CS5201 Intelligent Systems

Paul Ahern

May 31, 2007

Abstract

Notes from lectures.

Contents

1 Key Facts to Know

1.1 Exam

Short answers preferred. Cover key terms and concepts.

1. Know Genetic Algorithm steps.
2. Know how to establish mean and confidence intervals for expected error of hypothesis (Mitchell, chapter 5)
3. Know Bayes Rule:
$$p(A \mid B) = \frac{p(B \mid A) \times p(A)}{p(B)}$$
4. Fuzzy Logic definitely coming up.

2 Lecture 3 October 2006

40% of course marks will be awarded on the basis of the TAC SCM assignment. Team effort to build a competitive agent.

3 Lecture 5 October 2006

Negnevitsky Chapter 1.

3.1 Background

- Intelligence is the ability to learn and understand, to solve problems and to make decisions.
- Artificial Intelligence is a science that has defined its goal as making machines do things that would require intelligence if done by humans.

- A machine is thought intelligent if it can achieve human-level performance in some cognitive task. To build an intelligent machine, we have to capture, organize and use human expert knowledge in some problem area.
- The realization that the problem domain for intelligent machines had to be sufficiently restricted marked a major paradigm shift in AI from general-purpose, knowledge-sparse, weak methods to domain-specific, knowledge-intensive methods. This led to the development of expert systems - computer programs capable of performing at a human-expert level in a narrow problem area. Expert systems use human knowledge and expertise in the form of specific rules, and are distinguished by the clean separation of the knowledge and the reasoning mechanism. They can also explain their reasoning procedures.
- One of the main difficulties in building intelligent machines, or knowledge engineering, is the *knowledge acquisition bottleneck* - extracting knowledge from human experts.
- Experts think in imprecise terms such as very often and almost never, usually and hardly ever, frequently and occasionally, and use linguistic variables such as high and low, fast and slow, heavy and light. Fuzzy logic or fuzzy set theory provides a means to compute with words. It concentrates on the use of fuzzy values that capture the meaning of words, human reasoning and decision making, and provides a way of breaking through the computational burden of traditional expert systems.
- Expert systems can neither learn nor improve themselves through experience. They are individually created and demand large efforts for their development. It can take from five to ten person-years to build even a moderate expert system. Machine learning can accelerate this process significantly and enhance the quality of knowledge by adding new rules or changing incorrect ones.
- Artificial neural networks, inspired by biological neural networks, learn from historical cases and make it possible to generate rules automatically and thus avoid the tedious and expensive processes of knowledge acquisition, validation and revision.
- Integration of expert systems and ANNs, and fuzzy logic and ANNs improve the adaptability, fault tolerance and speed of knowledge-based systems.

4 Lecture 10 October 2006

Negnevitsky Chapter 2.

4.1 Rules-based Expert Systems

- Knowledge is a theoretical or practical understanding of a subject. Knowledge is the sum of what is currently known.
- An expert is a person who has deep knowledge in the form of facts and rules and strong practical experience in a particular domain. An expert can do things other people cannot.
- The experts can usually express their knowledge in the form of production rules.
- Production rules are represented as IF <antecedent> THEN <consequent> statements. A production rule is the most popular type of knowledge representation. Rules can express relations, recommendations, directives, strategies and heuristics.
- A computer program capable of performing at a human-expert level in a narrow problem domain area is called an expert system. The most popular expert systems are rule-based expert systems.

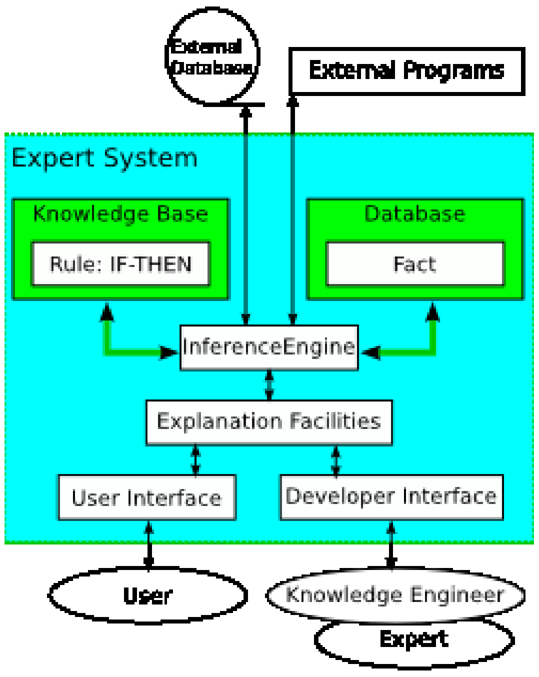


Figure 1: Newell and Simon, Production System Model

- In developing rule-based experts systems, shells are becoming particularly common. An experts system shell is a skeleton expert system with the knowledge removed. To build a new expert system application, all the user has to do is to add the knowledge in the form of rules and provide relevant data.

- A rule-based expert system has five basic components:
  1. *Knowledge Base* contains the domain knowledge represented as a set of rules.
  2. *Database* contains a set of facts used to match against the IF parts of rules.
  3. *Inference Engine* links the rules with the facts and carries out the reasoning whereby the expert system reaches a solution.
  4. *Explanation Facilities* enable the user to query the expert system about why a particular conclusion is reached and why a specific fact is needed.
  5. *User Interface* is the means of communication between a user and the expert system.

- Expert systems separate knowledge from its processing by splitting up the knowledge base and the inference engine. This makes the task of building and maintaining an expert system much easier.

Knowledge Engineer uses a shell (e.g. VP Expert) and populates the Knowledge Base with information from the expert(s).

Reasoning Engine is made up of:

- Inference Engine
- Explanation Facility

- User Interface

The Rules can express: Relations, Recommendations, Directives, Strategy or Heuristics. System should be deterministic.

5 Lecture 12 October 2006

Negnevitsky Chapters 2-3.

5.1 Inference

- There are two principal methods to direct search and reasoning: forward chaining and backward chaining inference techniques. Forward chaining is data-driven reasoning; It starts from the known data and proceeds forward until no further rules can be fired. Backward chaining is goal-driven reasoning; An expert system has a hypothetical solution (the goal), and the inference engine attempts to find the evidence to prove it.
- If more than one rule can be fired in a given cycle, the inference engine must decide which rule to fire. A method for deciding is called conflict resolution. Some methods:
  - Fire the rule with the highest priority.
  - Fire the most specific rule.
  - Fire the rule which uses the data most recently entered in the database.
- Rule-based experts systems have the advantages of natural knowledge representation, uniform structure, separation of knowledge from its processing, and coping with incomplete and uncertain knowledge.
- Rule-based expert systems also have disadvantages, especially opaque relations between rules, ineffective search strategy and inability to learn (humans required to update the rules).

How do we choose between backward or forward inference?

Answer: Study how a domain expert solves a problem. If a domain expert needs to gather information and then tries to infer, then forward inference is more likely to be suitable. If the expert hypothesizes about solutions, then use backward inference.

We can combine inference methods, but the basic inference mechanism is usually backward.

5.2 Uncertainty

Uncertainty is defined as the lack of the exact knowledge that would enable us to reach a perfectly reliable conclusion. Classical logic permits only exact reasoning. It assumes that perfect knowledge always exists and the law of the excluded middle can always be applied. $\neg$TRUE = FALSE.

Source of uncertain knowledge:

1. *Weak implications.* Domain experts and knowledge engineers must establish concrete correlations between IF and THEN parts of the rules. Therefore, expert systems need to have the ability to handle vague associations, for example by accepting the degree of correlations as numerical certainty factors.
2. *Imprecise Language.* Our natural language is ambiguous and imprecise. We describe facts with such terms as often and sometimes, frequently and hardly ever. As a result, it can be difficult to express knowledge in the precise IF-THEN form of production rules. Expert systems need quantified measures.
3. *Unknown Data.* When the data is incomplete or missing, the only solution is to accept the value *unknown* and proceed to an approximate reasoning with this value.
4. *Disagreement among experts.* Weighting associated to different expert opinions.

6 Lecture 17 October 2006

Negnevitsky Chapter 3.

6.1 Probability Theory

When examining uncertainty, we adopt probability as a model to predict future events.

$$P(success) = p = \frac{number\,of\,successes}{number\,of\,possible\,outcomes} = \frac{s}{s+f}$$

And likewise for failures, q . Note that $p + q = 1$. Now let A be some event and B be some other event. These are not mutually exclusive. The conditional probability that event A will occur, *given* that B has occurred is $P(A \mid B)$.

$$P(A \mid B) = \frac{number\,of\,times\,both\,A\,and\,B\,can\,occur}{number\,of\,times\,B\,can\,occur}$$

The probability of A and B both occurring, denoted $P(A \cap B)$, is the joint probability.

$$P(A \cap B) = p(A \mid B) \times p(B)$$

and is commutative (i.e. $P(A \cap B) = P(B \cap A)$. This allows us to derive the famous Bayesian Rule:

$$p(A \mid B) = \frac{p(B \mid A) \times p(A)}{p(B)}$$

If A is conditionally dependent on n other mutually exclusive events then:

$$p(A) = \sum_{i=1}^{n} p(A \mid B_i) \times p(B_i)$$

We shall now consider the case where A depends on two mutually exclusive events, B and $\neg B$. From the previous equation

$$p(B) = (p(B \mid A) \times p(A)) + (p(B \mid \neg A) \times p(\neg A))$$

and substituting this into Bayesian Rule gives

$$p(A \mid B) = \frac{p(B \mid A) \times p(A)}{(p(B \mid A) \times p(A)) + (p(B \mid \neg A) \times p(\neg A))}$$

This equation is used in the management of uncertainty in expert systems.

6.2 Reasoning in Expert Systems

Suppose rules in a knowledge base are represented as follows:

```
IF H is true
 THEN E is true, with probability p
```

Now consider, *if event E has occurred, how do we determine the probability that H occurred?*

Answer: Above equation, replacing for A and B. In this case, H is the hypothesis and E is the evidence.

$$p(H \mid E) = \frac{p(E \mid H) \times p(H)}{(p(E \mid H) \times p(H)) + (p(E \mid \neg H) \times p(\neg H))}$$

Single evidence E and m hypotheses imply:

$$p(H_i \mid E) = \frac{p(E \mid H_i) \times p(H_i)}{\Sigma_{k=1}^{m} p(E \mid H_k) \times p(H_k)}$$

Suppose the expert, given multiple (n) evidences, cannot distinguish between m hypotheses:

$$p(H_i \mid E_1, ..., E_n) = \frac{p(E_1, ..., E_n \mid H_i) \times p(H_i)}{\Sigma_{k=1}^{m} p(E_1, ..., E_n \mid H_k) \times p(H_k)}$$

An application of this equation requires us to obtain the conditional probabilities of all possible combinations of evidences for all hypotheses! This grows exponentially. Therefore, assume conditional independence if possible.

Let the posterior probability of hypothesis H_i upon observing evidences $E_1 ... E_n$ be:

$$p(H_i \mid E_1 ... E_n) = \frac{p(E_1 \mid H_i) \times ... \times p(E_n \mid H_i) \times p(H_i)}{\Sigma_{k=1}^{m} p(E_1 \mid H_k) \times ... \times p(E_n \mid H_k) \times p(H_k)}$$

This is a far more tractable solution and assumes conditional independence among different evidences.

7 Lecture 24 October 2006

Negnevitsky Chapter 3.

7.1 Bias of the Bayesian Method

The framework for Bayesian reasoning requires probability values as primary inputs. The assessment of these values usually involves human judgment. However, psychological research shows that humans either cannot elicit probability values consistent with the Bayesian rules or do it badly. The conditional probabilities may be inconsistent with the prior probabilities given by the expert.

In the Bayesian approach, an expert is required to provide the prior probability of hypothesis H and values for the likelihood of sufficiency, LS, to measure belief in the hypothesis if evidence E is present, and the likelihood of necessity, LN, to measure disbelief in hypothesis H if the same evidence is missing. The Bayesian method uses rules in the following form:

```
IF E is true {LS,LN}
THEN H is true (prior probability}
```

To apply the Bayesian approach we must satisfy the conditional independence of evidence. We also should have reliable statistical data and define the prior probabilities for each hypothesis. As these requirements are rarely satisfied in real-world problems only a few systems have been built based on Bayesian reasoning.

7.2 Certainty Factors Theory

Certainty factors theory is a popular alternative to Bayesian reasoning. The basic principles of this theory were first introduced in MYCIN, an expert system for diagnosis and therapy of blood infections and meningitis. The developers of MYCIN found that medical experts expressed the strength of their belief in terms that were neither logical nor mathematically consistent. In addition, reliable statistical data about the problem domain was not available.

The MYCIN team decided to introduce a certainty factor (cf), a number to measure the expert's belief. The maximum value of the certainty factor was +1.0 (definitely true) and the minimum -1.0 (definitely false). For example, if the expert stated that some evidence was almost certainly true, a cf factor of 0.8 would be assigned to this evidence.

In expert systems with certainty factors the knowledge base consists of a set of rules that have the following form:

```
IF <evidence>
THEN <hypothesis> {cf}
```

where cf represents belief in hypothesis H given that evidence E has occurred.

Certainty factors are used if the probabilities are not known or cannot be easily obtained. Certainly theory can manage incrementally acquired evidence, the conjunctions and disjunction of hypotheses as well as evidences with different degrees of belief.

Certainty factors provide a simple way of updating probabilities given new evidence. They are slightly dodgy theoretically, but in practice this tends not to matter too much. This is mainly because the error in dealing with certainties tends to lie as much in the certainty factors attached to the rules (or in conditional probabilities assigned to things) as in how they are manipulated. Generally these certainty factors will be based on rough guesses of experts in the domain, rather than based on actual statistical knowledge. These guesses tend not to be very good.

Both Bayesian reasoning and certainty theory share a common problem: finding an expert able to quantify subjective and qualitative information.

8 Lecture 31 October 2006

Negnevitsky Chapter 4.

8.1 Fuzzy Logic

Fuzzy logic reflects how people think. It attempts to model our sense of words, our decision making and our common sense. As a result, it is leading to new, more human, intelligent systems.

In 1965 Lotfi Zadeh, published “Fuzzy sets”. He extended possibility theory into a formal system of mathematical logic, and introduced a new concept for applying natural language terms. This new logic for representing and manipulating fuzzy terms was called fuzzy logic. Fuzziness rests on fuzzy set theory. Fuzzy logic is a small part of that theory.

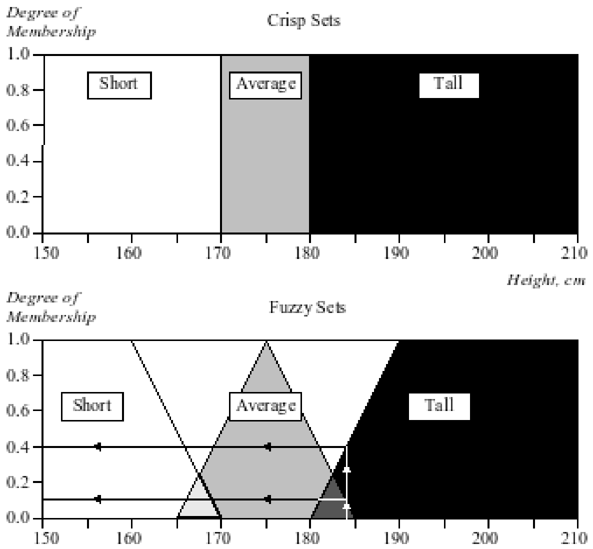


Figure 2: Crisp v Fuzzy Sets

- Fuzzy logic is multi-valued (unlike Boolean).
- It deals with degrees of membership and degrees of truth.
- Uses the continuum of logical values between 0 (completely false) and 1 (completely true).
- Things can be partly true and partly false at the same time.

- To express a continuous fuzzy set on a computer, we need to map it to a function.
- Map the elements to their degree of membership of each set.
- Typical functions include sigmoid and Gaussian.
- Linear Fit Functions are the fastest and used most in practice.

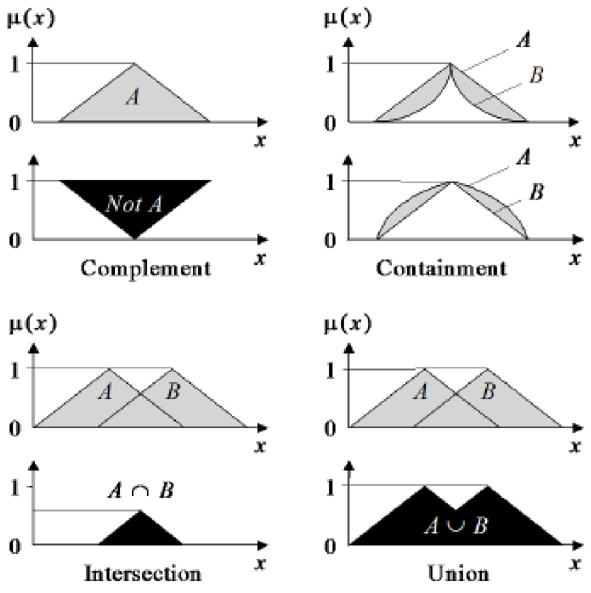


Figure 3: Fuzzy Operations

Complement $\mu_{\neg A}(x) = 1 - \mu_A(x)$.

Containment which sets belong to other sets.

Intersection $\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)] = \mu_A(x) \cap \mu_B(x)$.

Union $\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)] = \mu_A(x) \cup \mu_B(x)$, where $x \in X$ (i.e. universe of discourse X).

9 Lecture 2 November 2006

Negnevitsky Chapter 4.

9.1 Fuzzy Linguistic variables and hedges

A linguistic variable is used to describe a term or concept with vague or fuzzy values. These values are represented as fuzzy sets (e.g. Tall).

Hedges are fuzzy set qualifiers used to modify the shape of fuzzy sets. Hedges perform mathematical operations of concentration by reducing the degree of membership of fuzzy elements, dilation by increasing the degree of membership and intensification by increasing the degree of membership above 0.5 and decreasing those below 0.5.

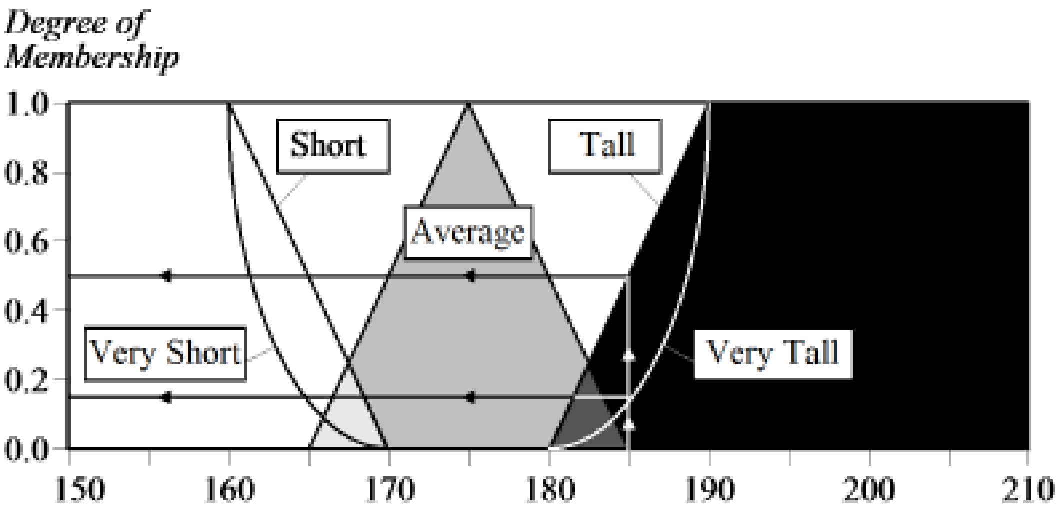
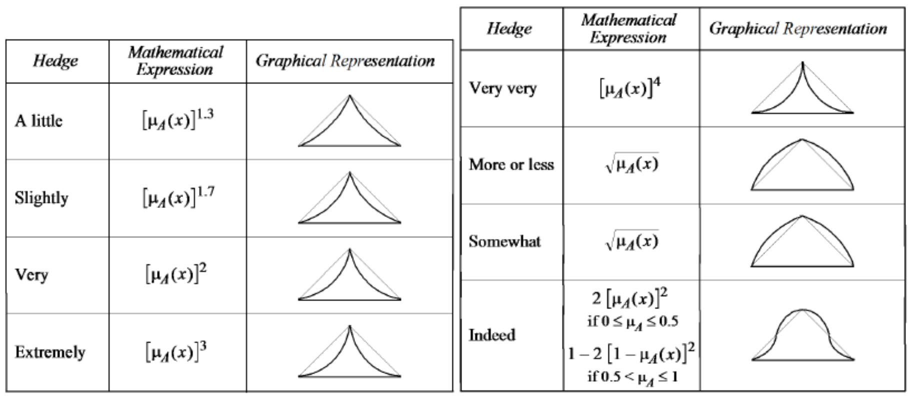


Figure 4: Examples of Hedges

Hedge formulae



Hedge	Mathematical Expression	Graphical Representation
A little	$[\mu_A(x)]^{1.3}$	
Slightly	$[\mu_A(x)]^{1.7}$	
Very	$[\mu_A(x)]^2$	
Extremely	$[\mu_A(x)]^3$	

Hedge	Mathematical Expression	Graphical Representation
Very very	$[\mu_A(x)]^4$	
More or less	$\sqrt{\mu_A(x)}$	
Somewhat	$\sqrt{\mu_A(x)}$	
Indeed	$2\,[\mu_A(x)]^2$ if $0 \le \mu_A \le 0.5$; $1 - 2\,[1 - \mu_A(x)]^2$ if $0.5 < \mu_A \le 1$	

Figure 5: Representation of Hedges

9.2 Fuzzy Reasoning

Zadeh (1973) suggested capturing human knowledge using Fuzzy Rules. A Fuzzy Rule can be defined as a conditional statement of the form:

IF x is A THEN y is B,

where x and y are linguistic variables; and A and B are linguistic values, determined by fuzzy sets on the universe of discourses X and Y , respectively. For example:

IF objectWeight is heavy THEN effortToLift is high,

uses a linguistic variable, objectWeight, can include the fuzzy sets light, medium and heavy to describe weight. Separate rules are not necessary for cut-off points. In practice, Fuzzy expert systems merge the rules and *reduce the number of rules by at least 90%*!

In a fuzzy system, all rules fire to some degree, or in other words they fire partially. If the antecedent is true to some degree of membership, then the consequent is also true to that same degree.

Monotonic Selection The value of the output or a truth membership grade of the rule consequent can be estimated directly from a corresponding truth membership grade in the antecedent. This form of fuzzy inference uses a method called monotonic selection.

9.3 Mamdani Method

The most commonly used fuzzy inference technique is the Mamdani method. Professor Ebrahim Mamdani of London University built one of the first fuzzy systems to control a steam engine and boiler combination. He applied a set of fuzzy rules supplied by experienced human operators in 1975.

Mamdani 4-step method:

1. Fuzzification of the input variables,
2. Rule evaluation,
3. Aggregation of the rule outputs,
4. Defuzzification.

9.4 Sugeno Fuzzy Inference

Mamdani-style inference, requires us to find the centroid of a two-dimensional shape by integrating across a continuously varying function. In general, this process is not computationally efficient.

Michio Sugeno suggested to use a single spike, a fuzzy singleton, as the membership function of the rule consequent. This is a fuzzy set with a membership function that is unity at a single particular point on the universe of discourse and zero everywhere else.

Sugeno-style fuzzy inference is very similar to the Mamdani method. Sugeno changed only a rule consequent. He used a mathematical function of the input variable (instead of a fuzzy set).

9.5 Mamdani versus Sugeno

Mamdani method is widely accepted for capturing expert knowledge. It allows us to describe the expertise in more intuitive, more human-like manner. However, Mamdani-type fuzzy inference entails a substantial computational burden.

Sugeno method is computationally efficient and works well with optimisation and adaptive techniques, which makes it very attractive in control problems, particularly for dynamic nonlinear systems.

10 Lecture 7 November 2006

Negnevitsky Chapter 4.

10.1 Service Centre case study

Steps:

1. Specify the problem and define linguistic variables.
   There are four: average waiting time, repair utilisation factor, number of servers, initial number of spare parts.
2. Determine fuzzy sets.
3. Elicit and construct fuzzy rules.
4. Encoding.
5. Evaluate and tune.

10.2 Tuning Fuzzy Systems

- Review model input and output variables, and if required redefine their ranges.
- Review the fuzzy sets, and if required define additional sets on the universe of discourse. The use of wide fuzzy sets may cause the fuzzy system to perform roughly.
- Provide sufficient overlap between neighbouring sets. It is suggested that triangle-to-triangle and trapezoid-to-triangle fuzzy sets should overlap between 25% to 50% of their bases.
- Review the existing rules, and if required add new rules to the rule base.
- Examine the rule base for opportunities to write hedge rules to capture the pathological behaviour of the system.
- Adjust the rule execution weights. Most fuzzy logic tools allow control of the importance of rules by changing a weight multiplier.
- Revise shapes of the fuzzy sets. In most cases, fuzzy systems are highly tolerant of a shape approximation.

11 Lecture 14 November 2006

Negnevitsky Chapter 7.

11.1 Genetic Programming

Any computer program is a sequence of operations (functions) applied to values (arguments).

Different programming languages may include different types of statements and operations, and have different syntactic restrictions.

Since genetic programming manipulates programs by applying genetic operators, a programming language should permit a computer program to be manipulated as data and the newly created data to be executed as a program.

For these reasons, LISP was chosen as the main language for genetic programming.

11.2 Pythagoras Theorem example

Steps required to set up a Genetic Programming run:

1. *Determine the set of terminals.* The terminals correspond to the inputs of the computer program to be discovered. Our program takes two inputs, a and b.
2. *Select the set of primitive functions.* The functions can be presented by standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions or domain-specific functions. Our program will use four standard arithmetic operations +, -, * and $\div$, and one mathematical function $\surd$.
3. *Define the fitness function.* A fitness function evaluates how well a particular computer program can solve the problem. For our problem, the fitness of the computer program can be measured by the error between the actual result produced by the program and the correct result given by the fitness case.
4. Decide on the parameters for controlling the run. For controlling a run, genetic programming uses the same primary parameters as those used for Genetic Algorithms. They include the population size and the maximum number of generations to be run.
5. Choose the method for designating a result of the run. It is common practice in genetic programming to designate the best-so-far generated program as the result of a run.

The run of genetic programming starts with a random generation of an initial population of computer programs. In the initial population, all computer programs usually have poor fitness, but some individuals are more fit than others.

Crossover and Mutation operations are performed on succeeding generations and the fitness increased.

12 Lecture 16 November 2006

Negnevitsky Chapter 7.

12.1 Evolutionary Approach

Intelligence can be defined as the capability of a system to adapt its behaviour to an ever-changing environment (to evolve).

Evolutionary computation simulates evolution on a computer.

According to Alan Turing, the form or appearance of a system is irrelevant to its intelligence.

The result of such a simulated evolutionary process is a series of optimisation algorithms, usually based on a simple set of rules.

Optimisation iteratively improves the quality of solutions until an optimal, or at least feasible, solution is found.

Evolutionary approach The evolutionary approach is based on computational models of natural selection and genetics. We call them *evolutionary computation*, an umbrella term that combines genetic algorithms, evolution strategies and genetic programming.

Evolutionary computation simulation: All methods of evolutionary computation simulate natural evolution by creating a population of individuals, evaluating their fitness, generating a new population through genetic operations, and repeating this process a number of times.

12.2 Genetic Algorithms

A genetic algorithm (GA) is a search technique used to find true or approximate solutions to (combinatorial) optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Steps:

1. Represent the problem variable domain as a chromosome of a fixed length, choose the size of a chromosome population N, the crossover probability p_c and the mutation probability p_m.

2. Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.

3. Randomly generate an initial population of chromosomes of size N: $x_1, x_2, \ldots, x_N$.

4. Calculate the fitness of each individual chromosome: $f(x_1), f(x_2), \ldots, f(x_N)$.

5. Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness.

6. Create pair of offspring chromosomes by applying genetic operators - crossover & mutation.

7. Place the created offspring chromosomes in the new population.

8. Repeat Step 5 until the size of the new chromosome population becomes equal to the size of the initial population, N.
9. Replace the initial (parent) chromosome population with the new (offspring) population.
10. Go to Step 4, and repeat the process until the termination criterion is satisfied.

GA represents an iterative process. Each iteration is called a generation. A typical number of generations for a simple GA can range from 50 to over 500. The entire set of generations is called a *run*.

Because GAs use a stochastic search method, the fitness of a population may remain stable for a number of generations before a superior chromosome appears.

Crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is an analogy to reproduction and biological crossover, upon which genetic algorithms are based.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. It is analogous to biological mutation.
The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified.

Termination A common practice is to terminate a GA after a specified number of generations and then examine the best chromosomes in the population. If no satisfactory solution is found, the GA is rerun.

13 Lecture 21 November 2006

Negnevitsky Chapter 7.

A simple example will help us to understand how a GA works.

Maximum value: Let us find the maximum value of the function $(15x - x^2)$ where parameter x varies between 0 and 15. For simplicity, we may assume that x takes only integer values.

Thus, chromosomes can be built with only four genes (a binary number with four digits).

Suppose that the size of the chromosome population N is 6, the crossover probability p_c equals 0.7, and the mutation probability p_m equals 0.001. The fitness function in our example is defined by: $f(x) = 15x - x^2$.

In natural selection, only the fittest species can survive, breed, and thereby pass their genes on to the next generation. GAs use a similar approach, but unlike nature, the size of the chromosome population remains unchanged from one generation to the next.

In this example, the ratio of the individual chromosome's fitness to the population's total fitness, determines the chromosome's chance of being selected for mating. The mechanism is similar to spinning a roulette wheel where the fitness ratios determine how large a segment of the wheel is assigned to each chromosome. The chromosomes average fitness improves from one generation to the next.

Crossover, In our example:

- We have an initial population of 6 chromosomes.
- To establish the same population in the next generation, the roulette wheel would be spun six times.
- Once a pair of parent chromosomes is selected, the crossover operator is applied.
- The crossover operator randomly chooses a crossover point where two parent chromosomes "break", and then exchanges the chromosome parts after that point.

- As a result, two new offspring are created.
- If a pair of chromosomes does not cross over, then the chromosome cloning takes place, and the offspring are created as exact copies of each parent.

Mutation Operator:

- Mutation is a change in the gene.
- Mutation is a background operator.
- Its role is to provide a guarantee that the search algorithm is not trapped on a local optimum.
- The mutation operator flips a randomly selected gene in a chromosome.
- The mutation probability is quite small in nature, and is kept low for GAs, typically in the range between 0.001 and 0.01.

14 Lecture 23 November 2006

Machine Learning, Mitchell, Chapter 1

14.1 Machine Learning

Three niches for machine learning:

1. Data mining - using historical data to improve decisions (e.g. medical records $\rightarrow$ medical knowledge).
2. Software applications which cannot be programmed by hand (e.g. autonomous driving, speech recognition).
3. Self customizing programs (e.g. newsreader that learns user interests, adaptive spam filters).

Definition: A computer program is said to *learn* from experience E with respect to come class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

14.2 Checkers learning program

- Task T: Playing Checkers.
- Performance Measure P: Percent of games won in the world tournament.
- Training Experience E: Games played against self.
- Target Function: $V : Board \rightarrow \Re$.
- Target Function Representation: $\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$. Where:
  - x_1: the number of black pieces on the board.
  - x_2: the number of red pieces on the board.
  - x_3: the number of black kings on the board.
  - x_4: the number of red kings on the board.
  - x_5: the number of black pieces threatened by red (i.e. which can be captured on red's next turn).

- x_6: the number of red pieces threatened by black.
- and $w_0 \dots w_6$ are numerical coefficients, or weights, to be chosen by the learning algorithm. These can be refined using the least mean squares (LMS) training rule. *See Mitchell pages 5-14.*

15 Lecture 28 November 2006

Machine Learning, Mitchell, Chapter 2

15.1 Concept Learning and General-to-Specific Ordering

- Concept learning can be cast as a problem of searching through a large predefined space of potential hypotheses.
- The general-to-specific partial ordering of hypotheses, which can be defined for any concept learning problem, provides a useful structure for organizing the search through the hypothesis space.
- The Find-S algorithm utilizes this general-to-specific ordering, performing a specific-to-general search through the hypothesis space along one branch of the partial ordering, to find the most specific hypothesis consistent with the training examples.

15.2 Find-S Algorithm

```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
    (a) For each attribute constraint $a_i$ in $h$
        i. If the constraint $a_i$ is satisfied by $x$ then do nothing
           Else replace $a_i$ in $h$ by the next most general constraint that is satisfied by $x$
3. Output hypothesis $h$
```

16 Lecture 30 November 2006

Machine Learning, Mitchell, Chapter 2

16.1 Inductive Learning

- The Candidate-Elimination algorithm utilizes the general-to-specific ordering to compute the version space (the set of all hypotheses consistent with the training data) by incrementally computing the set of maximally specific (S) and maximally general (G) hypotheses.
- Because the S and G sets delimit the entire set of hypotheses consistent with the data, they provide the learner with a description of its uncertainty regarding the exact identity of the target concept.
- Version spaces and the Candidate-Elimination algorithm provide a useful conceptual framework for studying concept learning. However, this learning algorithm is not robust to noisy data or to situations in which the unknown target concept is not expressible in the provided hypothesis space.

- Inductive learning algorithms are able to classify unseen examples only because of their implicit inductive bias for selecting one consistent hypothesis over another. The bias associated with the Candidate-Elimination algorithm is that the target concept can be found in the provided hypothesis space. The output hypotheses and classifications of subsequent instances follow *deductively* from this assumption together with the observed training data.
- An unbiased learner cannot make inductive leaps to classify unseen examples.

16.2 Candidate-Elimination Algorithm

- Initialize G to the set of maximally general hypotheses in H.
- Initialize S to the set of maximally specific hypotheses in H.
- For each training example d, do

```
1. If $d$ is a positive example
   (a) Remove from $G$ any hypothesis inconsistent with $d$
   (b) For each hypothesis $s$ in $S$ that is not consistent with $d$
       i. Remove $s$ from $S$
       ii. Add to $S$ all minimal generalizations $h$ of $s$ such that $h$ is consistent with $d$,
           and some member of $G$ is more general than $h$
       iii. Remove from $S$ any hypothesis that is more general than any other hypothesis
            in $S$
2. If $d$ is a negative example
   (a) Remove from $S$ any hypothesis inconsistent with $d$
   (b) For each hypothesis $g$ in $G$ that is not consistent with $d$
       i. Remove $g$ from $G$
       ii. Add to $G$ all minimal specializations $h$ of $g$ such that $h$ is consistent with $d$,
           and some member of $S$ is more specific than $h$
       iii. Remove from $G$ any hypothesis that is less general than any other hypothesis
            in $G$
```

The main application for concept learning has been in the field of Medicine.

17 Lecture 5 December 2006

Machine Learning, Mitchell, Chapter 3

17.1 Decision Trees

Decision tree learning provides a practical method for concept learning and for learning other discrete-valued functions. The ID3 family of algorithms infers decision trees by growing them from the root downwards, greedily selecting the next best attribute for each new decision branch added to the tree.

Entropy can be defined as a measurement of the difficulty of representing something. A set of data with high entropy requires more bits to encode it than a set with low entropy.

Use Decision Trees when:

- Discrete Values Attributes - preferably of small set size.
- Disjunctive hypothesis required (using OR).
- Possibly noisy training data (incorrect examples).

Top down induction Decision Trees.

Try to reduce the search space as much as possible - preferably to equally sized subsets.

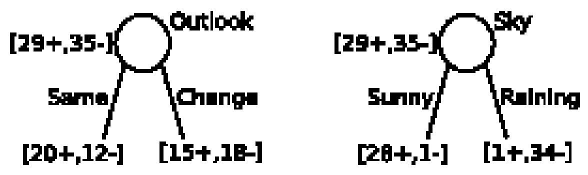


Figure 6: Alternate Questions

18 Lecture 7 December 2006

Machine Learning, Mitchell, Chapter 3

Use entropy to measure how much information is needed to describe data set. While noise has high entropy, structured data (e.g. A-Z 10,000 times) has low.

So pose questions in decision trees which split the search space as evenly as possible between their child nodes. However, far prefer questions which reduce entropy. If a question were found for the example pictured where the answers were [29+,0-] and [0+,35-] that would be ideal. Fully classifies training data and thus minimizes entropy.

Entropy of data can be thought of as the number of bits needed to represent the information.

For example, if all the training examples were positive then that would be easy to represent. If roughly half were negative then that is more difficult to encode.

$Entropy(S) = (-P_{\oplus} log_2 P_{\oplus})(-P_{\ominus} log_2 P_{\ominus})$[1]

Information Theory is the source of this concept of entropy.

19 Lecture 12 December 2006

Machine Learning, Mitchell, Chapter 3

19.1 ID3

Greedy algorithm - always ask the question which most reduces the entropy of the information set. Quickly finds solution, but may find a local minimum.

Inductive Bias Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

Occam's Razor: Prefer the simplest hypothesis that fits the data (*lex parsimoniae* law of succinctness).

ID3 searches a complete hypothesis space (i.e. the space of decision trees can represent any discrete-valued function defined over discrete-values instances). It thereby avoids the major difficulty associated with approaches that consider only restricted sets of hypotheses: that the target function might not be present in the hypothesis space.

The inductive bias implicit in ID3 includes a *preference* for smaller trees; that is, its search through the hypothesis space grows the tree only as large as needed in order to classify the available training examples.

Overfitting the training data is an important issue in decision tree learning. Because the training examples are only a sample of all possible instances, it is possible to add branches to the tree that improve performance on the training examples while decreasing performance on other instances outside this set. Methods for post-pruning the decision tree are therefore important to

[1] $P_{\oplus}$ proportion of training examples which are positive; $P_{\ominus}$ proportion of training examples which are negative; $log_2 \frac{1}{2} = -1$.

avoid overfitting in decision tree learning (and other inductive inference methods that employ a preference bias).

19.2 Rule post-pruning

1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.
2. Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
3. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Advantages of converting decision trees to rules before pruning:

1. Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path. In contrast, if the tree itself were pruned, the only two choices would be to remove the decision node completely, or to retain it in its original form.
2. Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, we avoid messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the sub-tree below this test.
3. Converting to rules improves readability. Rules are often easier for people to understand.

19.3 ID3 Algorithm

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. *Target_attribute* is the attribute whose value is to be predicted by the tree. *Attributes* is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

1. Create a *Root* node for the tree
2. If all *Examples* are positive, Return the single-node tree *Root*, with label = +
3. If all *Examples* are negative, Return the single-node tree *Root*, with label = -
4. If *Attributes* is empty, Return the single-node tree *Root*, with label = the most common value of *Target_attribute* in *Examples*
5. Otherwise begin:
   (a) $A \leftarrow$ the attribute from *Attributes* that best[2] classifies *Examples*
   (b) The decision attribute for *Root* $\leftarrow A$
   (c) For each possible value v_i of A:
      i. Add a new branch below *Root*, corresponding to the test $A = V_i$
      ii. Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A

[2]The best attribute is the one with the highest information gain.

iii. If $Examples_{v_i}$ is empty

A. Then below this new branch add a leaf node with label = the most common value of *Target_ attribute* in *Examples*

B. Else below this new branch add the sub-tree *ID3(*$Examples_{v_i}$*, Target_ attribute, Attributes*$-\{A\}$*)*

19.4 Extensions to ID3

Continuous Valued Attribute Construct discrete-valued attribute that represents continuous values. Create Boolean attributes $\Rightarrow$ between negative to positive transitions (and positive to negative transitions). E.g. in a range of temperatures.

Attributes with many values Use a Gain ratio (divide by the number of possible values in the entropy reduction calculation) to remove bias in favors of such attributes.

Attributes with costs Divide Gain ratio by cost to learn consistent tree with a low expected cost. E.g. Blood Tests are more expensive to perform than X-rays.

Unknown Attribute Values Use training example anyway $\Rightarrow$ assign values most consistent with those in other examples (e.g. most common or average).

20 Lecture 16 January 2007

Machine Learning, Mitchell, Chapter 4;
Artificial Intelligence, Negnevitsky, Chapter 6.

20.1 Artificial Neural Networks

An Artificial Neural Network (ANN) consists of a number of very simple and highly interconnected processors, called neurons, which are analogous to the biological neurons in the brain. The neurons are interconnected by weighted links that pass signals from one neuron to another. Each link has a numerical weight associated with it. Weights are the basic means of long-term memory in ANNs. They express the strength or importance of each neuron input. A neural network 'learns' from repeated adjustments of these weights.

20.1.1 Biological Motivation

ANNs are modelled on the human brain, which consists of 10^{11} neurons, each connected, on average to 10^4 others.

The fastest biological neuron switching times are known to be on the order of 10^{-3} seconds (quite slow compared to computer switching speeds of 10^{-10} seconds). Yet humans are able to make complex decisions surprisingly quickly.

For example, it requires approximately 10^{-1} seconds to visually recognise your mother. Notice that the sequence of neuron firings than can take place during this 10^{-1} second interval cannot possibly be longer than a few hundred steps, given the switching speed of single neurons. This observation has led many to believe that the information-processing abilities of such biological systems must follow from highly parallel processes.

One motivation for ANN systems is to capture this kind of highly parallel computation.

20.1.2 Appropriate Problems for ANNs

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras or microphones.

It is appropriate for problems with the following characteristics:

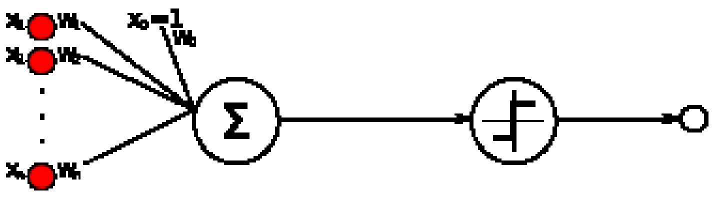


Figure 7: Perceptron

- Instances are represented by many attribute-value pairs.
- The target function output may be discrete-valued, real-valued or a vector of several real- or discrete-valued attributes.
- The training examples may contain errors.
- Long training times are acceptable.
- Fast evaluation of the learned target function may be required.
- The ability of humans to understand the learned target function is not important.

Examples:

- Driving a car (ALVINN)
- Speech Recognition (automated telephone systems)
- Image Classification (Security, Medical)
- Financial Prediction (Learn investor behaviour in response to newspaper articles)

20.1.3 Perceptrons

A Perceptron is an artificial neuron. It combines the values and weightings of its inputs into a single output.

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

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) \begin{cases} 1 & if\, \vec{w} \times \vec{x} > 0 \\ -1 & otherwise \end{cases}$$

The Perceptron is a single layer neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The training technique used is called the perceptron learning rule. The perceptron generated great interest due to its ability to generalize from its training vectors and work with randomly distributed connections. Perceptrons are especially suited for simple problems in pattern classification.

21 Lecture 18 January 2007

Machine Learning, Mitchell, Chapter 4

21.1 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. 0.01) called *learning rate*

The learning rate is often decreased when the remaining error is small to fine-tune the results of training.

Least Mean Square (LMS) rule used in calculating Gradient Descent. Work out gradient (in error curve) and choose change in weights which will get you down the slope as quickly as possible. (E.g. If the error curve is given by the formula $y = ax^2 + bx + c$ then, to get the slope at any point, differentiate getting $\frac{dy}{dx} = 2ax + b$)

21.2 Gradient-Descent Algorithm

```
GRADIENT-DESCENT(training_examples, η)
    Each training example is a pair of the form ⟨$\vec{x}$, t⟩, where $\vec{x}$ is the vector of input values, and t
is the target output value. η is the learning rate (e.g., 0.05).

  • Initialize each $w_i$ to some small random value
  • Until the termination condition is met, Do
      – Initialize each $\Delta w_i$ to zero.
      – For each ⟨$\vec{x}$, t⟩ 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
            $w_i \leftarrow w_i + \Delta w_i$
```

21.3 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

22 Lecture 23 January 2007

Machine Learning, Mitchell, Chapter 4

22.1 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.

22.2 Sigmoid Unit

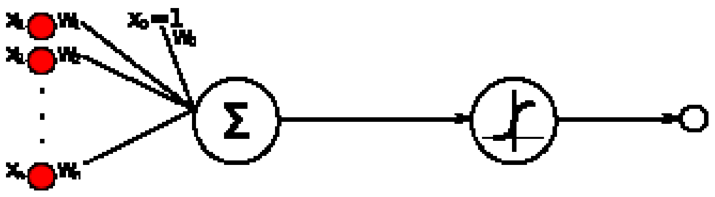


Figure 8: Sigmoid Unit

$\sigma(x)$ is the sigmoid function $\frac{1}{1+e^{-x}}$.
Nice property $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$.
We can derive gradient descent rules to train

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

22.3 Backpropagation Algorithm

The summary of the technique is as follows:

1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.
5. Assign "blame" for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat the steps above on the neurons at the previous level, using each one's "blame" as its error.

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

Points to note:

- 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 be 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.

23 Lecture 25 January 2007

Machine Learning, Mitchell, Chapter 4

Error plain is continuous so long as sigmoid units are used in the ANN.

The hidden layer(s) require enough neurons to encode all of the states that the ANN is to learn. With a single hidden layer of n neurons 2^n states can be learned. With two hidden layers, each of n neuron, 2^{n^n} states can be learned.

An ANN with two hidden layers takes much longer to teach than one with only a single hidden layer.

Every Boolean function can be represented by a single hidden layer. Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer. Any function can be approximated to arbitrary accuracy by a network with two hidden layers.

Rule of thumb is to start with a single hidden layer and add a second if ANN is not learning properly from training examples.

23.1 Convergence of Back-propagation

Gradient descent to some local minimum

- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train multiple nets with different initial weights

Nature of convergence

- Initialize weights near zero
- Therefore, initial networks near-linear
- Increasingly non-linear functions possible as training progresses

24 Lecture 30 January 2007

Machine Learning, Mitchell, Chapter 5

24.1 Evaluating Hypotheses

Key points:

- Sample error, true error
- Confidence intervals for observed hypothesis error
- Estimators
- Binomial distribution, Normal distribution, Central Limit Theorem
- Paired t tests
- Comparing learning methods

24.2 Sample Error and True 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 $\mathcal{D}$. $error_D(h) \equiv Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$.

The *sample error* of h with respect to target function f and data sample S is the proportion of examples h misclassifies $error_S(h) \equiv \frac{1}{n}\Sigma_{x \in S}\delta(f(x) \neq h(x))$. Where $\delta(f(x) \neq h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

1. Bias: If S is training set, $error_S(h)$ is optimistically biased $bias \equiv E[error_S(h)] - error_D(h)$ For unbiased estimate, h and S must be chosen independently
2. *Variance*: Even with unbiased S, $error_S(h)$ may still vary from $error_D(h)$

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

$$error_S(h) \pm \sqrt{\frac{error_S(h)(1-error_S(h))}{n}} \text{ (i.e. Standard Deviation).}$$

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

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

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

24.3 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 $\sigma_{error_S(h)}$

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

24.4 Normal Probability Distribution

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

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, $E[X] = \mu$.
- Variance of X is $Var(X) = \sigma^2$.
- Standard deviation of X, $\sigma_X = \sigma$.

24.5 Central Limit Theorem

Consider a set of independent, identically distributed random variables Y_{1n}, 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 $\frac{\sigma^2}{n}$.

25 Lecture 1 February 2007

Machine Learning, Mitchell, Chapter 5

25.1 Difference Between Hypotheses

Comparing the relative effectiveness of two learning algorithms is an estimation problem that is relatively easy when data and time are unlimited, but more difficult when these resources are limited. One possible approach is to run the learning algorithms on different subsets of the available data, testing the learned hypotheses on the remaining data, then averaging the results of these experiments.

When you subtract a normal distribution from another you get a normal distribution as the result. Subtracting estimated errors gives a new normal distribution with a new Standard Deviation, confidence interval, etc.

25.2 Paired t test

Given two paired sets X_i and Y_i of n measured values, the paired t-test determines whether they differ from each other in a significant way under the assumptions that the paired differences are independent and identically normally distributed.

To apply the test, let $\hat{X}_i = {}_i - \overline{X})$ and $\hat{Y}_i = {}_i - \overline{Y})$.

then define t by

$$t = (\overline{X} - \overline{Y})\sqrt{\frac{n(n-1)}{\Sigma_{i=1}^{n}(\hat{X}_i - \hat{Y}_i)^2}}$$

26 Lecture 6 February 2007

Machine Learning, Mitchell, Chapter 6

26.1 Bayesian Learning

Bayes Theorem

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

$P(h)$ = prior probability of hypothesis h (e.g. h is that someone has the common cold; then this is based on the number of people who have common cold in the general population).

$P(D)$ = prior probability of training data D (e.g. D is that the person has a cough; then this is based on the number of people in the general population who have a cough).

$P(h|D)$ = probability of h given D (e.g. probability that someone has the common cold given that they have a cough).

$P(D|h)$ = probability of D given h (e.g. probability that someone has a cough given that they have the common cold).

The final answer - the estimated probability that a patient has the common cold given that they have a cough - is known as the revised probability or the posterior probability.

Bayesian methods provide the basis for probabilistic learning methods that accommodate (and require) knowledge about the prior probabilities of alternative hypotheses and about the probability of observing various data given the hypothesis. Bayesian methods allow assigning a posterior probability to each candidate hypothesis, based on these assumed priors and observed data.

27 Lecture 8 February 2007

Machine Learning, Mitchell, Chapter 6

27.1 Choosing Hypotheses

Generally we want to choose the most probably hypothesis given the training data.

Maximum a posteriori hypothesis h_{MAP}: $argmax_{h \in H} P(h|D) = argmax_{h \in H} \frac{P(D|h)P(h)}{P(D)} = argmax_{h \in H} P(D|h)P(h)$.

If we assume $P(h_i) = P(h_j)$ then we can further simplify and choose the Maximum Likelihood h_{ML}: $argmax_{h_i \in H} P(D|h_i)$.

27.2 Probability Formulae

- Product Rule: Probability $(A \wedge B)$ of a conjunction of two events A and B: $P(A \wedge B) = P(A|B)P(B) = P(B|A)P(A)$.
- Sum Rule: Probability $(A \vee B)$ of a disjunction of two events A and B: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.
- Theorem of total probability: If events $A_1, \ldots, A_n$are mutually exclusive with $\Sigma_{i=1}^{n} P(A_i) = 1$ then, for any event B: $P(B) = \Sigma_{i=1} P(B|A_i)P(A_i)$.

27.3 Find-S

Consider the usual concept learning task

- Instance space X, hypothesis space H, training examples D.
- Consider the FIND-S learning algorithm (outputs the most specific hypothesis from the version space $VS_{H,D}$).

1. Assume a fixed set of instances $\langle x_1, \ldots, x_m \rangle$
2. Assume D is the set of classifications $D = \langle c(x_1), \ldots, c(x_m) \rangle$
3. Choose $P(D|h)$:
   (a) $P(D|h) = 1$ if h is consistent with D
   (b) $P(D|h) = 0$ otherwise
4. Choose $P(h)$ to be uniform distribution
   (a) $P(h) = \frac{1}{|H|}$ for all h in H
5. Then

$$P(h|D) \quad = \quad \begin{cases} \frac{1}{|VS_{H,D}|} & if\, h\, is\, consistent\, with\, D \\ 0 & otherwise \end{cases}$$

The Bayesian framework allows one way to characterise the behaviour of learning algorithms (e.g. FIND-S), even when the learning algorithm does not explicitly manipulate probabilities. By identifying probability distributions $P(h)$ and $P(D|h)$ under which the algorithm outputs optimal (i.e. MAP) hypotheses, we can characterise the implicit assumptions under which this algorithm behaves optimally.

28 Lecture 13 February 2007

Machine Learning, Mitchell, Chapter 6

Can characterise learning algorithms using equivalent MAP learners and making the prior assumptions explicit. Assumptions for the Candidate Elimination Algorithm turn out to be the same as for FIND-S.

28.1 Learning a Real Valued Function

Consider any real-valued target function f.

Training examples $\langle x_i, d_i \rangle$, where d_i is noisy training value.

- $d_i = f(x_i) + e_i$
- e_i is random variable (noise) drawn independently for each x_i according to some Gaussian distribution with mean$=0$

Then the maximum likelihood hypothesis h_{ML} is the one that minimizes the sum of squared errors: $h_{ML} = argmin_{h \in H} \Sigma_{i=1}^{m} (d_i - h(x_i))^2$.

28.2 Learning to Predict Probabilities

Consider predicting survival probability from patient data.

Training examples $\langle x_i, d_i \rangle$, where d_i is 1 or 0.

Want to train a neural network to outputs probability given x_i (not a 0 or 1).

In this case can show $h_{ML} = argmax_{h \in H} \Sigma_{i=1}^{m} d_i ln\, h(x_i) + (1 - d_i) ln(1 - h(x_i))$.

28.3 Minimum Description Length

Occam's razor: prefer the shortest hypothesis.

MDL: prefer the hypothesis h that minimizes $h_{MDL} = argmin_{h \in H} L_{C_1}(h) + L_{C_2}(D|h)$ where $L_C(x)$ is the description length of x under encoding C.

In other words, prefer the hypothesis that minimizes $length(h) + length(misclassifications)$.

28.4 Most Probable Classification of New Instances

So far we've sought the most probable hypothesis given the data (i.e. h_{MAP}).

Given a new instance x, what is its most probable *classification*?

In general the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities. If the possible classification of the new example can take on any value v_j from some set V, then the probability $P(v_j|D)$ that the correct classification of the new instance is v_j is just

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i) P(h_i|D)$$

The optimal classification of the new instance is the value $v_j$for which $P(v_j|D)$ is maximum.

28.5 Bayes Optimal Classification

$$argmax_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Let the set of possible classifications of the new instance $V = \{\oplus, \ominus\}$, and

$$P(h_1|D) = .4,\ P(\ominus|h_1) = 0,\ P(\oplus|h_1) = 1$$
$$P(h_2|D) = .3,\ P(\ominus|h_2) = 1,\ P(\oplus|h_2) = 0$$
$$P(h_1|D) = .3,\ P(\ominus|h_3) = 1,\ P(\oplus|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(\oplus|h_i)P(h_i|D) = .4$$
$$\sum_{h_i \in H} P(\ominus|h_i)P(h_i|D) = .6$$

and

$$argmax_{v_j \in \{\oplus,\ominus\}} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = \ominus$$

Any system that classifies new instances in this way is called a *Bayes Optimal Classifier*, or Bayes optimal learner.

29 Lecture 15 February 2007

Machine Learning, Mitchell, Chapter 6

29.1 Gibbs Classifier

The Bayes Optimal Classifier provides the best result but can be computationally expensive if there are many hypotheses.

Gibbs algorithm:

1. Choose one hypothesis at random, according to $P(h|D)$.
2. Use this to classify new instance.

Surprisingly, assuming that the target concepts are drawn at random from H, according to the prior probability distribution assumed by the learner, then the error of the Gibbs algorithm is at worst twice that of the Bayes Optimal classifier.

29.2 Naive Bayes Classifier

A step up from Gibbs, this is one of the most practical learning methods.

Use it when you have moderate or large training sets and the attributes that describe instances are conditionally independent given classification.

Assume target function $f :\rightarrow V$, where each instance of x is described by the attributes $\langle a_1, a_2, \ldots a_n \rangle$.

The most probably value of $f(x)$ is:

$$v_{MAP} = argmax_{v_j \in V} P(a_1, a_2, \ldots a_n|v_j)P(v_j)$$

The naive Bayes assumption:

$$P(a_1, a_2, \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

which gives the Naive Bayes classifier:

$$v_{NB} = argmax_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

The naive Bayes learning method involves a learning step in which the various $P(v_j)$ and $P(a_i|v_j)$ terms are estimated, based on their frequencies over the training data. The set of these estimates corresponds to the learned hypothesis. This hypothesis is then used to classify each new instance.

Even when the attributes of the training examples are not completely conditionally independent, naive Bayes still gives fairly good results.

Beware of attribute values with low (or no) occurrences in the training examples. Typical solution is to use the following Bayesian estimate:

$$\hat{P}(a_i | v_i) \leftarrow \frac{n_c + mp}{n + m}$$

where

- n is number of training examples for which $v = v_j$;
- n_c is number of examples for which $v = v_j$ and $a = a_i$;
- p is prior estimate for $\hat{P}(a_i | v_i)$;
- m is weight given to prior (i.e. number of "virtual" examples).

30 Lecture 20 February 2007

Machine Learning, Mitchell, Chapter 6

30.1 Bayesian Belief Networks

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities. BBNs allow stating conditional independence assumptions that apply to subsets of the variables.

Allows combining of prior knowledge about (in)dependencies among variables with observed training data.

A BBN can be represented as a directed acyclic graph.

31 Lecture 22 February 2007

Machine Learning, Mitchell, Chapter 6

31.1 Bayesian Belief Networks

Want to have graphs with small n-degrees (numbers of arcs going into a single node). The number of conditional probabilities is exponentially related to the number of these arcs.

Computational complexity of belief network is dependent on the structure of the graphs.

31.1.1 Inference

How can one infer the (probabilities of) values of one or more network variables, given observed values of others?

- Bayes net contains all information needed for this inference.
- If only one variable with unknown value, easy to infer it
- In general case, problem is NP-hard.
  In practice, can succeed in many cases
- Exact inference methods work well for some network structures
- Monte Carlo[3] methods "simulate" the network randomly to calculate approximate solutions.

31.1.2 Learning

Several variants of this learning task:

- Network structure might be known or unknown
- Training examples might provide values of all network variables, or just some

If structure known and observe all variables:

- Then it's easy as training a Naive Bayes classifier

Suppose structure known, variables partially observable
Similar to training neural network with hidden units
In fact, can learn network conditional probability tables using gradient ascent!
Converge to network h that (locally) maximizes $P(D|h)$.

31.2 Expectation Maximisation (EM) Algorithm

Repeatedly:

1. Calculate probabilities of unobserved variables, assuming h
2. Calculate new maximum likelihood hypothesis h' $E[lnP(D|h')]$ where D now includes both observed and (calculated probabilities of) unobserved variables

When structure unknown...

- Algorithms use greedy search to add/subtract edges and nodes

Note that only when there is a variance in the conditional probabilities does an edge need to be defined, otherwise remove it.
Uses:

- Train Bayesian Belief Networks.
- Unsupervised Learning (AUTOCLASS).
- Learning Hidden Markov Model.

EM general-method will not be coming up in exam.

[3]Monte Carlo Method: Generate Random input and observe outcomes. Assume probability distribution of input values.

32 Lecture 27 February 2007

32.1 GeNIe

The GeNIe (Graphical Network Interface) software package can be used to create decision theoretic models intuitively using the graphical click-and-drop interface.

33 Lecture 1 March 2007

33.1 GeNIe

34 Lecture 6 March 2007

Machine Learning, Mitchell, Chapter 13

34.1 Reinforcement Learning

In computer science, reinforcement learning is a sub-area of machine learning concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states.

The environment is typically formulated as a finite-state Markov decision process (MDP), and reinforcement learning algorithms for this context are highly related to dynamic programming techniques. State transition probabilities and reward probabilities in the MDP are typically stochastic but stationary over the course of the problem.

Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). The exploration vs. exploitation trade-off in reinforcement learning has been mostly studied through the multi-armed bandit problem.

Formally, the basic reinforcement learning model consists of:

1. a set of environment states S;
2. a set of actions A; and
3. a set of scalar "rewards" in $\mathbb{R}$.

At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives from the environment the new state s_{t+1} and a reward r_{t+1}. Based on these interactions, the reinforcement learning agent must develop a policy $\pi : S \to A$ which maximizes the quantity $R = r_0 + r_1 + \ldots + r_n$ for MDPs which have a terminal state, or the quantity $R = \Sigma_t \gamma^t r_t$ for MDPs without terminal states (where γ is some "future reward" discounting factor between 0.0 and 1.0).

Note that the target function is $\pi : S \to A$, but we have no training examples in the form $\langle s, a \rangle$, training examples are of the form $\langle \langle s, a \rangle, r \rangle$.

Thus, reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications, backgammon and chess.

34.2 Markov Decision Processes

Given: Finite set of states S; A set of actions A.

At each discrete time agent observes state $s_t \in S$ and chooses action $a_t \in A$.

The agent then receives immediate reward r_t and state changes to s_{t+1}.

Markov assumption: $s_{t+1} = r(s_t, a_t)$ and $r_t = \delta(s_t, a_t)$.
That is, r_t and s_{t+1} depend only on *current* state and action.
Functions δ and r may be non-deterministic.
Functions δ and r are not necessarily known to the agent.

35 Lecture 8 March 2007

Machine Learning, Mitchell, Chapter 13

35.1 Q Function

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)

It could then do a lookahead search to choose best action from any state s because $\pi^*(s) = argmax_a[r(s,a) + \gamma V^*(\delta(s,a))]$.

This works well if agent knows $\delta : S \times A \rightarrow S$ and $r : S \times A \rightarrow \Re$ But when it doesn't, it can't choose actions this way.

Define new function very similar to V^*.

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

If agent learns Q, it can choose optimal action even without knowing δ!

$$\pi^*(s) = argmax_a[r(s,a) + \gamma V^*(\delta(s,a))]$$

$$\pi^*(s) = argmax_a Q(s,a)$$

Q is the evaluation function the agent will learn.

Note Q and V^* are closely related: $V^*(s) = max_{a'} Q(s,a')$

Which allows us to write Q recursively as $Q(s_t,a_t) = r(s_t,a_t) + \gamma max_a Q(s_{t+1},a')$

Let $\hat{Q}$ denote learner's current approximation to Q. Consider training rule $\hat{Q}(s,a) \leftarrow r + max_{a'}\hat{Q}(s',a')$

where s' is the state resulting from applying action a in state s.

35.2 Q Learning for Deterministic Worlds

For each s, a initialize table entry $\hat{Q}(s,a) \leftarrow 0$

Observe current state s

Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s,a)$ as follows:

$$\hat{Q}(s,a) \leftarrow r + \gamma max_{a'}\hat{Q}(s',a')$$

- $s \leftarrow s'$

$\hat{Q}$ converges to Q.

35.3 Non-deterministic Case

Q learning generalises to non-deterministic worlds.

Alter training rule to

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + max_{a'}\hat{Q}_{n-1}(s',a')]$$

where $\alpha_n = \frac{1}{1+visits_n(s,a)}$.

$\hat{Q}$ still converges to Q.

36 Lecture 13 March 2007

Machine Learning, Mitchell, Chapter 13

36.1 Temporal Difference Learning

The Q learning algorithm learns by iteratively reducing the discrepancy between Q value estimates for adjacent states. In this sense, Q learning is a special case of a general class of *temporal difference* algorithms that learn by reducing discrepancies between estimates made by the agent at different times. We can design an algorithm that reduces discrepancies between the current state and more distant descendants or ancestors.

In other words, the goal of learning is to make the learner's current prediction for the current input pattern more closely match the next prediction at the next time step.

One step lookahead:
$Q^{(1)}(s_t,a_t) \equiv r_t + \gamma max_a \hat{Q}(s_{t+1},a)$
Why not two steps?
$Q^{(2)}(s_t,a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 max_a \hat{Q}(s_{t+2},a)$
Or n?
$Q^{(n)}(s_t,a_t) \equiv r_t + \gamma r_{t+1} + \ldots + \gamma^{(n-1)} r_{t+1} + \gamma^n max_a \hat{Q}(s_{t+n},a)$
Blend all of these:
$Q^{\lambda}(s_t,a_t) \equiv (1-\lambda)\left[Q^{(1)}(s_t,a_t) + \lambda Q^{(2)}(s_t,a_t) + \lambda^2 Q^{(3)}(s_t,a_t) + \ldots\right]$
Equivalent expression:
$Q^{\lambda}(s_t,a_t) = r_t + \gamma\left[(1-\lambda) max_a \hat{Q}(s_t,a_t) + \lambda Q^{\lambda}(s_{t+1},a_{t+1})\right]$
$TD(\lambda)$ algorithm uses above training rule.

- Sometimes converges faster than Q learning
- Converges for learning V^* for any $0 \le \lambda \le 1$ (Dayan, 1992)
- Tesauro's TD-Gammon uses this algorithm

36.2 Subtleties and Ongoing Research

- Replace $\hat{Q}$ table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\hat{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming

37 Lecture 15 March 2007

Demo of Simbrain.

38 Lecture 20 March 2007

38.1 Review of Course

In the exam there is usually a basic questions about the fundamentals of AI and the motivation of the entire field.

Knowledge Representation in Expert Systems

Know the terminology: Conjunctions (AND), Disjunctions (OR), Antecedent (IF part) and Consequence (THEN part).

Know team members and their roles in building an expert system: Domain Expert, Knowledge Engineer, Programmer, Project Manager and end User.

Describe the Newell and Simon Model ⇒ Separate Knowledge Base and Database feed Inference engine which in turn feeds the explanation facility and user interface.

Probabilistic Reasoning

Know and be able to explain Bayes Rule.

Fuzzy Logic

Definitely coming up in exam.

Be able to describe a problem in terms of fuzzy sets.

Compare and contrast crisp set theory (and operators) with that of fuzzy set theory.

Know about Mamdani inference method and be able to compare Sugeno method.

Terminology: Universe of Discourse (X-axis).

39 Lecture 22 March 2007

39.1 Review of Course (continued)

Genetic Algorithms

Know the main features of the Neo-Darwinian approach.

Know the steps in Basic Genetic Algorithm.

Machine Learning

History of Machine Learning is not coming up.

Know about the Inductive Learning Hypothesis (can learn from examples).

We will not be asked to write out algorithms.

The most important algorithm to know about is Candidate Elimination.

We do not need to know about Inductive Systems and Equivalent Deductive Systems.

Decision Tree Learning

Need to understand the ID-3 Learning Algorithm, and why entropy is important. Why it is best to ask questions that maximize your information gain; cut your candidate solution space in half.

Know about overfitting, but don't worry about the methods to detect and fix it.

Note That Artificial Neural Networks are the method of choice if the training data is noisy (contains errors).

Neural Networks

Neural Networks and Fuzzy Systems are the two most important topics.

Need to be able to explain why it is preferable to use sigmoid functions rather than Perceptrons: The continuous output allow Gradient Descent to find minima.

We do not need to know the back-propagation algorithm, but we do need to be able to explain why it works with gradient descent. Reduces error by adjusting weights.

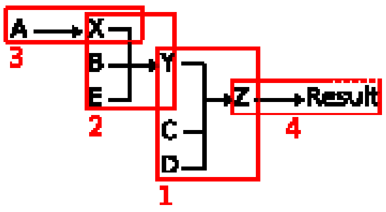


Figure 9: Inference Chain

We need to be able to explain gradient descent (including the order of learning weights in Back Propagation).

40 Lecture 29 March 2007

40.1 Exam

Exam is worth 60%; Answer 5 of 7 questions.

Sample questions and answers follow, questions taken from last year's paper.

1 (a) What is a Production System Model?
List the five basic components.
Draw a diagram.

- Based on the idea that humans solve problems by applying their knowledge, expressed as a production rule, to a given problem, represented with problem-specific information.

1. Knowledge Base - holds set of rules.
2. Database - holds set of facts.
3. Inference Engine - Software for firing rules and learning new facts.
4. Explanation Facilities - Software for describing what rules were fired.
5. User Interface.

(4 marks)

1 (b) Draw an inference chain for this rule base:

1. $Y \&\& C \&\& D \rightarrow D$
2. $X \&\& B \&\& E \rightarrow Y$
3. $A \rightarrow X$
4. $Z \rightarrow$ result is true

(5 marks)

1 (c) Is Forward Chaining data or goal driven?
Explain your answer.

Data driven. Learn new facts from facts already known.

Order of firing of rules in Forward Chaining: $3, 2, 1, 4$.

(5 marks)

1 (d) What is Bayes Rule? How does it calculate hypothesis likelihood?

[This question is about Bayesian Reasoning within rule-Based systems]

h is hypothesis, D is observed data.

Bayes Rule: $p(h|D) = \frac{p(D|h) \times p(h)}{p(D)}$

It allows reasoning under uncertainty to encompass prior information in probability of hypothesis and probabilities of certain data given the hypothesis.

You can include probabilities in the results or just pick hypothesis with greatest probability given the data.

(5 marks)

2 (a) Name three processes at Neo-Darwinism is based on.

1. Reproduction
2. Mutation
3. Competition/Selection

(3 marks)

2 (b) Briefly describe the process of evolutionary computation simulation.

1. Create a population of individuals.
2. Evaluate their fitness.
3. Generate a new population through genetic operations.
4. Repeat from step 2 a number of times.

(4 marks)

2 (c) What are GAs?

See notes.

2 (d) We wish to get the "peak" or max of a function with two variables $f(x, y) = (1-x)^2 e^{-x+y} \ldots$. Suggest a way of modelling this problem.

Represent x and y as a string of bits. Normalise x and y values. Explain how the number of bits used will determine the accuracy of the results (e.g. 4−bits mean 16 possible values).

2 (e) How do you expect the fitness to change as the number of generations increases?

Best solution found will monotonically increase.

(2 marks)

41 Lecture 30 March 2007

41.1 Exam (continued)

TAC will not be coming up in the exam. We don't need to know proofs.

Answer 5 of 7 questions. Artificial Neural Networks and Fuzzy Logic is definitely coming up. The other questions will cover topics from: Genetic Algorithms, Concept Learning, Bayesian Learning, Reinforcement Learning, Decision Trees and Rule-Based systems (possibly more)

Topic areas:

41.1.1 Decision Trees

How are they constructed? What questions to ask?
Theory:

- What is entropy? A measure of the number of bits needed to encode information.
  Know how to calculate it. Count number of positive ($\oplus$) and negative ($\ominus$) examples for each attribute: $E(a_i) = -P \oplus log_2 P \oplus -P \ominus log_2 P \ominus$.
- Information Gain. What questions should be asked to eliminate the maximum number of options?
  Build up a picture of entropy of each attribute. Choose the attribute which when asked about decreases the overall entropy by most.
- What should be the root node?
- Classify examples.
- When are Decision Trees best suited?
  DTs give answers very quickly. They are more flexible than concept learning. Can cope with noisy training data and disjunctive hypotheses.
- Example applications: Medical Diagnosis and Bank Loan application decisions.

41.1.2 Concept Learning

- Version spaces and Hypothesis Spaces.
  - Subsets $\Rightarrow$ Specificity
  - Super-sets $\Rightarrow$ Generality
  - Intersection $\Rightarrow$ Incomparable
- What are Version Spaces?
  Set of hypotheses consistent with training examples.
- How does the Candidate Elimination algorithm work?
  Representation
  Most general and most specific sets of hypotheses maintained and used to search the space between them.
- Pseudo Code will not be asked for, but know how to describe the working of the algorithm.

41.1.3 Bayesian Learning

- Minimum Decision Length (MDC) and its relationship to Occam's Razor.
- Naive Bayes Learning and Bayesian Belief Networks (know their differences and the advantages of each; When would each be used?)
- Bayes Optimum Classifier - theoretical best classifier possible. Problems with scalability.
  What changes can be made to make it more efficient? Know justification for Gibb's Algorithm - Expected error no worse than twice that of BOC.
- Practical Examples. Know the Newgroups example - What shortcuts were made? (e.g. excluding works which we not topic-specific).

41.1.4 Reinforcement Learning

- Markov Decision Process (MDP); assumptions $S_{t+1} = \delta(S_t, a_t)$ (where t is time, s is state and a is action); $R_t = r(s_t, a_t)$ (R is reward); Goal is to learn policy Π.
- Discount factors
- Deterministic Worlds with absorbing state.
  - Understand Q learning and the training rule used to learn Q.
  - Be able to describe Q values and how they are calculated.
  - In deterministic worlds with infinite repeated games Q converges. Reduce the learning rate α to converge in non-deterministic worlds.