.5	1
0	1	1
0.5	0	0
0.5	0.5	1
0.5	1	1
1	0	0
1	0.5	0.5
1	1	1

Fuzzy Rules

- A fuzzy rule takes the following form:

if A op x then B = y

where op is an operator such as >, < or ==

- For example:

IF temperature > 50 then fan speed = fast

IF height == tall then trouser length = long

IF study time == short then grades = poor

Fuzzy Inference

- Inference in fuzzy expert systems uses *Mamdani inference*
- Mamdani inference derives a single crisp value (a recommendation) by applying fuzzy rules to a set of crisp input values (e.g., from a set of sensors, or a human, etc.):

Step 1: Fuzzify the inputs.

Step 2: Apply the inputs to the antecedents of the fuzzy rules to obtain a set of fuzzy outputs.

Step 3: Convert the fuzzy outputs to a single crisp value using defuzzification.

Fuzzy Expert Systems

- A fuzzy expert system is built by creating a set of fuzzy rules and applying fuzzy inference
- In many ways this is more appropriate than standard expert systems since expert knowledge is not usually black and white but has elements of gray
- The first stage in building a fuzzy expert system is choosing suitable linguistic variables
- Rules are then generated based on the expert's knowledge using the linguistic variables

Fuzzy Expert Systems

Suppose you are designing an anti-lock braking system for a car, which is designed to cope when the roads are icy and the wheels may lock. The rules for the system might be as follows:

Rule 1: IF pressure on brake is medium
THEN apply the brake

Rule 2: IF pressure on brake is high AND car speed is fast
AND wheel speed is fast THEN apply the brake

Rule 3: IF pressure on brake is high AND car speed is fast
AND wheel speed is slow THEN release the brake

Rule 4: IF pressure on brake is low
THEN release the brake

Fuzzy Expert Systems

The first step is to define the fuzzy sets (the linguistic variables):

- Brake pressure (P)
- Wheel speed (W)
- Car speed (C)

The range of the fuzzy membership values are as follows:

Pressure:

$$\text{Low (L)} = \{(0, 1), (50, 0)\}$$

$$\text{Medium (M)} = \{(30, 0), (50, 1), (70, 0)\}$$

$$\text{High (H)} = \{(50, 0), (100, 1)\}$$

Wheel speed and Car speed:

$$\text{Slow (S)} = \{(0, 1), (60, 0)\}$$

$$\text{Medium (M)} = \{(20, 0), (50, 1), (80, 0)\}$$

$$\text{Fast (F)} = \{(40, 0), (100, 1)\}$$

Fuzzy Expert Systems

In addition, we need the output rules for apply the brake and release the brake:

Brake apply (A) = $\{(0, 0), (100, 1)\}$

Brake release (R) = $\{(0, 1), (100, 0)\}$

The membership functions for a specific input of {pressure = 60, wheel speed = 55, and car speed = 80} are as follows:

$M_{PL}(60) = 0$	$M_{PM}(60) = 0.5$	$M_{PH}(60) = 0.2$
$M_{WS}(55) = 0.083$	$M_{WM}(55) = 0.833$	$M_{WF}(55) = 0.25$
$M_{CS}(80) = 0$	$M_{CM}(80) = 0$	$M_{CF}(80) = 0.667$

Next, we need to apply these fuzzy values to the antecedent's of the system's rules.

Fuzzy Expert Systems

Rule 1: IF pressure on brake is medium
THEN apply the brake

Rule 1, taken on its own, tells us that the degree to which we should apply the brake is the same as the degree to which the pressure on the brake pedal can be described as “medium.”

Since $M_{PM}(60) = 0.5$, Rule 1 gives us a value of 0.5 for the instruction “Apply the brake.”

Fuzzy Expert Systems

Rule 2: IF pressure on brake is high AND car speed is fast
AND wheel speed is fast THEN apply the brake

$$M_{PH}(60) = 0.2, M_{CF}(80) = 0.667, M_{WF}(55) = 0.25$$

The conjunction of two or more fuzzy variables is the minimum of the membership values, hence the antecedent for Rule 2 has the value of 0.2. Thus Rule 2 gives us a fuzzy value of 0.2 for the instruction “Apply the brake.”

Fuzzy Expert Systems

Rule 3: IF pressure on brake is high AND car speed is fast AND wheel speed is slow THEN release the brake

$$M_{PH}(60) = 0.2, M_{CF}(80) = 0.667, M_{WS}(55) = 0.083$$

This gives us a value of 0.083 for “Release the brake.”

Fuzzy Expert Systems

Rule 4: IF pressure on brake is low

THEN release the brake

$M_{PL}(60) = 0$. This gives us a value of 0 for “Release the brake.”

Now we have four fuzzy values: 0.5 and 0.2 for “Apply the brake” and 0.083 and 0 for “Release the brake.”

Fuzzy Expert Systems

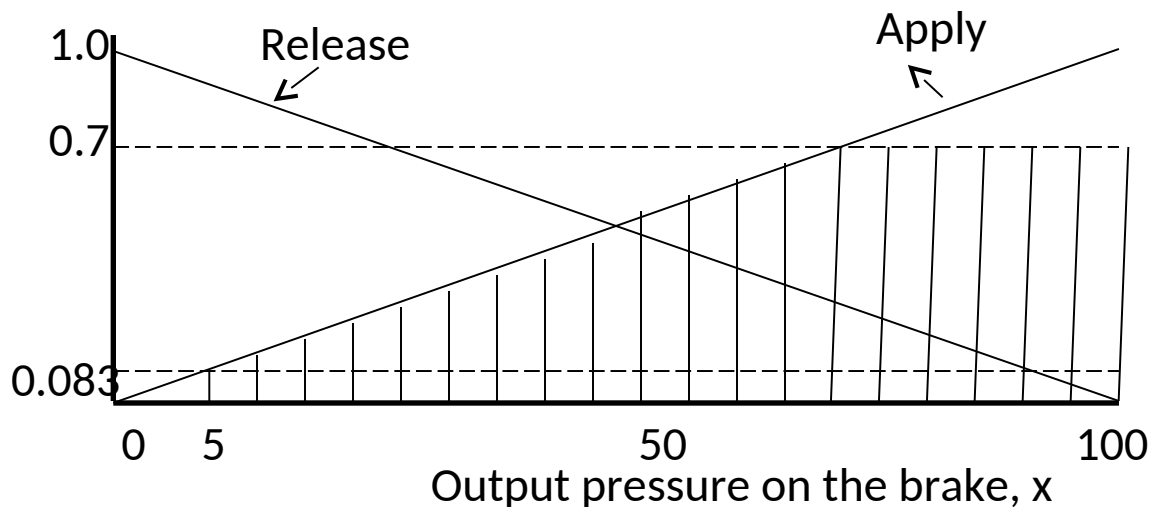
Now we need to combine these values together. We could sum the values, or take the minimum, or take the maximum. The appropriate combination will depend on the nature of the problem being solved. In this case it makes sense to sum the values because the separate rules are giving different reasons for applying or releasing the brakes, and those reasons should combine together cumulatively.

Hence we end up with a value of 0.7 for “Apply the break” and 0.083 for “Release the brake.”

Fuzzy Expert Systems

To use this fuzzy output, a crisp output value must now be determined from the fuzzy values. The process of obtaining a crisp value from a set of fuzzy variables is known as *de-fuzzification*. This can be done by obtaining the center of gravity, COG, of the “clipped” membership functions (hashed area in the figure below shows the combined fuzzy output of the four rules).

$$\text{COG} = (\sum x M(x)) / (\sum M(x))$$



Clip membership in *Apply* to 0.7 and membership in *Release* to 0.083. At each sample point x , find $\max(M_{\text{Apply}}(x), M_{\text{Release}}(x))$ and calculate COG.

Fuzzy Expert Systems

For our example,

$$\begin{aligned}\text{COG} &\cong \frac{(5 \times 0.083) + (10 \times 0.1) + (15 \times 0.15) + \dots + (70 \times 0.7) + \dots + (100 \times 0.7)}{0.083 + 0.1 + 0.15 + \dots + 0.7 + \dots + 0.7} \\ &\cong \frac{621.415}{9.483} \\ &\cong 65.53\end{aligned}$$

Hence, the crisp output value for this system is 65.53, which can be translated into the value of 65.53 units of pressure applied by the brake to the wheel of the car.