Augsburger and Hoag: Pharmaceutical Dosage Forms: Tablets, Volume 2, Chapter 4

Date Submitted:

Knowledge-Based Systems and Other AI Applications for Tableting

Y. Peng and L.L. Augsburger

Y. Peng
Affiliation: University of Maryland, Baltimore County
Address: 1000 Hilltop Circle, Baltimore, MD 21250
Phone: 410-455-3816
Fax: 410-455-3969
Email: ypeng@csee.umbc.edu

Larry L. Augsburger
Affiliation: University of Maryland School of Pharmacy
Address: 20 N. Pine Street, Baltimore, MD 21201
Phone: 410-706-7615
Fax: 410-706-0346
Email: laugsbur@rx.umaryland.edu

Contact author: Y. Peng

Text pages: 62
References: 73
Tables: 0
Figures: 12
Chapter Outline (main topics only):

Key Words: knowledge-based system, expert system, neural network, decision tree, first order logic, backpropagation learning, Bayesian network, probability theory, fuzzy logic, possibility theory, genetic algorithm, evolutionary computing, hybrid system, support vector machine, semantic web

Pharmaceutical Dosage Forms: Tablets, 3rd Edition, Volume 2

CHAPTER 7: KNOWLEDGE-BASED SYSTEMS AND OTHER AI APPLICATIONS FOR TABLETING

Y. Peng and L. Augsburger

I. INTRODUCTION AND THE SCOPE OF THE CHAPTER

The pharmaceutical industry is under continual pressure to speed up the drug development process, reduce costs, and improve process design. At the same time, FDA's new Process Analytical Technology (PAT) initiatives encourage the building of product quality and the development of meaningful product and process specifications that are ultimately linked to clinical performance. Together, these two issues present significant challenges to formulation and process scientists because of the complex, typically non-linear, relationships that define the impact of multiple formulation and process variables (independent variables) and such outcome responses (dependent variables) as drug release, product stability, and others. The number of variables that must be addressed is substantial and include, for example, the level of drug substance, the types and levels of various excipients, potential drug-excipient interactions, and their potential positive or negative interactions with a host of process variables. Often, the relationships between these variables and responses are not understood well enough to allow precise quantitation. And, since an optimal formulation for one response is not necessarily an optimal formulation for another response, product development is further confounded by the need to optimize a number of responses simultaneously.

Clearly, formulation scientists work in a complex, multidimensional design space. In recent decades, scientists have turned more and more to such tools as multivariate analysis and response surface methodology, knowledge-based systems and other artificial intelligence applications to identify critical formulation and process variables, to develop predictive models, and to facilitate problem solving and decision making in product development. The goal of this chapter is to address artificial intelligence applications and describe their role in supporting formulation and process development.

A *knowledge-based system* (KB) [1, 2, 3] , also known as *expert systems*, is an intelligent computer program that attempts to capture the expertise of experts who have knowledge and experience in a specific domain or area (e.g., granulation). A KB system is designed to simulate the expert's problem solving process or to achieve problem solving to the level similar to or better than domain experts. The use of KB systems in support of formulation or process development is relatively new in pharmaceutical technology, with applications appearing around the mid-1980s. Among these pharmaceutical applications are KB systems for formulating tablets and capsules, process troubleshooting, and the selection of equipment. Such systems have the potential to shorten development time and simplify formulations. Moreover, KB systems can provide the rationale for the decisions taken, serve as a teaching tool for novices, and accumulate and preserve the knowledge and experience of experts. However, KB systems suffer from the limitation that they literally are not creative. That is, they can deal only with situations that have been anticipated in the program.

A *neural network* (NN) [3, 4, 5] is a computer program that attempts to simulate certain functions of the biological brain, such as learning, abstracting from experience, or

generalizing. Designed to discern relationships or patterns in response to exposure to facts (i.e., "learning"), the models developed through a NN may be viewed simply as multiple non-linear regression models. NNs thus enable data developed in the laboratory to be transformed into pattern recognition models for a specific domain, such as tableting or granulation, which would make it possible for formulators to generalize for future cases within certain limits. One limitation of NN is that the effectiveness of a model is limited by the training data itself. Another limitation is that in most cases, NNs lack explanation capabilities, making it difficult or impossible to obtain a justification for the results. Although they have been used in other applications for more than 50 years, NNs have only been applied to pharmaceutical development since the early 1990s. Over the past 15 years or so, NNs have demonstrated a substantial applicability in a number of product development situations, such as predicting granulation and tablet characteristics and predicting drug release from immediate release formulations and controlled release formulations. The development of hybrid systems that integrate NNs and knowledge-based systems potentially can take advantage of the strengths of both NNs and knowledge-based systems while avoiding the weaknesses of either.

In the sections that follow, we will discuss the design of knowledge-based systems, neural networks, and other artificial intelligence systems, and demonstrate their practical application to product development. The focus will be on oral solid dosage forms in general and on tablets in particular.

II. KNOWLEDGE-BASED (KB) SYSTEMS

Knowledge-based systems are intelligent systems that explicitly encode, store, and make use of domain knowledge in problem-solving. Knowledge-based system, when they first

appeared in the early 1970s, were often called "expert systems" because either the domain knowledge they had was a direct encoding of the expertise of domain experts or their performances reached the level of human experts. For example, MYCIN, a medical diagnostic expert system developed at Stanford University in the 1970s, was able to make correct diagnoses for blood infections with the accuracy comparable to physicians experienced in infectious diseases [6]. As depicted in Fig. 1 below, a typical knowledge-based system has two major components, the *knowledge-base* (KB) where the encoded domain knowledge is stored and the *inference engine* which uses the knowledge in KB to draw new conclusions or to initiate new actions based on the case input and according to certain inference rules. Some KB systems also have a *learning* component, which learns new knowledge or revise existing knowledge based on case data, sometimes with the help of feedback on the inference results.

Figure 1 about here

The defining feature of KB systems is how domain knowledge is represented in the KB. The issue of knowledge representation (KR) includes both the *syntax* of the language in which the knowledge is encoded and the language's *semantics* which connects the encoded knowledge with the real world objects it is intended to represent. Moreover, KR is closely related to the inference engine of the system; each type of KR often requires its own set of inference rules and the reasoning procedure for using these rules.

Most modern KB systems are based on formal logics, more specifically on the type of logic known as *first order predicate logic* or first order logic (FOL), a formal system for

deductive reasoning. Therefore this section will start with a brief introduction to FOL before getting into specific KB systems. As representatives, we have chosen to cover only two types of KB systems, rule-based systems and decision trees, for their relative maturity and their popularity in practical applications.

II.1 FIRST ORDER LOGIC

First order logic (FOL) formalizes deductive reasoning [1]. It models classes of objects and their properties by a type of special functions known as *predicate*. Each predicate has a name and a list of arguments. For example, predicate Human(x) stands for the class of humans and Red(y) for things that have color red. For any particular object, a predicate can only have one of the two values, *True* (1) and *False* (0), depending on whether the object is an instance of that class. For example, Human(Confucius) = True and Human(Tweety) = False. More complex expressions or sentences can be formed by connecting predicates with logical operators such as *And* ($\wedge$), *Or* ($\vee$), *Not* ($\neg$), and *If-then* ($\rightarrow$)[1]. Special means are provided for stating whether a statement is true for all objects or only for some; they are *universal quantifier* ($\forall$) and *existential quantifier* ($\exists$). With some syntactic rules, one can write FOL sentences articulating the meaning of often ambiguous English sentences. For example, "All humans are mortal" can be written as

$$\forall x\, Human(x) \rightarrow Mortal(x)\,.$$

"Not all roses are red" can be written as either

$$\neg\forall x\, Rose(x) \rightarrow Red(x)\,, \text{ or alternatively } \exists x\, Rose(x) \wedge \neg Red(x)\,.$$

[1] These operators are also known in logic literature as conjunction, disjunction, negation, and implication, respectively. There are other logical operators, which are less popular and can be expressed by the operators listed here.

FOL uses deductive rules to derive new sentences representing new conclusions. For example, if knowing "All humans are mortal" (the major premise) and "Confucius is a human" (the minor premise), then one can draw a new conclusion according to the syllogism of deduction that "Confucius is mortal". This in the formal system of FOL can be done as follows:

(1) $\forall x\, Human(x) \rightarrow Mortal(x)$

(2) $Human(Confucius)$

(3) $Human(Confucius) \rightarrow Mortal(Confucius)$

(4) $Mortal(Confucius)$

where the new sentence at step (3) comes from (1) by the rule of *Universal Instantiation*, the final conclusion at step (4) from (2) and (3) by the rule of *modus ponens*.

Techniques have been developed to support automatic deductive reasoning. The most noted technique is the *resolution* rule, a single rule that replaces all other deductive rules such as "universal instantiation" and "modus ponens" if the FOL sentences are transformed into the disjunctive normal form[2].

FOL-based intelligent systems solve problems by deductive proofs. To use such a system, one first encodes the domain knowledge (e.g., "All humans are mortal") in FOL sentences and stores them in the knowledge base. Then the goal of the problem solving (e.g., to show "Confucius is mortal") is posted as a theorem or query (also in FOL sentences). The system's inference engine (a deductive reasoner) is then trying to

[2] Any FOL sentence can be transformed into a disjunctive normal form, which is a conjunction of disjunctions. A disjunction, called a clause, can be written either as a disjunction of literals, for example, $(\neg a \vee \neg b \vee c)$, or as an implication $(a \wedge b \rightarrow c)$.

automatically prove this theorem from the given case specific input (e.g., “Confucius is a human”) using the knowledge in the KB.

FOL is very powerful in terms of expressing precisely the domain knowledge. Furthermore, it has been established that if the theorem is indeed true, the system will prove it in a finite number of steps. However, this great expressiveness comes with a price. First of all, automatic deduction is very expensive because it is in essence a search process to find a particular sequence of deductions leading to the theorem among a huge number of possible deductive sequences without much of guidance. To make things even worse, the search process may proceed indefinitely if the theorem is in fact not true. This so-called semi-decidable problem happens rarely in practical applications, but it cannot be avoided completely, as shown by Gödel's incompleteness theorem.

The rigidity of the syntax and semantics of the language also cause problems. First, it is not always easy or even appropriate to encode knowledge in FOL sentences since not every piece of knowledge one knows is logical. For example, it is difficult to represent uncertain relations which are often measured by numerical values (e.g., 80% of flu patients have sore throat) and to represent actions (e.g., if the pressure in the container is higher than 100 then set off the alarm). Secondly, it is difficult to learn domain knowledge in the form of FOL sentences from case data except some simple relations. Finally, FOL is difficult to use for those who do not have training in logic or AI.

II.2 RULE-BASED SYSTEMS

Rule-based systems are probably the most widely used knowledge-based systems in real world applications, and most expert systems referred to in the literature are rule-based systems [2, 3]. As can be seen shortly, this type of system is very close to FOL systems.

The great practicality of rule-based systems comes from relaxation of the rigidity of FOL and adaptation of some extra-logical heuristics.

In many application domains it is very natural for people to express their knowledge and experience in the form of "**if** x **then** y". This is what we take to express rules in rule-based systems. More precisely, a rule has the form of

C1, C2, … Cn => A1, A2 …Am.

where C1, C2, … Cn are the conditions, and A1, A2 …Am are consequents which can be either new assertions or actions. This rule can be read as "If C1, C2, … Cn are ALL true for the current case then take the actions of A1, A2 …Am". The following is an actual example rule written in CLIPS, a popular language for defining rule-base systems:

```
(defrule determine-gas-level
        (working-state engine does-not-start)
        (rotation-state engine rotates)
        (maintenance-state engine recent)
    => (assert (repair "Add gas."))).
```

Here the reserved word "defrule" indicates that this paragraph defines a rule named "determine-gas-level". This rule has three conditions, each of which can be understood as an "attribute/value" pair (e.g., the attribute "engine's working state" has the value "does not start"). Note that these conditions can also be viewed as predicates of FOL. The consequent part is a repair action of "Add gas".

The next example rule was taken from a hybrid intelligent system for the formulation of BCS Class II drugs in hard gelatin capsules [7]:

```
bcs_Class(Id, 2) :- dose_value/Sol_value>250,
                    Perm_value > 0.0004.
```

This rule, written in Prolog, says that a drug with the given "Id" belongs to BCS Class II if the ratio of its dose and solubility is greater than 250 and its permeability is greater than 0.0004.

The knowledge base of a rule-based system is called the Rule Base (RB) where the rules are stored. The case specific input data is given as a list of assertions about the case which are also in the form of predicates or attribute/value pairs. These assertions are put in the Working Memory (WM) and are referred as WM elements. The inference starts with an attempt to match the WM with the condition part of any rules in the RM. If a match is found, that is the current WM can make all conditions in that rule true, then this rule is considered applicable to this case (or can be *fired*). Firing a rule may cause changes to WM (remove/add/change some elements there), and the *match-fire* process repeats with the new WM.

It is often the case that a WM may match more than one rule. Rule-based systems adopt some heuristics to select one of the matched rules to fire at the time. This makes the inference process a depth-first search. When the search reaches a dead end (where the WM cannot match any rule) or it goes too far along a path, the inference engine back tracks for other paths.

The reasoning process described here is called *forward chaining* because it follows the direction of the arrow (from conditions to consequents) when rules are used. It is also called data driven because the process starts with the input data in WM and can potentially derive all consequents implied by the input data. This kind of inference mode is suitable for applications such as monitoring a patient or a nuclear power plant and deciding appropriate actions to take based on the monitors' readings.

One problem for this forward chaining reasoning is its lack of attention. The search space (the set of all consequents derivable from the input data) is in general very large, and many consequents derived may be completely unrelated to the goal of the problem solving. To ease this problem, a different procedure, called *backward chaining*, was developed. In contrast to forward chaining, backward chaining starts with the goal G one wants to establish, and it tries to match G with the consequent side of any rules in the RB. Suppose a match is found with the rule C1, C2, C3 => G. From this rule we know that to show G is true we only need to show that C1, C2 and C3 are true. In other words, the goal G is replaced by three sub-goals C1, C2, and C3. We then repeat this process for each of these sub-goals until a true fact (either in the RB or in the case input) is reached in each thread. Since this kind of inference starts with the goal and proceeds with sub-goals, it is also called *goal driven*. As an example, according to the rule given earlier, the query of whether a given drug belongs to BCS Class II in backward chaining reasoning will be reduced to two subqueries:

```
dose_value/Sol_value>250:-
Perm_value > 0.0004:-
```

To achieve efficiency, rule-based systems circumvent some theoretical difficulties of FOL by heuristics. One such heuristic is that if we fail to establish A, then we treat A as false. With this so-called “negation as failure”, the semi-decidability problem of FOL is avoided. The drawback of adopting these heuristic provisions is that we cannot define a formal semantics for the system, the connection between the rules and the real world relies on the understanding between the system designer and the user. Consequently, the inference result is not guaranteed to be true as is with FOL.

Generality and expressiveness are also sacrificed for efficiency. For example, all variables in the rules are assumed universally quantified, there is no way to express existential qualification, and the predicates on either side of the arrow are restricted to be conjunctions (AND relations). Some subtle relations expressible in FOL may not be expressed in rules. For example, we may write a rule "if someone is the father of a human then he must also be a human" as

Father(x, y), Human(y) => Human(x).

However, it is difficult, if not impossible, to write a rule for "every human must have a human father" because existential quantification is needed here[3].

Similar to FOL, it is difficult to learn rules from data or to associate uncertainty with rules. Research has been conducted, sometimes extensively, on these issues in the past, and many approaches and methods have been proposed and experimented [1]. However, none has received wide acceptance by AI practitioners.

II.3. DECISION TREES

Fig. 2 below depicts a decision tree for a simple classification task: classify given objects into two groups, labeled + and –, respectively. The classification is according to three attributes of each object: the shape (square or round), size (big or small) and color (green, red, or blue). Instead of evaluating all attributes at the same time, the decision tree does the classification through a sequence of decisions, each of which is based on a single attribute. Each decision is represented by a non-leaf node in the tree, called *decision node*. Branches of a decision node correspond to possible values for that attribute. Leaf nodes of the tree are class nodes with the class labels.

[3] This can be easily written as a FOL sentence $\forall x\, Human(x) \rightarrow \exists y\, Father(y, x) \wedge Human(y)$.

Figure 2 about here

For example, to classify a big red square object, we start with the root (top), which makes decision according to the shape of the object. Since the object is square, we go down to the left branch and proceed to make the second decision based on the size. The process eventually leads to the second leaf node from the left at the bottom and we conclude that the given object belongs to class "+".

To construct a decision tree, one can start by selecting an attribute for the root, and its branches are determined by the attribute. The process is then repeated for each of the children of the root, and so on. One of the objectives for tree construction is to make the tree short (so that later on the decisions can be made fast). The attributes for decision nodes can be selected by experts based on their experiences and their understanding of the domain. They can also be learned from sample data. Here each sample is about one object, including its values for all of the attributes and the class label this object belongs to. For example, a sample for the tree of Figure 2 may look like

(big, blue, square, "-").

The label for each sample can be obtained from observation or assigned by humans. Using training samples with known class labels makes decision tree learning a supervised learning.

Among the many proposed methods for decision tree learning, the one that is most widely recognized is the ID3 algorithm by Quinlan [9]. ID3 is based on the notion of information gain when selecting attributes: choose the one that has the largest expected

information gain. For a group of training samples T, the information gain of partitioning T based on attribute X is measured by

$$Gain(X, T) = Info(T) – Info(X, T).$$

Here Info(T) is measured by the entropy of the T's probability distribution over the classes. To measure Info(X, T), we first partition T by X, then calculate the entropy for each subset Ti in the partition, and finally add these entropies together, each weighted by the size of Ti. Selecting the attribute that gives the maximum information gain guarantees to result in the smallest expected size of the tree. Fig. 3 bellow presents an example of decision tree learning. The table on the left of the figure contains 12 samples and their class labels. The tree on the right is learned using these 12 samples by ID3 algorithm. Comparing with the tree in Fig. 2, which also correctly classifies all of the 12 samples, the tree in Fig. 3 is much smaller (10 nodes vs. 13 nodes) and shorter (average height of leaf nodes of 2.166 vs. 2.75).

Figure 3 about here

Note that the decision at each node is simple and uses only the information local to that decision (e.g., the root of the tree in Fig. 3 only cares about the color of the object without concerning with its shape and size), and that the decisions are irrevocable. These are the main reasons for its computational efficiency. Also note that this strategy of decomposing a large decision into a sequence of small decisions is taken by people everyday in dealing with complex problems. For this reason, people often use decision trees as a modeling

tool to capture and mimic human experts' decision process. An example decision tree that models the formulation of BCS Class II drugs in hard gelatin capsules can be found in [7].

II.4 LANGUAGES AND TOOLS

Many tool sets are available, both commercially and in public domain, to support the various knowledge-based systems, including those reviewed in this section. As mentioned earlier, each knowledge representation paradigm is associated with its own inference mechanism, so these tools usually include a language for encoding the domain knowledge in the given KR and an inference engine. The tool will construct the knowledge base from the encoded knowledge, and the inference engine will be evoked by either the case-specific input data (for forward chaining) or the goal to be achieved (for backward chaining) or both. Many tools come with graphic interface to help interacting with the user.

CLIPS and Jess

Early tools are so-called "expert system shells" such as EMYCIN (from Stanford University) and OPS5 (from Carnegie Melon University), which, as the term implies, came from real expert systems. For example, EMYCIN is the shell of the blood infectious disease expert system MYCIN. It retains everything of MYCIN except the content of the KB. To build a new expert system for some other application (say car diagnosis), one can simply fill the KB with domain knowledge encoded in MYCIN's language.

OPS5, a forward chaining rule-based system language, was further developed into CLIPS (C Language Integrated Production System) at NASA [10]. CLIPS and its later version in Java named Jess (Java Expert System Shell) [11], developed at Sandia National Lab, are probably the most widely used tools for constructing and running forward chaining rule-

based systems. Both CLIPS and Jess are in public domain and can be downloaded from a number of websites[4].

Prolog

Prolog, standing for "programming in logic", is a language that implements a subset of FOL. Sentences in Prolog are restricted to *Horn clauses*. A Horn clause is a disjunction of literals in which at most one literal is positive. Prolog is quite strong in its expressing power, it can be used to represent almost all we want to express in most applications. For example, a fact that John is a male can be written as

```
:-Male(John)
```

where "`:-`" is for logic operator "implication". We can also represent the rule that "if x is a parent of y and x is male then x is the father of y" as

```
Father(x, y):-Parent(x, y), Male(x);
```

and goals we want to prove as

```
Father(John, Bill):-
```

and so on. What are not allowed in Prolog are those disjunctive clauses with more than one positive (e.g., those on the left side of the implication) such as

```
Father(x, y) ∨ Mother(x, y):-Parent(x, y)
```

or

```
Father(John, x) ∨ Mother(Mary, y).
```

Prolog systems also adopt some extra-logical provisions for efficiency and convenience. For example, the search for the solution is done by depth-first search plus backtracking, and "negation as failure" is adopted for circumventing semi-decidability problem. Most Prolog systems conduct logical reasoning in the backward chaining fashion, making them

[4] For example, http://www.ghg.net/clips/CLIPS.html and http://herzberg.ca.sandia.gov/jess/

popular tools for constructing backward chaining rule-based systems. Recently, forward chaining Prolog systems also have been developed (e.g., XSB [12]).

Logica's PFES

Product Formulation Expert System (PFES) was developed by Logica as a reusable software kernel to support a generic formulation task [13] in a number of industrial sectors, especially in pharmaceuticals. It was designed to speed up the selection of product ingredients, and the subsequent testing, analysis, and adjustment formulation procedures. Like CLIPS, PFES also uses exclusive forward chaining in the inference. An example of PFES application to tablet formulation can be found in [14]

Decision Trees

Because both the structure and its inference logic of decision tree are relatively simple, one can afford to implement a decision tree in a number of ways. It can be coded directly with any general purpose programming language such as C, C++, Java, or LISP (a primary AI programming language). It can also be implemented using expert system shells. For example, the Capsugel expert system, which is a decision tree in logic, was first implemented in C [15], and later re-implemented in SICStus Prolog (a Prolog system developed by Swedish Institute of Computer Science) for added flexibility to introduce additional rules [8].

If the purpose is to learn a decision tree from a collection of labeled samples, then the best available tool is probably a software package called C4.5 [9]. The core of C4.5 is the ID3 algorithm described earlier. It extends the basic ID3 learning with capabilities 1) to handle missing values in training samples; 2) to accommodate attributes with continuous

value ranges; 3) to prune the learned decision trees; and 4) to avoid *overfitting*[5] . It is also able to derive implication-like rules from the learned tree.

III. NEURAL NETWORKS AND NEURAL COMPUTING

The logic-based approach of knowledge-based intelligent systems was inspired by *high* level human reasoning and cognition activities, and it attempts to model such activities in a formal way. In contrast, neural networks take a different approach in solving complex problems typically requiring human intelligence. This approach attempts to model the low level activities of the nerve systems in human and animal brain. The origin of the present day neural networks can be traced back to Pitts and McCulloch's 1943 model of biological neurons [16], which can be shown to be able to realize all Boolean functions. Hebbian's rule [17], a simple rule proposed in 1949 for modifying synaptic strengths in a nerve system, is also very influential in learning methods for various neural network models.

III.1. OVERVIEW OF NEURAL NETWORKS

In essence, a neural network as a computational model can be viewed as the following. The network has a large number of nodes connected by weighted links. To some extent, one can view a neural network as a simplification of a biological nerve system where nodes correspond to neurons and weighted links to synaptic strengths between neurons. Each node has certain activation level and can send its activation as output to other nodes that are connected to it. It can also receive activations from other nodes and update its own activation according to certain rules or functions. This kind of interactive activities

[5] Overfitting refers to a common problem for machine learning that the learned model fits the training data very well but performs poorly with previously unseen data.

between nodes may be triggered by certain external input; the interaction continues until a stable state is reached over the network. At this time the pattern of activations over the network of nodes provides a solution to the problem.

Next we briefly describe the main components of NN [4].

Nodes

A node in a NN has one or more inputs from other nodes, and one output to other nodes, the values for its input and output can be binary (0 or 1), bipolar (-1 or 1), or continuous (either bounded or unbounded). The output represents the current activation level of the node and it is determined by the inputs and the activation function (also called node function) associated with that node. Typically, as illustrated in Fig. 4, the activation function takes the weighted sum of the inputs from other nodes as its input and computes the node activation (output) by a simple mathematical function *f*.

Figure 4 about here

Nodes with nonlinear node function play crucial roles in neural computation. Commonly used nonlinear functions include step functions, sigmoid functions, and Gaussian functions. The input to the function is $x = w_1 x_1 + \ldots + w_n x_n$, the weighted sum of node inputs, where w_i is the weight associated with the input x_i, and the function value $y = f(x)$ is the node output.

A *Step function* (also known as a *threshold function*) is a binary function with only two possible function values (or two states). Which of the two values will be the output

depends on whether x is below or above the given threshold. For example, as depicted in Fig. 5 (a)

$$y = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

is a step function with threshold = 0. A variation of the threshold function is the "Ramp function", as shown in Fig. 5 (b), it provides a linear transition region between the two states.

Sigmoid function. One limitation of the step and ramp activation functions is that they are not everywhere differentiable, making mathematical analysis of NN models using such node functions very hard. Sigmoid (S-shape) functions overcome this difficulty by approximating the shape of the step/ramp functions with differentiable ones. There are a few candidates for sigmoid functions, the two most widely used ones are

- Logistic function: for example $y = \frac{1}{1+e^{-cx}}$ where c is a constant called slope.
- Hyperbolic tangent function: $y = \frac{e^{cx} - e^{-cx}}{e^{cx} + e^{-cx}}$.

As depicted in Fig. 5 (c), a logistic function is rotationally symmetric about the point (0, 0.5), and it asymptotically approaches the two extreme values with x of great magnitude (to 1 when $x \to \infty$ and 0 when $x \to -\infty$). Also note that change of x will cause large change in y when the magnitude of x is small (e.g., $|x| << 1$), and cause little change in y when $|x| >> 1$. In the latter case, we say that the function moves into a saturation region, where further increases of the magnitude of x would have not effect on the output of the function. The shape of the function curve is related to the slop c, smaller c yields flatter

curve and larger *c* leads to steeper curve, and when *c* is really very large, the logistic function approaches the threshold function.

The hyperbolic tangent function has the same properties as the logistic function except that its two extreme values are -1 and 1.

Another non-linear function with significant applications is the *Gaussian function*. Its curve has a bell shape, the output takes the maximum value at the center and approaches zero when the distance to the center goes to infinity.

Figure 5 about here

Links and Link Strengths

As mentioned earlier, individual nodes in NN have very limited computing power because their node functions are very simple. Despite of this, NN have been shown to possess great computing power, capable of solving many difficult problems. This power comes from the richness of the connectivity of the networks. Put in another way, while the KB systems encodes its problem-solving knowledge in the logical sentences and rules in the knowledge base, the knowledge in NN is capture by the inter-node connections and the associated connection strengths.

Links have directions, the weights on the links from node A to B and from B to A may have different values and even different signs. The weights can be discrete (binary, bipolar or other integer values) or real values. There are three kinds of nodes, depending on whether the node's input and output links are within the network or not. They are the input nodes (those that receive external input from the environment); output nodes (those

that present the output to the environment) and *hidden nodes* (those that do not have any interaction to the environment). Note that input and output nodes may overlap, but not with hidden nodes.

Inter-node connections define the architecture (or structure or topology) of a NN. Different NN models are developed for different types of applications, which differ with each other often on their architectures. Here are some widely used NN architectures.

Fully connected NN. Every node is connected by a link to every other node (including itself). One renowned example of this architecture is the Hopfield model, widely used as a basis for various NN models for associative memories and optimization. A fully connected network with randomly generated weights can be viewed as a model of total ignorance, and thus can be used as the starting network for learning.

Recurrent NN. A network not necessarily fully connected but containing at least one directed cycle. Therefore, a node can influence itself via the cycle, and the network forms a dynamic system. Mathematical analysis of recurrent networks is often complicated.

Acyclic NN. A network without a directed cycle. This type of network is easier to analysis than recurrent networks.

Layered NN. Nodes in a layered NN are grouped into layers, two nodes are connected only if they are either in the same layer or in adjacent layers.

Two-layer recurrent NN. There is no intra-layer connection, and nodes between the two layers are often fully connected. AS a dynamic system, outputs of nodes in one layer become inputs to the nodes in the other layer, and the interaction takes iterations to reach equilibrium, a state in which no node will change its activation. Example NN models with this kind of architecture include bidirectional associative memories in which

patterns in one layer can be recalled by patterns presented to the other layer, and Self-organizing maps (SOP) that can be trained so that the topological relations existing in the input layer is preserved in the output layer.

Multi-layer feedforward NN. A network that is both acyclic and layered (with at least two layers, not counting the input layer). In addition, there is no connection between nodes in the same layer. This architecture is the basis for the most widely used NN model in practical applications, the celebrated *backpropagation* model, which we will give a much more thorough coverage short.

Neural Network Learning

One of the noted strengths of neural networks is their ability to learn problem-solving knowledge from the sample data. What makes learning relatively straightforward is that learning in NN is basically a process of modifying the connection strengths by repeated presentations of training samples. Learning in most NN models is kind of a variation of the Hebbian learning rule, which says the strength between nodes A and B shall be increased if both A and B are excited (both are positive) when the given training sample is presented to the network.

One type of learning is called *supervised* when each training sample include both the input pattern describing a problem and the desired or target pattern representing the correct solution to the problem with the given input. In other words, the learning is seen as being supervised by a teacher, who for each input pattern provides the desired output pattern. During the training, the input pattern of a sample is presented to the input nodes, then the network's internal computation generates an output based on its current

connection weights. This output is compared with the desired output, the difference then drives a modification to the current weights in the network.

In contrast, *unsupervised* learning learns associations and regularities from training samples without the benefit of answers or even any hints of correct answers from the teacher. The third type of learning, the *reinforcement* learning, is in between of these two. Similar to unsupervised learning, each training sample for reinforcement learning contains only the input pattern, not the desired output. When an input pattern is presented, the computed output is fed to a judge or arbitrator which will provides a feedback of either this output is good (and the system is awarded, say, keeping the current weights unchanged) or bad (and the system is punished by requesting a change of the weights). The difficulty here is, when change is called for, one has to figure out which of the weights shall be changed and how much the change should be.

III.2 BACKPROPAGATION NETWORKS

The name of the Backpropagation (BP) network comes from its error *backpropagation* learning algorithm [4, 18). Due to its popularity in real world applications, many people take BP networks as the synonym of neural networks. BP networks also find a variety of applications in the area of drug formulation [3].

BP Network Architecture

As mentioned earlier, a BP network is a multi-layer feedforward network. In addition, it must have at least one layer of nonlinear hidden nodes with sigmoid node functions[6]. Fig. 6 below depicts a two layer BP network. Note that the input layer is not counted here

[6] Some people refer BP network as multi-layer perceptron for historical reasons because it is a generalization of a famed early neural network model *Perceptron*. Strictly speaking, a multi-layer perceptrons use threshold node functions, not sigmoid ones.

because nodes in input layer are not processing units, they are merely place holders for the external input without performing any computation. Most of the discussions in this subsection are based on two layer networks (with only one hidden layer), the key results can be easily generalized to networks with more than one hidden layer.

Figure 6 about here

We adopt the following convention for notations. Values of all nodes on each of the layers form a vector, we denote the vectors on input, hidden, and output layers $x = (x_1, \ldots, x_i, \ldots, x_n)$, $x^{(1)} = (x_1^{(1)}, \ldots, x_j^{(1)}, , \ldots, x_m^{(1)})$, and $o = (o_1, \ldots, o_k, \ldots, o_l)$, respectively. We denote the two weight matrices as $W^{(1)}$ (from input to hidden) and $W^{(2)}$ (from hidden to output). Each weight matrix is a set of weight vectors, one for each node, so for example, $W^{(1)} = (w_1^{(1)}, \ldots, w_j^{(1)}, \ldots, w_m^{(1)})$, and the weight vector $w_j^{(1)} = (w_{j,1}^{(1)}, \ldots, w_{j,i}^{(1)}, \ldots, w_{j,n}^{(1)})$ is a collection of weights from each of the input nodes to hidden node *j*.

The computation in a BP network is simple and straightforward. When an input pattern or vector $x = (x_i, \ldots, x_n)$ is presented to the input layer, it is passed through input nodes to the hidden layer. Each hidden node computes its output value by

$$(1) \qquad x_j^{(1)} = S(\sum_i w_{j,i}^{(1)} x_j),$$

where *S*(.) denotes the sigmoid function. Taking $x_j^{(1)}$ from all hidden nodes as inputs, each output node computes its output in a similar fashion

$$(2) \qquad o_k = S(\sum_j w_{k,j}^{(2)} x_j^{(1)}).$$

General Function Approximator

It is clear that a BP network defines a multi-variant function $o = f(x)$, for an given input vector x, the function value of f is computed according to Eqs. (1) and (2). Changes of weights in $W^{(1)}$ and $W^{(2)}$ will change function f. An interesting question is, what kinds of mathematical functions a BP network can compute, or put it in another way, for an arbitrary mathematical function *F*, does there exist a set of weights so that $f(x) = F(x)$ for all inputs *x*. It has been proven mathematically that feedforward networks with at least one hidden layer of non-linear nodes are able to approximate any L_2 functions (all square-integral functions, including almost all commonly used mathematical functions) to any given degree of accuracy, provided there are sufficient many hidden nodes. In this sense, BP networks are called General function approximators. This representational power of BP networks lies primarily on the nonlinearity of the hidden nodes. Nonlinear output nodes alone cannot play the trick, as has been shown that a perceptron (a single nonlinear output node with weighted links from inputs) cannot solve problems that are not linearly separable (for example weights can be found for a perceptron to solve logical functions AND and OR, but not Exclusive-Or). It can be shown easily that adding linear hidden nodes, no matter how many layers, to a perceptron does not increase its computing power.

Knowing the representation power of BP networks is only half of the story, a follow up question is, for a given (L_2) function *F*, how can we construct a feedforward network and find a set of weights so that the network approximates *F* well? Brutal force search for the weights is computationally intractable because the search space (of all possible weights)

is a multi-dimensional continues one. Backpropagation algorithm is a learning algorithm that can quickly find a good set of weights from a set of training samples.

BP algorithm is an example of supervised learning. A training sample therefore consists of two parts: an input pattern $x_p = (x_{p,i}, \ldots, x_{p,n})$ and its desired output pattern $o_p = (o_{p,i}, \ldots, o_{p,l})$. From the function approximation point of view, we can think that the set of *P* samples are taken from a unknown function *F*, i.e., for each x_p, $o_p = F(x_p)$.

BP algorithm is also an example of error driven learning. For each x_p, we can compute the output pattern o_p based on the current weights. Then the learning is to be guided by the difference between the desired and the actual outputs: $\delta_p = d_p - o_p$ in vector notation and $\delta_{p,k} = d_{p,k} - o_{p,k}$ for individual output nodes. The general principle is that we want to modify the weights in such a way that the error $\delta_{p,k}$ gets reduced. To see this intuitively, consider $w^{(2)}_{k,j}$ in $W^{(2)}$ (from hidden node *j* to output node *k*) and $w^{(1)}_{j,i}$ in $W^{(1)}$ (from input node *i* to hidden node *j*) in Fig. 7 below.

Figure 7 about here

It is straightforward to see how $w^{(2)}_{k,j}$ should be changed with the error $\delta_{p,k} = d_{p,k} - o_{p,k}$. If both $\delta_{p,k}$ and $x^{(1)}_j$ are positive then we increase $w^{(2)}_{k,j}$ (which increases $o_{p,k}$ and in turn decreases $\delta_{p,k}$). The same goes when both $\delta_{p,k}$ and $x^{(1)}_j$ are negative. On the other hand, $w^{(2)}_{k,j}$ should be reduced when the signs of $\delta_{p,k}$ and $x^{(1)}_j$ are different. The update for $w^{(2)}_{k,j}$ outlined here is seen clearly an application of Hebbian rule.

It is not so easy for updating weight $w_{j,i}^{(1)}$ because we do not have the desired output value for hidden node *j* and thus cannot directly compute the error for node *j*. The novice idea behind BP learning is its way to compute the error for hidden nodes. Since $w_{j,i}^{(1)}$ influences $x_j^{(1)}$ (Eq. (1)), and $x_j^{(1)}$ is taken as input by all output nodes (see Fig. 7), $w_{j,i}^{(1)}$ affects errors $\delta_{p,k} = d_{p,k} - o_{p,k}$ for all output nodes and thus its update should be determined by all of these errors. Specifically, BP algorithm calculates the error for hidden node *j* as a weighted sum of errors on all output nodes.

The actual weight update rules for BP learning are derived following the mathematical approach known as gradient descent. This approach determines the change to each weight in isolation (as if all other weights remain unchanged) and along the direction that maximizes the reduction to the total error $E_p = \sum_k \delta_{p,k} = \sum_k (d_{p,k} - o_{p,k})$. Specifically, for each weight *w* (either in $W^{(1)}$ or $W^{(2)}$), the change, Δw is determined as

$$(3)\ \Delta w = -\eta \frac{\partial E}{\partial w} = -\eta \frac{\partial}{\partial w} \sum_k (d_{p,k} - o_{p,k}),$$

i.e., the change to *w* is proportional and negative (thus the name of gradient *descent*) of the partial derivative of *E*. Here η in (3) is a constant known as the *learning rate*, which determines the size of changes at each step of learning. Partial derivatives $\partial E / \partial w$ for individual weights can be derived from (3) since *E* is a function of these weights (see Eqs. (1) and (2)). Specifically, let $net_l^{(.)}$ denote the total weighted input to node *l*, we have

$$(4)\ \Delta w_{k,j}^{(2)} = \eta \cdot \mu_k^{(2)} \cdot x_j^{(1)}$$

for all weights in $W^{(2)}$, where

$$(5)\ \mu_k^{(2)} = (d_{p,k} - o_{p,k}) S'(net_k^{(2)})$$

is the error term on output node *k* and $S'(net_k^{(2)})$ is the derivative of its activation function. And for weights in $W^{(1)}$, we have

$$(6)\ \Delta w_{j,i}^{(1)} = \eta \cdot \mu_j^{(1)} \cdot x_i$$

where

$$(7)\ \ \mu_j^{(1)} = (\sum_k \mu_k^{(2)} w_{k,j}^{(2)}) \cdot S'(net_j^{(1)})$$

is the error term for hidden node *j*, which can be calculated by first ***back propagating*** the errors from the output nodes, weighted with the corresponding weights in $W^{(2)}$, and then multiplying with the derivative of the node's activation function.

The learning process repeats the following steps, starting from an initial set of weights for $W^{(1)}$ and $W^{(2)}$:

1. pick up a training sample (x_p, o_p);
2. calculate the output pattern o_p by Eqs. (1) and (2);
3. calculate errors $\delta_p = d_p - o_p$ at output nodes;
4. update weights in $W^{(1)}$ by Eqs. (4) and (5);
5. update weights in $W^{(2)}$ by Eqs. (6) and (7).

This process continues until all weights stop to change (i.e., the process converges) or other termination criterion is satisfied.

The process outlined above is called a *sequential learning* because training samples are selected one at a time in a sequence and weights are changed per each selected sample. Learning can also be conducted in another mode, known as *batch learning*, which is the same as the sequential learning except that actual weight changes do not occur with each sample, instead, the calculated Δw for each of the *P* samples are cumulated. When all *P*

samples are processed, the cumulated Δw are averaged (over P) and used to make actual changes to the weights.

Properties of BP Learning

It has been proved mathematically that the BP learning always converges if the learning rate is sufficiently small. This is because the gradient descent guarantees that the total error $E = \sum_p E_p$ can only decrease at each step of learning. However, it is not guaranteed to converge to a set of weights that reduces the total error to zero. That is, it is only guaranteed that the learning converges to a *local minimum error state*, i.e., any small change of the learned weights will always cause E to increase. We can compare the gradient descent approach of BP learning with hill-climbing. If one, when climbing a hill, always moves along the steepest direction, he will certainly reaches the top of the hill, which is higher than its immediate vicinity but not necessarily higher than summits of other hills and mountains.

Several features of BP learning make it very attractive to practical applications. First, as discussed earlier, any L_2 function can be represented by a BP network, and in many cases such a network can be trained using BP learning with great accuracy. Secondly, it is fairly easy to apply BP learning to problems at hand. Unlike other formalism such as those logic-based approaches, BP learning does not require substantial prior knowledge or deep understanding of the domain itself, it only requires that a good set of training samples is available. This makes it a powerful modeling tool for ill-structured, ill-understood problems. Thirdly, the implementation of the core BP algorithm is very simple. And finally, like many other NN models, BP learning naturally tolerates noise and missing values in training samples. In most of the cases, noise and missing values

only degrade the learning quality, not lead to a completely wrong model nor disrupt the learning itself (graceful degrading).

On the other hand, BP learning can be frustrating, even when one has a good set of training samples. First, the learning often takes a long time to converge when there are many hidden nodes in the network and the sample set is large. Secondly, there is not much one can do if the learning converges with a large total error E except possibly to re-run the learning with a different set of parameters and initial weights and pray for a better result.

Quite a few proposals have been made to speedup the learning process. For example, one proposal suggests that the weight update rules not only include the terms caused by the error as given in Eqs. (4) and (6) but also the changes of previous steps (called momentum terms). This method avoids sudden change of directions of weight update, smoothens and often speeds up the learning process. Another widely used method is called Quickprob [19]. Instead of slowly approaching the final weights through many iterations of (4) and (6), this method, whenever possible, calculates (by some simple procedure) the weights that are close to a local minimum error state. Other methods speedup the learning by manipulating the learning rates in different ways.

Another problem with BP learning is that what can be learned (i.e., the weights) are merely operational parameters, not general, abstract knowledge of the domain. As such, a trained BP network behaves like a black box, it produces an answer (in the form of the output pattern) for any given problem (as specified by the input pattern) but is not able to explain why the answer is correct or how good this answer is.

Finally, like many learning methods that build models from data, we are facing the problem of overfitting. That is, the trained network may fit the training samples perfectly (i.e., the total error *E* is very close to zero) but it does not produce correct or good outputs for previously unseen inputs. If overfitting happens we say the trained network generalizes poorly. Overfitting problem can be eased by moving the weight matrices slightly away from the local error state. This can be done by adding noise into the sample set or stopping the learning earlier before the minimum error state is reached. The most widely used strategy in dealing with overfitting is known as *cross-validation*. Instead of using all samples for training, this strategy leaves a small portion (say 10%) of them as test data. The learning periodically pauses and checks the error over the test data, and it stops when error over test data starts to increase.

Parameter Selections and Other Practical Concerns

Learning algorithm is only part of the task of implementing BP learning, the other, more subtle part, is how to initialize the network and how to select learning parameters. Since the number of nodes in input and output layers are determined by the problem one intends to solve, so the initialization of the network topology involves only the determination of the number of hidden layers and their size. Theoretically, a single hidden layer is sufficient for any complex problems, however, there is no theoretical result on minimum necessary number of nodes in that hidden layer. The practical rule of thumb is to have twice as many hidden nodes as the input nodes for binary/bipolar data and many more for real value data. It has been reported in the literature that networks of multiple (2 – 4) hidden layers with fewer nodes may be trained faster for similar quality

in some applications. After the hidden layers are decided, the weights for all links in the network are usually set to some small randomly generated initial values.

Besides the network topology, the quality of learning is also depending on the quality and quantity of training samples. The samples should be a good representation of the domain, they should be randomly sampled or guided by the domain knowledge if such knowledge is available. There is no theoretically ideal number for the samples, intuitively this number is dependent of the number of weights in the network and the accuracy desired for the results. Some has suggested the number of samples can be estimated as $|W| / e$ where $|W|$ is the total number of weights in the network and e is the acceptable error bound.

Another important parameter is the learning rate η. The gradient descent requires η be as small as possible, however, too small rate makes the learning extremely slow. Common practice suggests to start with η less than or equal to 1.

Finally, we need to select a criterion for terminating the learning. One obvious criterion is when $E \leq e$ if the acceptable error e is given. This criterion may not always be practical because of the "local minima" discussed earlier. Instead, people often stop the learning when Δw, the weight change, becomes very small for every weight. Finally, one can set a maximum number of iterations for the learning and stop the process when this number is reached.

III.3. OTHER NEURAL NETWORK MODELS

A large number of neural network models have been developed in the past few decades, with different mechanisms and often for different types of applications. Here we list a

few representative NN models for their popularity and potential for pharmaceutical applications.

RBF Networks

RBF network is perhaps the most widely used NN model, after only the backpropagation networks [20]. A RBF network is very similar to the BP network, the main difference is that it uses radial basis function (RBF), not the sigmoid function, as the node function. A typical RBF for this type of networks is the Gaussian function. As can be seen in Fig. 5 (c), the output of a RBF node depends on the distance of the input vector to the vector stored in the node; and the output is maximal if the distance is zero. Similar BP network, RBF network is also a universal function approximator, and can be trained by supervised learning. It has been found that RBF networks often performed better than BP networks in function approximation and classification.

Competitive Learning and Self-Organizing Map

Competitive learning is a kind of unsupervised learning often involves a single layer of output nodes. When a training sample is presented as the input vector to the network, all output nodes *compete* with other, and the node whose weight vector is closest to the input vector wins. The winner then has its weight vector updated (moving further closer to the input vector) while all other output nodes will have their weights unchanged. Competitive learning learns regularity, clustering, similarity among the training data without the supervision of a teacher.

Self-organizing map (SOM) is a special competitive learning network with the aim of preserving the topological order (neighborhood relation) among the training samples [21]. SOM differs from other competitive learning networks on how the weights shall be

updated after the winner is determined for a given training sample. Not only the winner but also its neighboring output nodes will have their weight vectors changed toward the training sample. As the result, when two input vectors that are similar to each other are applied to the trained SOM, the corresponding output nodes will be close to each other, thus the topological order is said to be preserved. SOM model is motivated by sensory maps in biological nerve systems (e.g., retinotopic map) which preserve topological orders, but its applications go far beyond the simulation of biological maps.

Support Vector Machine (SVM)

Single layer NNs have limited computing power. This is demonstrated by the problem of *linear separability*. Suppose we want to build a two class classifier for data points. For some datasets, a linear separator (a line for 2D data and a hyperplane for higher dimensional data) is sufficient to separate the data points in the two classes. For other datasets there is no linear separator, rather the separators must be non-linear [7].

Multi-layer NNs such as BP networks overcome the linear separability problem by including a layer of hidden nodes of non-linear functions. The price paid for the greatly increased computing power is the time it takes to train the network.

Support vector machine (SVM) [22] is a relatively new supervised learning method that overcomes this problem: it is able to learning non-linear separators at a much faster speed. This nice property helps SVM to quickly gain popularity since mid-1990. A full coverage

[7] A well known linear non-separable problem is the logical operation of "Exclusive Or", denoted ⊕. A ⊕ B = true if and only if either A and B are both true or both false. The four possible value assignments of A and B can be represented as four data point (1, 1) (0, 0), (1, 0), and (0, 1) in a 2-dimensional space. Then ⊕ put the four points into two classes, those with truth value 1 ((1, 1) and (0, 0)), and those with truth value 0 ((1, 0) and (0, 1)). It is clear that there is no line on the 2D space that can separate these two classes.

of SVM is beyond the scope of this chapter, readers interested in this method can start from the detailed tutorial by Burger [23]. Roughly speaking, SVB is based on a simple property: if data points are not linearly separable in a given space, then they can become linearly separable if they are mapped into a space of sufficiently higher dimension. Directly finding a linear separator in the high dimensional space (called the feature space *F* of the given data) is time consuming and is in danger of serious overfitting. SVB overcome these as follows. Since finding a separator can be cast as a quadratic programming problem that is based on the inner product of very pair of data points $x_i \cdot x_i$, then it is based on the inner product becomes $F(x_i) \cdot F(x_i)$ for the feature space *F*. SVM does not directly work with $F(x_i) \cdot F(x_i)$ but utilizes some function called *kernel function* that computes $F(x_i) \cdot F(x_i)$ from $x_i \cdot x_i$. An example kernel function is $F(x_i) \cdot F(x_i) = (x_i \cdot x_i)^2$. Efficient learning methods based on kernel functions have been developed and implemented in various SVM packages.

III.4. NEURAL NETWORK DEVELOPEMNT TOOLS

Many dozens of NN development tools have been developed in the past two decades or so. Many of them are in public domain (e.g., DPD++, JavaNNS, SNNS, etc.), others are commercial products (e.g., BrainMaker, NeuralMaker, NeuralShell, etc.). The set of NN models included in each tools package may be quite different, but almost all of them include BP networks. Most of tools in public domain were developed by academic research groups, and they often come with the source code. This allows the users to modify the NN models to their particular needs, and facilitates the integration of a NN as a component into a larger system. Commercial products, on the other hand, usually come

with much better user interface and many auxiliary tools (e.g., statistical analysis procedures, procedures for pre and post processes). Some products offer API (Application programming interface) via which the modules can be accessed and executed by the user's own program. This is very important for users who may need to modify the NN models in the package or integrate them with other programs.

Two NN toolkits are worth specially mentioning. The first is *MATLAB Neural Network Toolbox*[8] from The MathWork, which extends MATLAB "for designing, implementing, visualizing, and simulating neural networks". Since MATLAB itself is a numerical computing environment and a programming language, one can call NN models like any other MATLAB functions, and can easily build interface between NN models and other computing modules written in MATLAB. The second tool is *CAD/Chem* and its successor *INForm* by Intelligensys, which is specialized in formulation modeling and optimization for chemists and product designers and has found wide pharmaceutical applications[9]. Using BP neural networks, CAD/Chem helps the product design by automatically learning the underlying relationships between product ingredients, process parameters and resulting properties. It also provides modules for fuzzy logic and genetic algorithms (which will be introduced shortly in the next section) and statistical analysis tools that are needed for formulation optimization.

IV. OTHER MODELS FOR INTELLIGENT SYSTEMS

Other models, based on different principles and theories, have been developed for building intelligent systems. In this section we briefly introduce a few of them, the

[8] http://www.mathworks.com/products/neuralnet/.
[9] http://www.intelligensys.co.uk/models/inform.htm.

Bayesian networks, *fuzzy logic*, and *evolutionary computing*. These models have quite different characteristics than the logic-based systems and neural computing, and they all have found a wide range of applications.

IV.1. BAYESIAN NETWORKS

Bayesian networks (BN), also called Bayesian belief networks, belief networks, or probabilistic causal networks, are a widely used mathematical model for knowledge representation and reasoning under uncertainty [24. In this graphical model, nodes represent random variables and the probabilistic interdependencies between random variables are represented by their interconnections. The joint probability distribution of these variables is decomposed into a set of conditional probability tables, one for each of these variables.

Formally, a BN of *n* variables $X = \{X_1,..., X_n\}$ is a directed acyclic graph (DAG) of *n* nodes and a set of directed arcs, with conditional probability tables (CPT) attached to each of the *n* nodes. Nodes correspond to random variables, denoted X_i $(i = 1, \cdots, n)$. Each variable is associated with a finite set of mutually exclusive states. The lower case x_i denotes an instantiation of X_i to a particular state, and $x = \{x_1,..., x_n\}$ represents a joint assignment or an instantiation to all variables in X. An arc $< X_i, X_j >$, represents a direct causal or influential relation from X_i to X_j. This arc also indicates that X_i and X_j are probabilistically dependent of each other. The quantitative part of the interdependence is modeled by the CPT $P(X_i \mid \pi_i)$ of each variable X_i where π_i is the set of all parent nodes of X_i. If X_i is a root in the DAG which has no parent nodes, then $P(X_i \mid \pi_i)$

becomes $P(X_i)$, the prior probability of X_i. A conditional independence assumption is made for Bayesian networks:

(8) $P(X_i \mid \pi_i, S) = P(X_i \mid \pi_i)$,

where *S* is any set of variables that are not descendants of X_i. Based on this independence assumption, the joint probability distribution of *X* can be computed from local conditional probability tables (CPT) by the following chain rule: for any $X = x$,

(9) $P(x) = \Pi_{i=1}^{n} P(x_i \mid \pi_i)$.

With the joint probability distribution, BN supports, at least in theory, any probabilistic inference in the joint space. In other words, any probabilistic query concerning these variables can be computed from the joint distribution through Bayesian conditioning. Fig. 8 below gives a simple example BN, including its DAG, CPTs and the joint distribution.

Figure 8 about here

The conditional independence assumption can also be described by the notion of *d-separation* in terms of the network's topology. Fig. 9 below depicts examples of d-separation for the three types of connections in the network. In the situation of a serial connection, A and C can influence each other in either direction unless B is instantiated (A and C are said to be d-separated by B). In the diverging connection case, B and C are dependent of each other unless A is instantiated (B and C are said to be d-separated by A). In a converging connection, influence can only be transmitted between B and C if either

A or one of its descendants is instantiated, *otherwise*, B and C are said to be d-separated by A.

Figure 6 about here

If A and B are not d-separated, they are *d-connected*. In a BN, if A and B are d-separated, they are in independent of each other, and the changes in the belief of *A* have no impact on the belief of B.

From the above three cases of connections, it can be shown that the probability distribution (or belief) of a variable X_i is only influenced by its parents, its children, and its children's parents, these variables form the Markov Blanket M_i of X_i. If all variables in M_i are instantiated, then X_i is d-separated from the rest of the network, i.e., $P(X_i \mid X \setminus \{X_i\}) = P(X_i \mid M_i)$.

A typical probabilistic reasoning with BN is known as *belief update*: what would be the probability (or belief) of a variable if some other variable(s) are known to be in (or be instantiated to) certain state(s). If we denote the instantiated variables as *e* (called evidence), then what we are looking for is the posterior distribution $P(X_i \mid e)$ for any uninstantiated variable X_i. Other, more complicated probabilistic queries can also be answered. One example is the maximum a posteriori problem $\max_y P(y \mid e)$, i.e., finding the most probable instantiation of a set of variables $Y \subset X$, given *e*.

Solving these problems directly using the joint distribution $P(X)$ is practically infeasible because the size of the distribution grows exponentially with the size of the network (in

the order of $2^{|X|}$). Various efforts have been made to explore the graph structure and d-separation in developing more efficient computation. The most noted is the junction tree algorithm. This algorithm groups together those BN nodes that are tightly related into "cliques" and converts the BN into a *tree* of cliques called junction tree. CPTs are also converted into potentials for cliques. The junction tree significantly lowers the time complexity for probabilistic reasoning (from $2^{|X|}$ to $2^{|C_{\max}|}$ where $c_{\max}$, the largest clique in the junction tree, is usually a small subset of *X*).

Almost all BN packages (commercial or in public domain) implement the junction tree algorithm to support exact reasoning. However, even $2^{|C_{\max}|}$ is a huge number when the network is really large and dense, making exact solution computationally intractable. For these large networks, people turn to methods for approximate solutions. The most widely used are various stochastic simulation techniques [25]. These techniques aim to reduce the time complexity of exact solutions via a two-phase cycle: local numerical computation followed by logical sampling, which yields increasingly accurate results when the iteration continues. Different sampling methods have been investigated, including for example forward sampling, importance sampling, Gibbs sampling, etc [1].

BN is powerful as a modeling tool for domains in which the relationship among their entities and components are not certain or cannot be described logically, and it provides efficient methods for probabilistic inference. However, construction of a BN is not an easy task. For small and simple problems, it might be possible to draw the network structure (i.e., the DAG) based on domain experts' knowledge and understanding of the causal relations between the entities of interest. However, it is difficult to obtain CPTs from the experts even for small BNs because people do not think things in terms of

probability tables. Alternatively, we can construct the BN by learning both the DAG and the CPTs from the data [26].

It is easier to learn CPTs if the DAG is already known, it is much harder to learn DAG. Some methods separate these two tasks, learning DAG first and then CPTs [27]; others learn both at the same time [28]. For most of the existing BN learning methods, a training sample is required to be an full instantiation of $X = x$. Techniques have been developed to deal with missing values (some variables in some samples do not have a value) and missing variables (variables not in X, if present then a simpler probabilistic model can be built for the samples). Two criteria are followed by most learning methods. The first one is *fidelity*, the model (the learned BN) must be consistent (or with as little inconsistency as possible) to the training samples. This criterion is often judged by how close the probability distribution of the BN is to the distribution exhibited by the samples according to some distance measure (e.g., Kullback-Leibler distance or cross entropy). Since there are many BNs whose distributions are equally close to that of the samples, the quality of a learned BN is further judged by the second criterion, simplicity because a simpler BN runs faster in reasoning. The often used measure for simplicity is the maximum or average number of parents per node in the learned network. Many learning methods consider the fidelity the hard criterion and must be satisfied first, others try to strike a balance or compromise between the two [29].

Since there are too many possible BNs for a given set of variables (e.g., there are 25 different DAG of 3 variables, and the number jumps to 10^{18} with 10 variables), it is computationally intractable to guarantee finding the best BN according to any criteria. Therefore, the learning methods all follow some heuristic rules to focus the attention in

search for a good but not necessarily the best BN. Even with these heuristics, BN learning, like backpropagation learning in neural networks, usually takes a long time to complete.

IV.3 FUZZY LOGIC AND POSSIBILITY THEORY

Like probability theory, fuzzy logic is another formalism widely used to deal with uncertainty [30, 31]. However, as shall be seen shortly, the kinds of uncertainty these two formalisms attempt to model are conceptually quite different. Probability theory is based on the set theory; likewise, fuzzy set theory sets the mathematical foundation for fuzzy logic. In the ordinary set theory, a set A is associated with a Boolean membership function $f_A(.)$: for any object x, $f_A(x) = 1$ if $x \in A$, and 0 otherwise. If x is a random variable such as those representing outcomes of random experiments, then the chance that it is a member of A is the probability $P(x \in A)$. Please note that the uncertainty here is about the outcome *before* the experiment (we are not certain whether it will be head or tail before a coin is tossed). However, the outcome becomes certain after the experiment (the coin can only land on one face).

In contrast, we often face vague linguistic terms such as "tall person" and "fast car". If one tries to build a set that contains objects satisfying such a term, he will find it difficult to define a line to separate members and non-members. For example, it is easy to say a person with the height of 210 cm a member of "tall person" and of 140 cm not a member. However, it would be difficult to judge a person of 175 cm, because he is kind of tall but not really very tall. Fuzzy set theory is invented to characterize this kind of uncertainty, which is about facts (height = 175cm), not chances of things in the future. By extending the membership function of the ordinary set theory, the fuzzy membership function

becomes $F_A(x) = y$ where $0 \le y \le 1$ is the degree that *x* is thought to belong to set (or concept) *A*.

Fig. 10 below depicts three examples of fuzzy membership functions for the sets of young people, teenagers, and mid-aged people. The degree that a particular person is in such a set depends on that person's age and the set's membership function. For example, according to these functions, a thirty year old person is definitely not a teenager, and is more of a mid-aged person than a young.

Figure 10 about here

Similar to predicates in logic and prior distributions in probability theory, membership functions for sets of interest quantify one's understanding of the domain. Like other knowledge representations, these functions can be obtained from the domain experts and can also be learned from data.

Fuzzy logic treats fuzzy membership functions as (fuzzy) predicates, and defines logical operators. For example, we have

negation: $\neg F_A(x) = 1 - F_A(x)$;

conjunction: $F_A(x) \wedge F_B(x) = \min\{F_A(x), F_B(x)\}$; and

disjunction: $F_A \vee (x) F_B(x) = \max\{F_A(x), F_B(x)\}$.

Fuzzy logic is a natural choice for constructing expert systems with rules of vague terms. For example, consider the statement concerning drug formulation that disintegrants can be added to increase the drug's soluability. This piece of knowledge can be easily encoded as a fuzzy rule:

IF not soluable THEN add more disintegrant.

Note here that both “soluble” and “add more” are linguistically vague and thus can be represented as fuzzy predicates (with their particular fuzzy membership functions).

Two interesting observations can be made in comparison with the rule based systems. First, recall that it is often the case that more than one logical rule can match their conditional parts with the current working memory content. It is difficult to select one over others since logically they match the current WM equally well. However, the matches with fuzzy rules are fuzzy (a value between 0 and 1 not either 0 or 1) and often not equal, so we can rank the rules according to the numerical values of their matches and select the highest ranked one. In our drug formulation rule above, if the current formulation has very low solubility, then it matches the rule’s conditional part (“not soluble”) with a very high degree (close to 1), making it very likely to be selected and more disintegrant is added.

Second observation is also related to the numerical nature of the function values. In a rule-based system, if a rule is applied it will very unlikely to be applied again to the same data items because whatever actions this rule calls for has already been done. However, this is not the case for fuzzy rules. For example, application of the solubility rule once may only increase the solubility to a degree (say from 0.1 to 0.2), leaving the rule still applicable. What we see here is an iterative process in which the solubility of the drug increases at each iteration with more disintegrant added into the formulation. It is these features that make the fuzzy logic based expert system a popular choice for process control with wide variety of applications from home appliance control to subway locomotive auto piloting.

The relation between fuzzy logic and probability theory remains controversial. Some, including the inventor of fuzzy logic Lotfi Zadeh, consider they are two separate formalisms for different types of problems. Zadeh has created the *possibility theory* from fuzzy logic, which can be viewed as parallel to probability theory [. Many others consider fuzzy logic as a new way to express probabilities, and some went even further as claiming that everything one can do with fuzzy logic can be done by probability theory. A minority felt another way around and consider fuzzy logic is more expressive and it includes probability theory as a subtheory.

IV.3 EVOLUTIONARY COMPUTING

Evolutionary computing is a computational paradigm that seeks the globally optimal solutions for complex problems. Typically this kind of problem has many solutions, some of which are considered good or better than others according to certain criterion represented as an objective function. The goal here is to find the best from a huge solution space according to the objective function. Evolutionary computing is based on the technique called *genetic algorithm* (GA), which emulates the biological evolution process [33, 34]. As shown in Fig. 11 below, GA starts with an initial population of individuals. Each individual represents a solution, the information it carries, including a description of the solution and features/attributes that contribute to the goodness of the solution, can be viewed as a sequence of chromosomes characterizing this solution. The goodness of an individual is computed by the *fitness function*, a name borrowed from the Darwinian evolutionary principle of "survival of the fittest". From the optimization point of view, the fitness function is a realization of the problem's objective function. Two computational procedures are executed in producing the next generation of population.

The first is the one that selects parents for reproduction, this is a random process, but with higher probabilities for the individuals with better fitness function values. The selected parents are then paired and sent to the second procedure, reproduction, where cross-over generates offspring, each of which taking half of its chromosomes from each of the two parents. The hope is that by combining the features of both parents, some of the children may be better solutions than their parents. The mutation during reproduction makes random changes to some chromosomes. This is necessary because, among other things, it allows introduction of new, previously unknown chromosomes. The reproduction continues with more and more individuals produced for the new generation until the population size limit is reached. The process then repeats with the new generation of population.

Figure 11 about here

Evolutionary computing can be viewed as a stochastic search process because the randomness involved in the parent selection and mutation. It can be shown that if 1) the size of the population is allowed to be sufficiently large, 2) the process is allowed to run for a sufficiently long time, and 3) the true randomness is followed in the process, then the globally optimal solution is guaranteed to be generated.

Since usually we do not know what the *best* solution looks like or its fitness function value, there is no way one can tell the best solution is already generated and the process should terminate. Therefore we use other criteria for termination. One such criterion is if the objective (fitness) function value of the best solution in the current population falls

into the acceptable range (if such a range is provided); another is when the fitness of the best individual does not improve for a large number of generations. In either case, the global optimality is likely but not guaranteed.

V. SOME PRACTICAL APPLICATIONS IN PRODUCT AND PROCESS DEVELOPMENT

V.1 APPLICATION OF KNOWLEDGE-BASED SYSTEMS

The reader is referred to Lai's useful 1991 review [35] for an earlier discussion of the application of expert systems to pharmaceutical technology.

Immediate Release oral Solid Dosage Forms

A few examples of the application of KB systems to immediate release tablet formulation have appeared in the open literature. Podczeck [36] described a system based in part on rules constructed from laboratory experiments designed to study the relationship between independent and dependent variables. These rules were combined with others in the expert system to determine the formulation composition.

The Cadila System (Cadila Labortories, India) for tablet formulation was developed by Ramani et al. [37]. Written in Prolog, this interactive menu-driven program first requires the user to enter information on the drug properties. The system then consults its knowledge bases and selects compatible excipients with the required properties and gives their recommended proportions. A best formulation may be selected from among several feasible alternative formulations that can be generated by the expert system. The system can be queried for explanations of the decisions taken in arriving at formulations.

Rowe [38, 39] described a tablet formulation expert system that uses Logica's PFES (Product Formulation Expert System) shell. Similar to the Cadila System, the user inputs

the basic information on a new drug substance, e.g. physicochemical and mechanical properties, dose, strategy based on number of fillers. The formulation may be optimized based on the results of testing the initial formulation. The optimization is interactive with the formulator who, based on experience and expertise, can override and modify the recommendations of the expert system within a relatively broad range. The selection of ingredients and their proportions for the initial formulation are based on algorithms and production rules determined from an extensive study of previously successful formulations and certain other rules.

Related Applications

Expert systems have also been developed for certain related applications. In one of the earliest examples, Lai [40] described a prototype expert system for selecting a mixer. The system was written in TURBO Prolog which used a backward-chaining inference mechanism. Production rules were developed from a knowledge base obtained from published papers. In another example, Murray [41] described an expert system for trouble-shooting and diagnostics of a Korsch rotary tablet press. A detailed decision-tree structure was developed for each major subsystem of the tablet press, e.g. hydraulic force overload, automatic lubrication, main drive, force feeding, tablet weight verification and others. The user's answers to a series of questions enables the decision-tree structure to ascertain the symptoms or circumstances related to a specific problem and determines in what direction the diagnostic process should be approached. The system then prompts the user through a series of diagnostic or remedial measures that previously have been shown to be effective. This knowledge base is intended to be updated periodically with information derived from recent problems that have been solved and documented. A KB

system designed to diagnose and provide solution to defects in film coated tablet has also been described [42, 43].

Hard Shell Capsules

KB systems have also been developed to support formulation development for hard shell capsules. This topic is relevant to this discussion because modern capsule fillers for powder or granular formulations resemble tablet presses in that they employ both compression and ejection processes, i.e. capsule plugs are formed from the powder or granular formulation by a gentle compression or tamping process and the plugs are ejected into empty capsule shell. Moreover, the formulations for hard shell capsules typically employ the same excipients, such as fillers, lubricants, glidants and others as found in tablet formulations [44]. Bateman [45] described an expert system developed using Logica's PFES shell. This is a customized system that incorporates the practices and policies of the Sanofi Research Center. Knowledge acquired through a coordinated series of meetings with formulators was incorporated by software engineers by encoding an appropriate set of rules that reproduce the formulation experts' decision making. Another part of the system important to making formulation decisions is the excipients database. The formulation experts identified the most important properties to consider, e.g. particle size, bulk density, acid-base reactivity, amine reactivity, aqueous solubility, hygroscopicity and others. Because the information on these properties found in the literature is based on different analytical methods and therefore couldn't be correlated, it was decided to make these measurements in-house. In a preliminary validation, three chemical entities were selected to challenge the system. The formulations generated by

the KB system were judged by experienced formulators to be acceptable for manufacture and initial stability evaluation.

Unlike the Sanofi system, Capsugel's CAPEX expert system is a centralized system that incorporates worldwide industrial experience to support the formulation of powders in hard gelatin capsules [15, 46]. Development of the system was initiated at the University of London under the sponsorship of Capsugel, a division of Pfizer, Inc. The Capsugel expert system consists of three databases. One of these is 'past knowledge' which was collected from the published literature and includes information on excipients used in many marketed formulations in Europe and the US. The second database contains experiential and non-proprietary information acquired from industrial experts through classical knowledge engineering techniques. The third database consists of information generated through statistically designed laboratory studies aimed at filling knowledge gaps and providing quantitative information. These databases provided the knowledge base from which the facts and rules were derived to construct the decision trees and production rules that comprise the expert system. The system was programmed in Microsoft C and the core system was linked to a dBase driven database. The system has since been converted to a Microsoft Windows–based platform that significantly enhances its ease of use.

Under Capsugel's continuing sponsorship, the program created at the University of London was further developed and enhanced by the efforts of the University of Kyoto and the University of Maryland through additional laboratory research and a series of panel meetings in Europe, Japan, and the United States with industrial, regulatory, and academic experts.

V. 2 APPLICATION OF NEURAL NETS

Interest in the use of NNs in pharmaceutical technology and product development has been growing and has been the subject of several reviews [47 – 50]. This interest in the non-linear processing ability of NNs as a way to manage and solve pharmaceutical problems should not be surprising. The relationships that exist between formulation and process variables and desired outcomes are complex and typically non-linear. The non-linear processing ability and unique structure of NNs offer substantial promise in dealing with the problems we face in pharmaceutical product development and technology. The primary goals of applying NNs to pharmaceutical problems are optimization and prediction. The NN model that predominates in these areas is the feed forward/back propagation network, which often is simply referred to as the backpropagation network [51].

Powder Properties and Unit Operations

Several applications have been reported that deal with powder properties and certain unit operations. For example, Kachrimanis et al. [52] evaluated the effects of bulk, tapped and particle density, particle size and particle shape on the flow rate of three common excipients (Emcompress, Starch and Lactose) through circular orifices. Four sieve fractions were studied. The experimental data were modeled using a backpropagation NN. They found that the predictions of the NN were superior to those of a classic flow equation since the NN does not require a separate regression for each experiment and its predictive ability was higher. Behzadi et al. [53] reported on the validation of a modified fluid bed granulator. Sucrose was granulated under different operating conditions and their effects on the size distribution, flow rate, repose angle, and tapped and bulk volumes

of the granulation were measured. A generalized regression neural network (GRNN, a variation of radial basis function networks) was used to model the system. A good correlation was found between the predicted and experimental data.

Immediate Release Oral Solid Dosage Forms

A few reported studies employing NNs have addressed immediate release oral solid dosage forms. Using a backpropagation algorithm to build the NN, Kesevan and Peck [54] attempted to predict tablet and granulation characteristics from material and process variables. The variables considered were granulation equipment, diluent, method of addition of binder, and binder concentration. Although the prediction of granulation properties (geometric mean particle size, flowability, bulk and tapped densities) were found satisfactory, predictions of the hardness and friability of the resultant tablets were less than satisfactory. However, the NN prediction in all cases was found better or comparable to conventional regression methods. The authors suggested that the NN prediction of hardness and friability may be improved by providing more data and additional independent variables.

Bourquin et al. [55] carried out a study aimed at investigating the influence of a number of formulation and compression parameters on tablet crushing strength, percent dissolved after 15 minutes, and time to 50% dissolution. The drug substance was granulated in two different formulations. Compression parameters consisted of three levels each of matrix filling speed, precompression force, compression force and rotation speed. The dataset was mapped using three techniques: (1) a generalized feed forward NN employing a hyperbolic tangent function as an arbitrary non-linear activation function for all processing elements, (2) a hybrid network composed of a self-organizing feature map,

and (3) classic response surface methodology. NN models using an arbitrary function were found to have better fitting and generalization abilities than the response surface technique. The arbitrary hyperbolic tangent function was chosen to represent non-linearity in the data.

Ebube et al. [56] found that a NN accurately predicted in vitro dissolution based on several experimental variables, provided the NN variables were optimized and training and validation sets were appropriately selected.

Working with a high-dose plant extract, Rocksloh et al. [57] optimized the crushing strength and disintegration time of the tablets after substantial experimentation. Best results were found with a plant extract that had been granulated by roller compaction prior to tableting. In an attempt to learn more about the different effects, feedforward NNs and a partial least squares multivariate method were used to analyze the data, with the result that NNs were found more successful in characterizing the effects that affect crushing strength and disintegration time. Shao et al. [58] found both NNs and neurofuzzy logic to successfully develop predictive models for the crushing strength and dissolution of an immediate release formulation, but the latter logic had the additional advantage of generating rule sets for the cause-effect relationships in the experimental dataset.

Peng et al. [59] used trained NN models to predict the dissolution profiles of immediate release beads loaded with 40% acetaminophen. The beads were prepared by extrusion and spheronization. The training set consisted of 18 batches that were prepared based on a full-factorial design. The variables were extruder type, screw speed, spheronization

speed and spheronization time. The NN model trained with a genetic algorithm exhibited better predictability than that trained with a neural algorithm.

Kuppuswamy et al. [60] used a back propagation network to model the relationship between the hardness and friability of direct compression tablets produced from nine mixtures of varying compactibility and tableting indices (Hiestand). The goal was to predict the hardness and friability of tablets from the index values. It was concluded that tableting indices did not have a general ability to predict compactibility since quantitative prediction was only possible when the model was trained with similar materials. Different materials having closely similar indices could have widely differing compactibility.

Modified Release Oral Solid Dosage Forms

A review of the literature suggests a strong interest on the part of researchers in applying NNs to the development of modified release oral solid dosage forms. One of the earliest reports in this application area is that of Hussain et al. [61] who used a backpropagation network to discern the complex relationship between certain formulation variables and the in vitro release of chlorpheniramine maleate from a hydrophilic matrix capsule system. They found that NN analysis could predict the response values for a series of validation experiments more precisely than response surface methodology. In a later study, Hussain et al. [62] describe the use of a non-linear feed forward network to recognize the relationships between the drug, formulation properties and the in vitro release of the drug from hydrophilic matrix tablets. Eleven drugs were studied in three different ratios with hydroxypropyl cellulose. The drugs were characterized by their intrinsic dissolution rate, salt type, pKa, and molecular weight. Three polymer molecular

weight grades were characterized by their hydration times. The NN developed from this dataset was used to predict the in vitro release profile of the drugs, and the prediction error (RMS) was found acceptable for most, but not all, of the drugs and polymer ratios. The authors concluded that even though the formulation examples and test conditions are simplistic, the results of the study are useful in that they demonstrate the potential advantages and limitations of this approach.

Takahara et al. [63] reported the use of a multi-objective optimization technique based on a NN for a sustained release tablet. The quantities of microcrystalline cellulose, hydroxypropyl methylcellulose and the tablet compression pressure were considered the causal factors. The drug release order and release rate were the responses. The response surface of a NN was used to recognize the non-linear relationship between the causal factors and the responses. Simultaneous optimization was carried out by minimizing the generalized distance between the predicted values of each response and the optimized one.

Similarly, Takayama et al. [64] described the application of simultaneous optimization incorporating a NN to theophylline hydrophilic matrix controlled release tablets. The levels of a commercial 80:20 hydroxypropyl methylcellulose: lactose mixture and cornstarch, and the compression pressure were the causal factors. The release profiles were represented by the sums of the fast and slow release fractions. Release parameters were the initial weight, the rate constant in the fast release fraction and that in the slow release fraction. A desired set of release parameters were obtained based on human pharmacokinetic data. NN response surfaces were used to recognize the non-linear relationships between the causal factors and the responses. Simultaneous optimization was performed using a generalized distance function method which minimizes the

distance between the predicted values of each response and the desirable one that was optimized individually. Fairly good agreement between the observed and predicted release parameters was found. The use of the generalized distance function combined with a generalized regression neural network (GRNN) to optimize aspirin extended release tablets has been reported [65]. The tablets were formulated using Eudragit L 100 as the matrix substance.

Chen et al. [66] combined a NN with pharmacokinetic simulations to design a controlled release tablet formulation for a model sympathomimetic drug. Ten independent variables for 22 tablet formulations provided the model input. In vitro cumulative percent of drug released at ten different sampling times was the output. The NN was developed and trained using CAD/Chem software, and the trained model was used to predict the best compositions based on two desired in vivo release profiles and two desired in vitro dissolution profiles. Three of four predicted formulations exhibited very good agreement between the NN-predicted and the observed in vitro dissolution profiles based on similarity metrics (f1, f2). Chen et al. [67] later used the above data as the basis to compare four commercially available NN software packages (NeuralShell2, BrainMaker, CAD/Chem, NeuralWorks) for their ability to predict in vitro drug release. The percent dissolved at ten different sampling times was the output. The slopes of predicted vs. observed percentage of drug dissolved ranged from 0.95 to 1.01 (r^2 = 0.95-0.99) for the four optimized models. The authors concluded that all four programs gave reasonably good predictions from this dataset, but one (NeuroShell2) was preferred based on similarity metrics, exhibiting lower f1 and higher f2 values compared to the others.

NNs also can be used to rank which of various formulation and process variables are most critical in influencing responses. For example, Leane et al. [68] described the successful use of Input Feature Selection (IFS) to identify the most important factors affecting in vitro dissolution from enteric coated minitablets. Using Trajan software, IFS was implemented in two ways: stepwise algorithms that progressively add or remove variables and a genetic algorithm. NNs were then trained using the backpropagation algorithm to determine whether or not the IFS had correctly identified any unimportant inputs.

In other applications to modified release tablets, NNs have been applied to the optimization of osmotic pump tablets [69] and to model bimodal delivery [70]. In the latter, the precision of the predictive ability of different training algorithms was compared.

Experience with a Hybrid "Expert Network" System

Under the sponsorship of Capsugel, a feasibility study was carried out at the University of Maryland to link an expert system for capsule formulation support with a neural net [7, 8]. The goal was to create an intelligent system that can generate capsule formulations that would meet specific drug dissolution criteria for BCS Class II drugs, i.e. drugs that would be expected to exhibit dissolution rate-limited absorption. Piroxicam was selected as a model Class II drug with which to demonstrate feasibility. A modified expert system patterned after the Capsugel system was created for this project. The new system provided an opportunity to build certain additional features into the decision process and to use a more effective and more flexible programming language package. Unlike the original Capsugel system written in C, this expert system was constructed as a rule-based system and encoded in Prolog. This structure provides certain advantages. In Prolog,

knowledge is separated from the inference engine. Thus, the designer need only provide knowledge base, since the inference mechanism is provided by the language package. Another advantage is that the rules are local and relatively independent of the inference engine. This feature makes maintenance and updating of the knowledge base easy. A Prolog rule-based system is also more suited to managing complex formulation problems than a decision tree because it can represent more-complicated decision logic and more abstract situations.

Figure 12 about here

As depicted in Fig. 12, the expert system is linked to a neural network to form a hybrid system. The expert system is the "decision module" that generates a proposed formula based on data and requirements input by the user; the NN, trained by backpropagation algorithm, serves as the 'prediction module' that predicts the dissolution performance of the proposed formulation. The "control module", driven by the difference that might exist between the desired dissolution rate and the predicted dissolution rate of the proposed formulation controls the optimization process. The control module inputs the formulation from the decision module to the prediction module to compute the predicted dissolution rate and asks for the user's acceptance of the currently recommended formulation based on that predicted dissolution rate. If the user accepts the formulation, the control module will terminate the formulation process. If not acceptable, the control module will present a set of choices of parameter adjustments (e.g. excipients levels) to the user for improving the dissolution rate. This prototype was found to have good

predictability for the model compound, piroxicam. Later, a more generalized version of this system which included parameters to address wettability and the intrinsic dissolution characteristics of the drugs was found to show good predictability for several BCS II drugs representing a broad range in solubilities [71]. The approach demonstrated here for capsule formulations should be readily adaptable to tablets.

VI. THE FUTURE

Product development is a complex, multi-factorial problem requiring specialized knowledge and often years of experience. The need to speed up the development process and modernize manufacture and control will drive academic and industry researchers to develop a more fundamental understanding of product and process that will enable the identification and measurement of critical formulation and process attributes that relate to product quality and to model the relationships between product quality attributes and measurements of critical material and process attributes. The contributions of knowledge-based systems and other artificial intelligence techniques can make to decision making, product and process optimization and identifying critical variables, and codifying and preserving knowledge have already been demonstrated through numerous examples. But their full potential in pharmaceutical technology has not been realized. That will require more a fundamental understanding of our systems and a stronger commitment to build collaborative relationships with artificial intelligence and information technology specialists who can help us exploit artificial intelligence and translate the problems and goals of pharmaceutical technology into practical solutions. Most AI methods with substantial applications in drug formulation (e.g., rule-based systems, decision trees, BP neural networks, genetic algorithms) were “old” techniques

developed in 1970s and 80s. Since then quite a number of techniques for intelligent systems have reached a level of maturity for real world applications. It would be interesting to see how these "new" techniques can be applied to help solving difficult problems in drug formulation. For example, support vector machine (SVM) has been reported to outperform several other machine learning techniques, including BP networks, RBF networks, and decision trees, in both learning time and the validation accuracy in data analysis for drug discovering [72]. Similar performance would be expected when SVB is applied to tableting and other formulation optimization tasks.

Similarly, the technique of Bayesian networks, especially BN learning, is also very promising in drug formulation. BN can be seen to be advantageous over neural networks in several aspects. First, BN models the interdependencies between variables, not just black box associating input patterns to their desired outputs as does BP networks, so every part of the BN can be evaluated and validated by human experts. Secondly, NN, when used as a predictor, generates the most likely/plausible output pattern for a given input pattern. In contrast, BN provides posterior distribution of output variables for the given input, as such not only one can find the most probable output pattern, but also knows its likelihood, and the likelihoods of other good patterns that ranked lower than the best one. Thirdly, the Bayesian analysis can be done using any combination of variables as the conditionals. This kind of flexibility goes far beyond what can be supported by any NN models, making BN a powerful modeling tool for what-if analysis.

Another new technique of interest is *Semantic Web* (SW) [73]. Unlike most AI techniques reviewed in this chapter, SW is not a technique for data analysis or knowledge representation; rather it is a technique that helps a better sharing of data and knowledge.

Pages in the current World Wide Web are intended for human consumption. Their contents are not understood by computer programs. To make web pages understandable by programs, the semantic web extends the current web by providing additional markups to articulate the semantic or meaning of the web contents. The semantic markups are according to shared ontologies written in a standard web ontology definition language (OWL) based on a variation of first order logic known as the description logic. Semantic web thus can be viewed as a web of data that is similar to a globally accessible database. How to build a shared ontology for drug formulation (as part of a much larger ontology for pharmaceuticals) and how to utilize the huge amount of data and knowledge that become available for machine processing is a research direction of great potential.

REFERENCES

1. Russell SJ, Norvig P. Artificial Intelligence: A Modern Approach. 2nd ed. Upper Saddle River: Prentice Hall, 2003.
2. Turban E. Expert Systems and Applied Artificial Intelligence. New York: Macmillan Publishing Co., 1992: 665-696.
3. Rowe RC, Roberts RJ. Intelligent Software for Product Formulation. Series in Pharmaceutical Sciences, New York: Taylor & Francis, 1998.
4. Mehrotra K, Mohan CK, Ranka S. Elements of Artificial Neural Networks. Boston, MA: MIT Press, 1997
5. Caudill M. Expert networks. In: Eberhart RC, Dobbins RW, eds. Neural Network PC Tools. San Diego, CA: Academic Press, 1990: 189-214.
6. Shortliffe EH. Computer-Based Medical Consultations: MYCIN. New York: Elesevier/North-Holland, 1976.
7. Guo M, Kalra G, Wilson W, Peng Y, Augsburger LL. A prototype intelligent hybrid system for hard gelatin capsule formulation development. Pharm Tech. 2002, 26 (9), 44-60.
8. Kalra G, Peng Y, Guo, M, Augsburger, LL. A hybrid intelligent system for formulation of BCS Class II drugs in hard gelatin capsules. Proceedings, International Conference on Neural Information Processing, Singapore, November 2002.
9. Quinlan JR. C4.5: Programs for Machine Learning. San Mateo, CA: Morgan Kaufmann, 1993.
10. Wygant RM. CLIPS – a powerful development and delivery expert system tool. Computers and Industrial Engineering, 1989, 17, 546-549.

11. Friedman-Hill E. Jess in Action: Java Rule-Based Systems. Greenwich CT: Manning Publications Co., 2003.
12. Sagonas K, Swift T, Warren DS. XSB as an Efficient Deductive Database Engine. Proceedings of ACM Conference on Management of Data (SIGMOD), 1994.
13. Skingle, B. An introduction to the PFES Project. In: Proceedings of the 10th International Workshop on Expert Systems and Their applications, 1990, 907-922.
14. Bentley P. Production Formulation Expert Systems (PFES). Rowe RC, Roberts RJ, ed. Intelligent Software for Product Formulation. Series in Pharmaceutical Sciences, New York: Taylor & Francis, 1998, 27 – 30.
15. Lai S, Podczeck F, Newton, JM, Daumesnil R. An expert system to aid the development of capsule formulations. Pharm Tech Eur. 1996, 8 (10), 60–68.
16. McCulloch WS, Pitts W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 1943, 5, 115-133.
17. Hebb DO. The Organization of Behavior. New York: Wiley, 1949.
18. Werbos PJ. The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting. New York: Wiley, 1994.
19. Fahlman SE. Faster-learning variations on back-propagation: An empirical study. In: Proceedings of the Connectionist Models Summer School, Los Altos CA: Morgan-Kaufmann, 1988.
20. Poggio T, Girosi F. Networks for approximation and learning. In: Proc. IEEE 1990, 78(9), 1484-1487.
21. Kohonen T. Self-Organizing Maps. Berlin: Springer, 1995.

22. Vapnik V. Estimation of Dependences Based on Empirical Data [in Russian]. Nauka, Moscow, 1979. (English translation: New York: Springer Verlag, 1982).
23. Burges CJC. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 1998, 2, 121 – 167.
24. Pearl J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Mateo, CA: Morgan Kauffman Publishers, 1988.
25. Pearl J. Evidential reasoning using stochastic simulation of causal models. Artificial Intelligence, 1987, 32, 245-257.
26. Heckerman D. A tutorial on learning with Bayesian networks. In: Jordan MI, ed. Learning in Graphic Models. Dordreckt, Netherlands: Kluwer, 1998, 301-354.
27. Cooper G, Herskovits E. A Bayesian method for the induction of probabilistic networks from data. Machine Learning, 1992, 9, 309-347.
28. Peng Y, Zhou Z. A Neural network learning method for belief networks", International Journal of Intelligent Systems, 1996, 11, 893-916.
29. Lam W, Bacchus F. Learning Bayesian belief networks: An approach based on the MDL principle. Computational Intelligence, 1994, 10, 269-293.
30. Zadeh LA. Fuzzy sets. Information and Control, 1965, 8, 338-353.
31. Zimmermann HJ. Fuzzy Set Theory. Dordreckt, Netherlands: Kluwer, 2001.
32. Zadeh LA. Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, 1978, 1, 3-28.
33. Holland JH. Adaption in Natural and Artificial Systems. Reading, MA: Addison-Wesley, 1975.

34. Fogel DB. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. Piscataway, NJ: IEEE Press, 2000.

35. Lai, FKY. Expert Systems as applied to pharmaceutical technology. In: Swarbrick J, Boylan J, eds. Encyclopedia of Pharmaceutical Technology. New York: Marcel Dekker, 1991, 361-378.

36. Podczeck F. Knowledge-based system for the development of tablets. Proceedings of the 11th Pharmaceutical technology Conference, 1992, 1, 240-264.

37. Ramani KV. An expert system for drug preformulation in a pharmaceutical company. Interfaces, 1992, 22, 101-108.

38. Rowe RC. Expert systems in solid dosage development. Pharm Ind, 1993, 55, 1040-1045.

39. Rowe RC. An expert system for the formulation of pharmaceutical tablets. DTI Manuf. Intel Newsltr, 1993, 14, 13-115.

40. Lai FKY. A prototype expert system for selecting pharmaceutical powder mixers. Pharm Tech, 1988, 12(8), 22-31.

41. Murray FJ. The application of expert systems to pharmaceutical processing equipment. Pharm Tech, 1989, 13(3), 100-110.

42. Rowe RC, Upjohn NG. An expert system for identifying and solving defects on film-coated tablets. Manuf. Intel Newsltr, 1992, 12, 12-13.

43. Rowe RC, Upjohn NG. An expert system for the identification and solution of film coating defects. Pharm Tech Int, 1993, 5(3), 34-38.

44. Heda PK, Miller FX, Augsburger LL. Capsule filling machine simulation I. Low force compression physics relevant to plug formation. Pharm Devel Tech. 1999, 4(2), 209-219.

45. Bateman SD. The development and validation of a capsule knowledge-based system. Pharm Tech. 1996, 20 (3), 174-184.

46. Expert System for Formulation Support. Brochure, Capsugel Library, Capsugel, Inc., Greenwood, SC, 1996.

47. Takayama K, Fugikawa M, Nagai T. Artificial neural network as a novel method to optimize pharmaceutical formulations. Pharm Res. 1999, 16, 1-6.

48. Takayama K, Fugikawa M, Obata F, et al. Neural network based optimization of drug formulations. Adv Drug Del Rev. 2003, 55, 1217-1231.

49. Ichikawa H. Hierarchy neural networks applied to pharmaceutical problems. Adv Drug Del Rev. 2003, 55, 1119-1147.

50. Sun Y, Peng Y, Chen Y, et al. Application of artificial neural networks in the design of controlled release drug delivery systems. Adv Drug Del Rev. 2003, 55, 1201-1215.

51. Erb RJ. Introduction to backpropagation neural network computation. Pharm Res. 1993, 10, 165-170.

52. Kachrimanis K, Karamyan, Malamataris S. Artificial neural networks (ANNs) and modeling powder flow. Int J Pharm. 2003, 250 (1), 13-23.

53. Behzadi SS, Klocker J, Hürdin, H, et al. Validation of fluid bed granulation utilizing artificial neural network. Int J Pharm. 2005, 291 (1-2), 139-148.

54. Kesevan JG, Peck GE. Pharmaceutical granulation and tablet formulation using neural networks. Pharm Dev tech. 1996, 1(4), 391-404.

55. Bourquin J, Schmidli H, van Hoogevest P, et al. Application of artificial neural networks (ANN) in the development of solid dosage forms. Pharm Dev Tech. 1997, 2(2), 111-121.

56. Ebube NK, McCall T, Chen Y, et al. Relating formulation variables to in vitro dissolution using an artificial neural network. Pharm Dev Tech. 1997, 2, 225-232.

57. Rocksloh K, Rapp F-R, Abed SA, et al. Optimizing of crushing strength and disintegration time of a high-dose plant extract tablet by neural networks. Drug Dev Ind Pharm. 2004, 25, 1015-1025.

58. Shao Q, Rowe RC, York P. Comparison of neurofuzzy logic and neural networks in modeling experimental data of an immediate release tablet formulation. Eur J Pharm Sci. 2006, 28, 394-404.

59. Peng Y, Geraldrajan M, Chen Q, et al. Prediction of dissolution profiles of acetaminophen beads using artificial neural networks. Pharm Dev Tech. 2006, 11, 337-349.

60. Kuppuswamy R, Anderson SR, Hoag SW, et al. Practical limitations of tableting indices. Pharm Dev Tech. 2001, 6(4), 505-520.

61. Hussain AS, Yu X, Johnson RD. Application of neural computing in pharmaceutical product development. Pharm Res. 1991, 8, 1248-1252.

62. Hussain AS, Sivanand P, Johnson RD. Application of neural computing inpharmaceutical product development. Drug Dev Ind. Pharm. 1994, 20, 1739-1752.

63. Takahara J, Takayama K, Nagai T. Multi-objective optimization technique based on artificial neural network in sustained release formulations. J Control Rel. 1997, 49, 11-20.

64. Takayama K, Morva A, Fujikawa M, et al. Formula optimization of theophylline controlled-release tablet based on artificial neural networks. J Control Rel. 2000, 68, 175-186.

65. Ibrić S, Jovanović M, Djurić Z, Parojčiet al. Artificial neural networks in the modeling and optimization of aspirin extended release tablets with Eudragit L100 as matrix substance. AAPS PharmScitech. 2003, 4 (1) Article 9 http://www.pharmscitech.org

66. Chen Y, McCall TW, Baichwal AR, et al. The application of an artificial neural network and pharmacokinetic simulations in the design of controlled-release dosage forms. J Control Rel. 1999, 59, 33-41.

67. Chen Y, Jiao, T, McCall TW, et al. Comparison of four artificial neural network software programs used to predict the in vitro dissolution of controlled-release tablets. Pharm Dev Tech. 2002, 7(3), 373-379.

68. Leane MM, Cumming I, Corrigan OI. The use of artificial neural networks for the selection of most appropriate formulation and processing variables in order to predict the in vitro dissolution of sustained release minitabs. AAPS PharmSciTech. 2003, 4(2) Article 26 http://www.pharmscitech.org

69. Wu T, Pan W, Chen J, et al. Formulation optimization technique based on artificial neural network in salbutamol sulfate osmotic pump tablets. Drug Dev Ind Pharm. 2004, 26, 211-215.

70. Ghaffari A, Addollahi H, Khoshayand MR, et al. Performance comparison of neural network training algorithms in modeling of bimodal drug delivery. Int J Pharm. 2006, 327(1-2), 126-138.

71. Wilson W, Peng Y, Augsburger LL. Generalization of a prototype intelligent hybrid system for hard gelatin capsule formulation development. AAPS PharmSciTech. 2005, 6(3), E449-E457. http://www.aapspharmscitech.org/view.asp?art=pt060356

72. Burbidge R, Trotter M, Buxton B, et al. Drug design by machine learning: Support vector machines for pharmaceutical data analysis. Computers in Chemistry, 2002: 26(1), 5-14.

73. Berners-Lee T, Hendler J, Lassila O. The semantic web. Scientific American, May 2001.

LIST OF FIGURES