Active training of backpropagation neural networks using the learning by experimentation methodology

Fu-Ren Lin
Department of Information Management, National Sun Yat-sen University, Kaohsiung, Taiwan 804, Republic of China

Michael J. Shaw
Department of Business Administration, University of Illinois at Urbana-Champaign, Champaign, IL 61820, USA

This paper proposes the Learning by Experimentation Methodology (LEM) to facilitate the active training of neural networks. In an active learning paradigm, a learning mechanism can actively interact with its environment to acquire new knowledge and revise itself. The learning by experimentation is an active learning strategy. Experiments are conducted to form hypotheses, and the evaluation of those hypotheses feeds back to the learning mechanism to revise knowledge. We use a backpropagation neural network as the learning mechanism. We also adopt a weight space analysis method and a heuristic to select salient attributes to perform new experiments in order to revise the network. Finally, we illustrate performance by solving the sonar signal classification problem.

Keywords: neural networks, Learning by Experimentation Methodology, active learning

1. Introduction

Learning is an interaction process between the learner and the environment. The learner receives information from the environment, and processes the information to form knowledge. The learner uses this learned knowledge to help in making decisions and in solving problems. The performance of problem solving will feed back to the learner. This process occurs iteratively and continues the “learning”. The environment serves as a stimulus to the learner, and the level and quality of information affect the strategy of learning. Generally, there are three main learning paradigms: supervised, semi-supervised, and unsupervised learning. The goal of the learning is to generate the regularities from the training set of input patterns. Among these three learning paradigms, although the level and quality of information from the environment vary, they share with one common learning strategy that learning comes passively from the environment.

Different to passive learning strategy, the *Learning by Experimentation Methodology* (LEM) is an *active* learning strategy. In an active learning paradigm, a learner should be able to interact actively with its environment to acquire new knowledge by generating hypotheses based on its own knowledge and testing them on its environment. The results of the hypothesis testing contribute heuristics for the learner to revise its knowledge. Learning by experimentation consists of two main tasks: *hypothesis formation* and *hypothesis revision*. To formulate hypotheses, the LEM begins with defining the problem domain, building the model, selecting the experimental goals and experimental design, and then performing experiments. The results of the experiments, which are distinct instances, can be generalized through inductive learning to form the hypotheses. At this point, the facts from experiments are not stored as individual data items. For experimenters, the abstracted hypotheses shed light on the interpretation of the problem, and provide insight for decision making. For hypotheses to be a theory, it must be examined for validity. The hypotheses can either be tested on existing facts or used to predict future events and unseen phenomena. The results, either confirming or denying the hypotheses, are the source of the hypothesis revision. Hypothesis revision through the LEM is through the observation of performance of existing hypotheses, and then the detection of the defective hypotheses to revise the hypotheses. The heuristics of experimental design, generated from the observation of the performance of existing hypotheses, will guide the following experiments. Such hypothesis revision cycles will continue until learners or decision makers accept them as the theory for solving problems of interest. We will enumerate the detailed stages of the LEM in section 2.

Learning by experimentation methodology is proposed and applied in various areas, such as AM [12], BACON [11], LEX [14], IDS [15], KEKADA [10], and PDS [16, 19]. These studies use the symbolic learning approach, but work on using neural networks to accomplish the LEM is still rare [22]. However, we found some advantages in using neural networks for learning by experimentation, motivating our pursuit of this research. First, learning by experimentation is an incremental learning process, and learning based on neural networks is essentially an incremental learning. Therefore, using neural networks is a straightforward option for inductive learning mechanisms. Second, learning in neural networks is more robust in dealing with noisy data than some symbolic learning algorithms. Hence, the concepts learned using neural networks are more accurate. Third, because the learned concepts are embedded in the connection of neural networks, once the concepts are learned it is very efficient to generate the output given the input variables. Therefore, this efficiency makes neural networks suitable for the real-time dynamic control systems. Finally, the property of the incremental learning of neural networks is very suitable for knowledge revision in the LEM.

This research uses the advantages of the characteristics of neural networks in the inductive learning mechanism and applies the learning by experimentation methodology to perform active learning. The incremental learning, shorter response time, and noise tolerance are very suitable for hypothesis formation and hypothesis revision. Therefore, the linkage of neural networks with the LEM is one of the main contributions. The other contributions of this work include methods for the salient attribute detection and the informative example selection for the active learning.

2. Learning by Experimentation Methodology

The Learning by Experimentation Methodology (LEM) is an active learning paradigm in which the learner should actively select examples from the environment as the input patterns for induction. It is also a supervised learning approach in which the experimentation acts as an oracle to generate the correct output patterns. Therefore, in the LEM, the labeled training examples are provided through the interaction between the learner and the experimentation. The learner decides values for independent variables, and the experimentation produces values for dependent variables corresponding to the independent variables. An inductive learning mechanism is used to form hypotheses from the training examples and to revise hypotheses according to the results of problem solving.

The framework of the LEM consists of two main tasks: *hypothesis formation* and *hypothesis revision*. First, during the hypothesis formation, a learner should define the problem domain, determine the variables, and build the model. Second, a learner can decide the experimentation goal, experimental design, and then perform experiments. Finally, from the experiments' results, which are distinct instances, an inductive learning mechanism can be applied to form hypotheses. The generated hypotheses have to be tested before they are viewed as theories. These can either be tested on seen events or used to predict future events. The results of testing, either support or deny hypotheses, feed back for the hypothesis revision. During the hypothesis revision, first, a learner should detect the defective hypotheses, locate the important variables, and then concentrate on the informative values. Second, experiments are performed by inputting these informative values as the independent variables, and training examples are collected for the further learning. Finally, the hypotheses are refined and then tested. These stages continue iteratively until the learner is satisfied with the learned hypotheses.

Figure 1 shows the spiral model of the life cycle for learning by experimentation, which demonstrates LEM's *dynamic*, *continuous*, and *self-adaptable* properties. The curve extends outward while the quality of hypotheses (theory) improved by theory revision. The curve stops growing when the performance of the learned knowledge exceeds the threshold of acceptance. It is dynamic because the monitoring of the system performance will trigger the theory revision process when the prediction accuracy of theory is below the revision threshold. It is continuous because the outside circle is not a direct leap from the inside core, but a continuous movement consisting of the iteration of hypothesis formation and revision stages. It is self-adaptable because the learning agent itself controls the learning process, including how to investigate

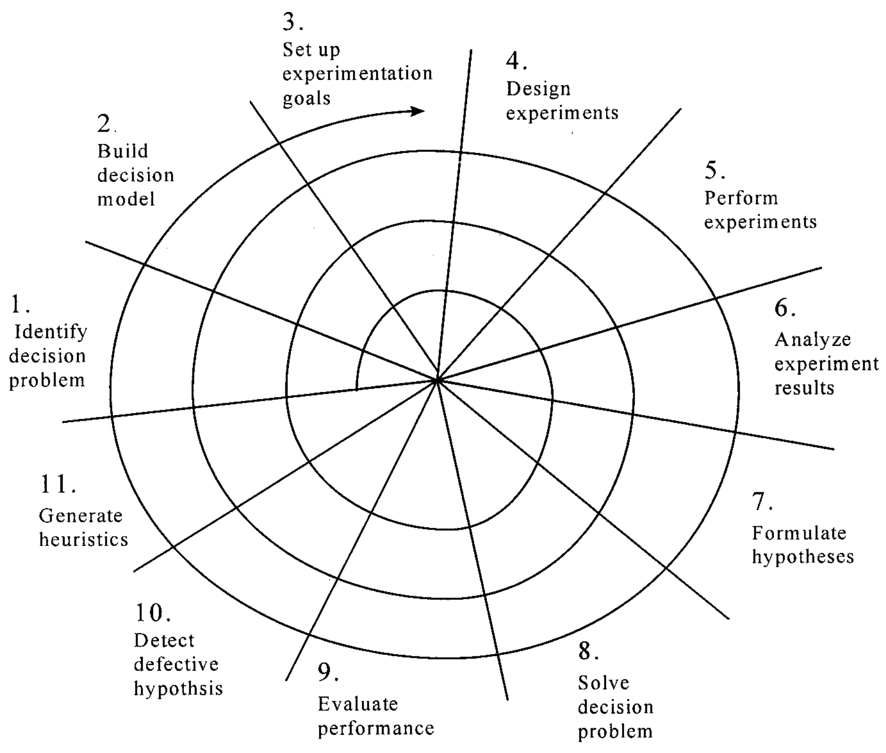
3. Set up experimentation goals
4. Design experiments
2. Build decision model
5. Perform experiments
1. Identify decision problem
6. Analyze experiment results
11. Generate heuristics
7. Formulate hypotheses
10. Detect defective hypothsis
9. Evaluate performance
8. Solve decision problem

Figure 1. The spiral model of the LEM.

the outside world using its own experimental design, obtain various quality of theory, and generate heuristics for refining the existing theory by monitoring the results of problem solving. The iteration of hypotheses formation and revision in the LEM can be further developed into four iterative phases: *experimentation*, *hypotheses formation*, *problem solving* and *performance evaluation*, and *hypotheses revision*, as shown in figure 2 and discussed in the following subsections. Table 1 summarizes the input, output and task for each phase.

2.1. Experimentation

Experimentation plays a fundamental role in scientific discovery. Scientists use experiments to investigate phenomena, gather data, and verify theories. In an experiment, there are *factors* which are observable and controllable and *responses* which are effects caused by the manipulation of factors. Therefore, instead of passively receiving instances from the outside world, experimentation serves as an active information collector. The goal of experimentation is to investigate the relationship between actions and responses of the system being examined in various system settings. Experimental design completely specifies the experimental test runs. There are

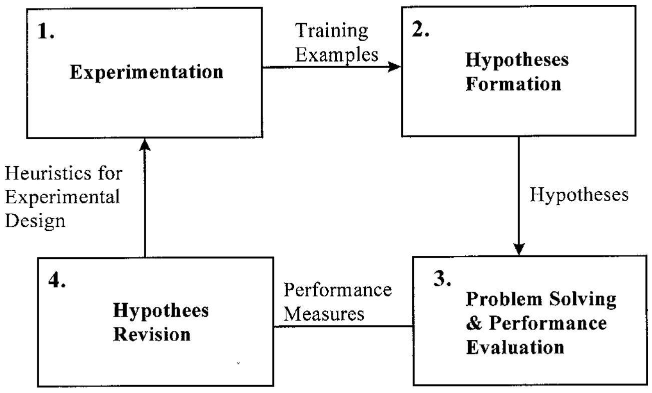

Figure 2. The iterative phases of the LEM.

Table 1

Phases of the learning by experimentation.

Phases	Inputs	Outputs	Tasks
Experimentation	Experiment factors	Experiment responses	Manipulating different factors to obtain various responses, and recording them
Hypothesis formation	Training examples	Hypotheses for solving domain problem	Generating hypotheses by generalizing the training examples
Problem solving and performance evaluation	Hypotheses for solving domain problem	Performance measures	Evaluating performance by applying the hypotheses
Hypothesis revision	Performance measures	Heuristics of experimental design	Examining the hypothesis space

several approaches to decide how to combine factors as the inputs of the experiment. The n^k factorial design, where n is the number of values for each factor and k is the number of factors, is the simplest one and is used for a small number of factors or values, but requires more runs. The n^{k-p} fractional factorial design reducing the number of runs $1/n^p$, where p is the number of factors which are confounded with other factors, is a more efficient experimental design method. *Blocking design* groups the experimental units or test runs into blocks so that those within a block are more homogeneous than those in different blocks. We can randomly assign all factor combinations of interest in the experiment to each block. For factors which have hierarchical relations, the *nested experimental design* can be used to determine the hierarchy of the factors, select the levels for each factor within the levels of the preceding factor, and randomize the run order or the assignment of factor-level combinations to experimental units. In the methodology of learning by experimentation, the experimentation is a mechanism that generates the training examples for hypothesis formation. The experimental design also takes into account the heuristics generated from the hypothesis revision.

Simulation modeling uses computer technology to imitate the operations of real-world facilities or processes, such as the manufacturing processes. It is typically used to simulate the systems that are too complex to evaluate analytically. Therefore, instead of performing experimentation on real-world systems, simulation models can be used to host the experimentation. Before administering the experimentation on simulation models, we must validate the agreement between the real-world system and the simulated models, and also verify that the computer programs perform the same operations as the simulated systems. After these validations and verifications, the experimental design mentioned above can be applied to the simulation models.

Experimentation is an active approach for decision makers or problem solvers to explore the environment by changing some situations in the given model and then observing the results of those changes. These changes can be systematic such as the experimental design mentioned above, or dynamically controlled by some heuristic criteria such as the *interestingness* used in AM system [12]. In the LEM, the experimentation works as an example generator. It is designed to demonstrate the performance of problem solving and then the situations, actions and results are stored as training examples for inductive learning. The design of experimentation is guided both by the systematic factor combination and heuristics.

2.2. Hypothesis formation

Hypothesis formation, in essence, is a process of inductive learning, especially *learning from examples*. Inductive learning considers the transformation from the instance space to the hypothesis space. Instance space consists of the labeled examples, $\langle x_1, x_2, x_3, \ldots, y \rangle$, where x_i is the value of the ith variable and y is the classification. Hypothesis space can be represented by various forms. For example, we can utilize a decision tree learning method and obtain decision trees, or use a neural network method and maintain neural networks. The tasks of inductive learning are to generalize the relationship among examples and represent them in a concise and consistent form (depending on the knowledge representation) to predict the unseen events. The performance of learning from examples is heavily influenced by the training examples. Therefore, the distribution of training examples, the noise, and the bias of sampling data are major considerations in inductive learning.

Since learning by experimentation is an active learning process by which a learning agent actively selects training examples for its own goal, the experimentation and hypothesis formation work is a supplier–consumer relationship. The examples generated from experimentation are the source for the learning from examples. On the other hand, the hypotheses thus obtained will be used to solve problems. This chain effect shows that the performance of the learning process is decided by the quality of the training examples, and the success of problem solving is decided by the quality of the learned hypotheses.

2.3. Problem solving and performance evaluation

The learned hypotheses can be tested by generated testing examples or can be used to solve problems in order to evaluate effectiveness and efficiency. The complexity of the problems and the comprehensiveness of the applied theory may influence the performance of problem solving. There are two kinds of testing approaches: *resubstitution estimate* and *holdout estimate* [2, pp. 353–359]. The resubstitution estimate uses the training examples to test the learned hypotheses, while the holdout estimate separates examples into training and testing sets, two mutual exclusive sets. The holdout estimate can extend to a more complicated but more comprehensive testing method, the *rotation estimate*. The rotation estimate, similar to cross-validation as used in statistics, divides the examples into n subsets, and each time takes one subsets as the testing set and the rest of the $n - 1$ subsets as the training sets. The subsets are interchanged until each has been used as the testing set.

Due to the incompleteness or incorrectness of the existing hypotheses, the knowledge base may require revision. The results of problem solving or testing are the source of feedback to the whole system of learning by experimentation. This feedback provides the information for the theory revision mechanism to refine the existing theory in the knowledge base.

The main barrier to the acceptance of neural network application has been the lack of explanation facilities. That is also the most difficult problem for us in using neural networks as the learning mechanism in the learning by experimentation methodology because we have to analyze the hypothesis space to generate heuristics for the hypothesis revision. Several researches have made contributions exploring this topic. Hwang et al. [8] propose network inversion on feedforward multi-layer perceptron. The inversion of a network will generate the input vector that can produce a desired output vector. The network inversion will help determine the decision hypersurface and allocate the codebook vector. Eberhart [3] uses the genetic algorithm as a means to achieve the neural network inversion. Yoon et al. [21] analyze the weight space and determine the relative effect of attributes toward the output classification. In this research, we follow a strategy similar to that of Yoon et al. to analyze the weight space in the feedforward neural network, and then detect the salient attributes which play an important role in the explanation of trained networks. We will describe this in subsection 4.1.

2.4. Hypothesis revision

The LEM hypothesis revision lies in the ability to actively select training examples. An active learning system, besides the concept formation agent, includes the example selector, performance evaluation, and the revision processor. By random selection, the example selector generates a random example by choosing at random one of the allowable values for each attribute presented in the domain theory. For a numeric attribute, we can first randomly select one of the subintervals, and then a value from this subinterval is assigned to that attribute. The near misses approach lies in the observation that negative examples which are close to being positive examples are useful in concept learning [20]. The example selector generates a near miss example by first randomly generating a positive example, and then one of the attributes is randomly chosen and its value is changed so that the example becomes a negative example. By selecting examples with the knowledge of the emerging theory, the example selector can present those examples that are currently misclassified by the emerging theory to the learning system. Rendell and Cho [17] propose a boundary concentration strategy which focuses the example selection on both the *near miss* and *near hit* examples. Because the clarification of the boundary of target concept brings valuable information to concept formation, near misses (i.e., the examples which are just outside the boundary of target theory) and near hits (i.e., the examples which are inside the boundary of target theory) provide the highest utility toward this purpose.

Eberhart [3] selects a hypersurface point and the conjugate input pair as a triple which are presented to an oracle for classification [3]. Results of the oracle's classification are used for further training of the neural network. If all three points of a triple are classified as the same class by the oracle, only the point on the hypersurface is used as an input pattern for training. Otherwise, all three points are used as input patterns for training. Cohn et al. [1] propose the SG-network based on the version space search to identify training examples from the region of uncertainty [1]. These two selective sampling approaches, also belonging to the category of boundary concentration strategy, have the limitation that they are only suitable for relatively simple concept classes. When the concept classes become complex, it is difficult to compute and maintain an accurate region of uncertainty (the gray area of the boundary for concept classes).

In this article, we propose the *nearest-neighbor heuristic* approach to select the informative examples for the active learning. The neighborhood radius is determined by the discretized cluster which is generated by a discretization algorithm called the *Chi-Merge algorithm* [9]. The Chi-Merge algorithm is aimed at collecting adjacent values which belong to the same classification together as a cluster. The two contiguous clusters can be merged together into one cluster depending on the χ^2 value. The details will be discussed in subsection 4.3. From this, subsection 4.4 summarizes the knowledge refinement on the feedforward neural network.

The feedback of performance measured from the hypothesis evaluation is the motivation of revision. The hypothesis revision mechanism consists of the following functions: (1) detecting the defective hypotheses, (2) extracting the salient attributes, and (3) generating the heuristics of experimental design for the next run of experimentation. The performance of the hypothesis is measured by comparing the predicted value with the actual output value. The detection of defective hypotheses is to search for hypotheses that have substantial differences between expectation and reality. Those hypotheses performing poorly (under some given threshold) in solving problems will be the focus of revision. From the collection of defective hypotheses, we can extract the attributes that make the effective hypotheses different from the defective hypoth-eses. These attributes will contribute to the hypothesis revision that will generate the heuristics of experimental design for the next run of experimentation.

3. Backpropagation neural networks

According to Lippman's classification for neural network classifiers [13], the focus in this article is the continuous-valued input, supervised, multi-layer perceptron classifier. This is a feedforward neural network in which a unit of layer i sums a collection of weighted inputs from layer $i - 1$ and passes the result through a non-linearality to units of layer $i + 1$. This process will continue until outputs are generated from the output layer. The learning in this network is the tuning of the weight between units in order to stabilize the network for correct outputs.

The backpropagation network (BPN) represents a feedforward network whose interconnection weights are updated through the backward propagation of residual error beginning from output units. Let $w_{ji}(t)$ denote the weight from unit i to j after epoch t. The weight assigned to network in the $(t + 1)$th epoch training is $w_{ji}(t + 1) = w_{ji}(t) + \Delta w_{ji}$. The performance of the network mainly depends on the method used to decide the Δw_{ji}. There are several methods to decide the Δw_{ji}, such as conventional back propagation using the first derivative of objective function proposed by Rumel-hart et al. [18], Newton's method using the first and second derivatives [16], and quick propagation by Fahlman [4, 5]. Among these, we will adopt the back propagation using the first derivative of objective function proposed by Rumelhart et al. [18] as the representative of neural networks for the LEM.

We follow Freeman and Skapura's book [6, p.102] to describe the procedure of training feedforward neural networks using the backpropagation algorithm. The following steps iterate until the standard deviation of the error terms is below a given threshold. The detailed formulae are described in the appendix.

Step 1. Apply the input example to the input units.

Step 2. Calculate the net-input values to the hidden layer units.

Step 3. Calculate the outputs from the hidden layer.

Step 4. Calculate the net-input values to the output units.

Step 5. Calculate the outputs of the output units.

Step 6. Calculate the error term for the output units.

Step 7. Calculate the error term for the hidden units.

Step 8. Update weights on the output layer.

Step 9. Update weights on the hidden layer.

In this network training, we consider both training rate η and momentum α. We also deal with the slow improvement problem of error deduction by keeping track of the standard deviation of error rate deduction from a moving sequence of epochs. If the standard deviation is lower than a given threshold over a moving sequence of epochs, we stop the training process.

4. Active training of backpropagation neural networks

The active training of backpropagation neural networks using the LEM *iteratively* invokes the following procedures:

Step 1. **Experimentation**:

- performing experiments.

Step 2. Hypothesis formation:

- preparing examples, and
- training neural network.

Step 3. Problem solving and performance evaluation:

- using the trained network to test on holdout examples, and
- if the performance is satisfied, then **stop**.

Step 4. Hypothesis revision:

- analyzing the weight space of learned networks, and
- selecting salient attributes and effective value ranges, go to step 1.

The experimentation follows the guidelines for systematic experimental design as described in subsection 2.1, and training neural networks as introduced in section 3. The following subsections will describe the analysis of the network and the selection of informative attributes in order to revise the networks.

4.1. The analysis of neural networks

Since the architecture of neural networks is a sub-symbolic structure, in contrast to the transformation from decision trees to rules, we cannot straightforwardly convert it into explicit rules. The knowledge of solving target problems or classifying phenomena is embedded in the network, which is determined by the structure of units and connection weights. Therefore, the knowledge refinement for this kind of sub-symbolic representation mainly lies in the exploration of the relationship between input pattern and output values through the connection. The analysis of relative strength for input variables with their output variables provides valuable information for experimental design to select informative examples for continuous improvement.

In a neural network, the weight space provides the connection strength between two units. A weight space analysis approach proposed by Yoon et al. [21] is used to extract explanation rules from networks. We adopt a similar approach to compute relative salience and effect of input attributes. For each attribute i (unit i of the input layer) and classification j (unit j of the output layer), we can compute the *relative salience* between this attribute and classification. Assume that a neural network has one input layer with m units, one hidden layer with n units, and one output layer with p units. The relative salience is computed as

$$RS_{ij} = \frac{\sum_{k=1}^{n} (w_{ki} \cdot w_{jk})}{\sum_{i=1}^{m} \left| \sum_{k=1}^{n} (w_{ki} \cdot w_{jk} \right|},$$

where RS_{ij} is the relative salience between the ith input unit and the jth output unit, w_{ki} is the weight between the ith input unit and kth hidden unit, w_{jk} is the weight between the kth hidden unit and jth output unit. RS_{ij} is the ratio of the strength between the ith input unit and jth output unit over the total strength of all of the input units and jth output unit.

We apply the weighted average to compute the *relative effect* of each input attribute. For the ith input value I_i, and the jth output value O_j, RS_{ij}, the effect of I_i upon O_j, is obtained by

$$RE_i = O_j \frac{I_i \cdot RS_{ij}}{\sum_{k=1}^{m} I_k \cdot RS_{kj}}.$$

RE_{ij} provides us with the relative importance of an attribute toward the output classification. An attribute with a higher absolute value of RE contributes more information toward the classification. We can set up a threshold value β as the lower bound to select relative effective attributes. In statistics jargon, this relative effects RE is equivalent to *main effects*.

4.2. *Selection of training examples*

Among the heuristics of the example selection, the near miss and the near hit, which search on the boundary of target concept, are usually used. These approaches are mainly used to clarify the boundary of target concept. In the learning by experimentation environment, the classification of examples is assigned after experiments, and before which there is no pre-determined boundary about the target knowledge. The performance of the neural network by testing examples is the only information fed back to the learning agent. To analyze those examples which are classified incorrectly is the focus of our heuristic.

We shall call the heuristic used here the *nearest-neighbor strategy*. For these false examples, we focus on the neighbor of each attribute value. The range of the nearest neighbor is defined by the discretization process which groups contiguous values of an attribute having the same classification into a cluster. For example, let a false example consist of a set of salient attributes $\{a_s\}$ and a set of unsalient attributes $\{a_{-s}\}$. Let the value of a_s be υ_s, and belong to a cluster between lb_k and rb_k; that is, $lb_k \leq \upsilon_s \leq lb_k$. υ_s's nearest neighbor is defined as being between lb_k and rb_k. The value in the next experiment will be selected randomly from range $[lb_k \,..\, rb_k]$ of a_s. The values for these unsalient attributes will be selected randomly from the range of the possible smallest and largest values of that attribute.

Definitions.

- BPN: the backpropagation neural network.
- S: the example space, $S = S_{train} \cup S_{test}$ and $S_{train} \cap S_{test} = \emptyset$.
- S_{test}: the testing examples, $S_{test} \subseteq S$.
- S_{false}: the testing examples which are classified incorrectly, $S_{false} \subseteq S_{test}$.
- S_{train}: the training examples, $S_{train} \subseteq S$.
- $[a_1, a_2, \ldots, a_n]$: a sequence of attributes of the example.
- A: the attribute space. i.e., $A = \{a_1, a_2, \ldots, a_n\}$.
- $[\upsilon_1, \upsilon_2, \ldots, \upsilon_n, c_i]$: a sequence of values for the $[a_1, a_2, \ldots, a_n]$ of the example.
- c_i: the classification.
- RE_{ij}: the effect of the ith attribute upon the jth class.
- β: the threshold value for RE_{ij}.
- A_{ef}: set of attributes whose RE value $\geq \beta$.
- S_{output}: the attribute-value pair for those salient attributes; i.e., $S_{output} = \{a_{ef} : d_{ef}\}$.

Heuristic.

Input: *S, BPN.*

Process:

```
REPEAT
Step 1. Choose $s = [\upsilon_1, \upsilon_2, \ldots, \upsilon_n, c_i]$ from $S_{false}$.
Step 2. Compute $RE_{ij}$ for $s$, where $i = 1$ to $n$.
Step 3. Assign attributes which $RE \geq \beta$ to $A_{ef}$.
Step 4. $A_{-ef} = A - A_{ef}$.
Step 5. Discretize attributes in $A_{ef}$. $\forall a_{ef} \in A_{ef}$, $a_i$ clusters into intervals $[lb_1 \,..\, rb_1]$,
        $[lb_2 \,..\, rb_2], \ldots, [lb_m, \ldots, rb_m]$.
```

Step 6. $a_{ef} \in A_{ef}$, $lb_i \leq v_{ef} \leq rb_i$, randomly generate a value d_{ef}, where $lb_i \leq d_{ef} \leq rb_i$.

Step 7. $S_{output} = S_{output} \cup \{d_{ef}\}$.

Step 8. $S_{false} = S_{false} - s$.

UNTIL $S_{false} = \varnothing$.

Output: S_{output}.

4.3. Attribute discretization method

The purpose of discretization is to convert continuous scale into discrete scale. The output of discretization is a set of clusters where values inside the same cluster have the same classification, and values in the contiguous clusters have a different classification. The discretization program based on the *Chi-Merge* algorithm [9] is used to generate the concise intervals for each numeric attribute. The Chi-Merge discretization algorithm begins with calculating the chi-square value for each pair of adjacent intervals, and then chooses two intervals with the lowest chi-square value to coalesce to one interval. This process is iterated to merge those intervals with lower chi-square value until every chi-square value of all pairs of intervals exceeds the *chi-square threshold.* The chi-square threshold is a significant level of chi-square distribution with a specific degree of freedom which is one less than the number of classes. For example, for a set of examples with two classes (i.e., one degree of freedom), the chi-square value at the 0.9 percentile level is 2.706. The formula of computing the chi-square value is

$$\chi^2 = \sum_{k=1}^{m} \sum_{j=1}^{k} \frac{(A_{ij} - E_{ij})^2}{E_{ij}},$$

where m is the number of intervals to be compared, k is the number of classes, A_{ij} is the number of entries in the ith interval jth class, E_{ij} is the expected frequency of A_{ij}. From this Chi-Merge algorithm, a sequence of clusters, $[lb_1 \mathinner{..} rb_1]$, $[lb_2 \mathinner{..} rb_2]$,..., $[lb_m \mathinner{..} rb_m]$ is obtained, where $lb_1 < rb_1$ and $rb_{i-1} < lb_i$. The boundary of a cluster serves as the range for the random selection in the heuristic of choosing values for attributes.

4.4. Knowledge refinement on neural networks

Based on the feedforward neural network architecture, we propose a knowledge refinement algorithm which can analyze the weight space of learned network, locate the salient variables, and suggest the range of possible values for experimental design. We describe this knowledge refinement algorithm as follows.

Definitions.

- *BPN*: the backpropagation neural network.

- S: the example space, $S = S_{train} \cup S_{test}$ and $S_{train} \cap S_{test} = \emptyset$.
- S_{test}: the testing examples, $S_{test} \subseteq S$.
- S_{train}: the training examples, $S_{train} \subseteq S$.
- e: the prediction error rate of BPN.
- h : the error rate threshold given by the user.

Algorithm.

```
Input: $S_{init}$, $BPN_{init}$, $h$.
Process:
    $S = S_{init}$.
    $BPN = BPN_{init}$.
    $S = S_{train} \cup S_{test}$ and $S_{train} \cap S_{test} = \emptyset$.
    Test the accuracy of current $BPN$ using $S_{test}$.
    WHILE $e > h$
    Step 1. Apply Heuristic (subsection 4.2) to select attributes and values for experiments.
    Step 2. Generate new examples $S_{new}$ from experimentation.
    Step 3. $S = S \cup S_{new}$.
    Step 4. $S = S_{train} \cup S_{test}$ and $S_{train} \cap S_{test} = \emptyset$.
    Step 5. Modify $BPN$ using $S_{train}$.
    Step 6. Test the accuracy of current $BPN$ using $S_{test}$.
    END of WHILE
Output: $BPN$.
```

5. An example

We now demonstrate this active learning scheme on the sonar signal classification problem [7]. In this problem domain, 208 examples are collected by bouncing sonar signals off two objects, metal cylinder and rocks, at various angles and under various conditions. These angles and conditions compose the 60 attributes of the example set. Because we have no such equipment to do the experiments as required by the LEM, we have used the 208 examples as an example pool. To perform experiments is to select qualified examples from the example pool. By doing this, we can also show the performance of the active training of neural networks. The procedure can be summarized as follows:

Step 1. Divide the pool into two subsets: training and testing sets.

Step 2. Use the training examples to train the backpropagation network.

Step 3. Test the trained network using testing examples.

Step 4. Evaluate the performance of the network in terms of prediction accuracy.

Step 5. Analyze the weight space.

Step 6. Discretize the attributes.

Step 7. Decide the salient attributes and effective value ranges.

Step 8. Select new examples from the example pool according to the decision made in the last step.

Step 9. Combine the new selected examples and original examples as the current training set.

Step 10. Reload the weight space into network.

Step 11. Go to step 2.

Following the above steps, we obtained the following two sets of results shown in tables 2 and 3. In these two learning process, we set $\beta = 0.3$; i.e., we select attributes whose relative effect (*RE* value) is greater than or equal to 0.3. We assigned 20% to h, the error rate threshold; i.e., when the prediction error rate is greater than 20%, we

Table 2

The first set of results.

Training set	New examples	Testing examples	Accuracy (%)
50	0	22	63.6
88	38	22	72.7
91	3	22	63.6
105	14	22	81.8

Table 3

The second set of results.

Training set	New examples	Testing examples	Accuracy (%)
70	0	30	60.0
86	17	30	66.7
97	11	30	66.7
117	20	30	73.3
122	5	30	70.0
132	10	30	73.3
149	17	30	70.0
174	25	30	86.7

continue the learning process. These results are obtained from a network with 60 input units, 10 hidden units, and one output unit. Such network structure was suggested by Gorman and Sejnowski [7].

In the first set of results (in table 2), 50 examples were randomly selected for training the neural network. The accuracy rate was only 63.6%. From the feedback of the falsely classified examples, we selected 38 more examples along with the first round of 50 examples (in total, 88 examples), which were used for training the neural network. The resulting performance in terms of classification accuracy increased to 72.7%. The LEM process continued until the accuracy rate exceeded 80%; e.g., the accuracy rate was 81.8% after the fourth iteration of the LEM process. The second set of results (in table 3) demonstrates a similar phenomenon as that of the first set of results

6. Conclusions

In this paper, we proposed a learning by experimentation methodology (LEM) to perform active learning. A backpropagation neural network was used as the learning mechanism. In order to embed the active learning capability into the network, we adopted a self-refinement method to revise knowledge. This self-refinement method consists of analyzing knowledge embedded in the connection weights, selecting salient attributes and effective value range for performing the experiments. In order to speed-up the consequent training on neural network, we saved the learned weights, and then loaded them into the network for the next round of training. The stored weight space works as a memory, which avoids training from scratch in each round of learning, and increases the learning process as well. The demonstration on sonar signal classification problem shows that this whole active learning methodology works, although we are unable to conduct experiments to produce new examples for knowledge revision.

In the future, we will apply this methodology on simulation models of the multi-agent information systems. In this problem domain, the simulation models will allow us to conduct experiments to generate new examples for knowledge revision. In these settings, we can wholly illustrate the performance of this methodology.

Appendix: Training a feedforward neural network using the backpropagation algorithm

The procedure of training a feedforward neural network using the backpropagation algorithm as given by Freeman and Skapura [6, p. 102] is listed as following. These nine steps iterate until the standard deviation of error terms is below a certain threshold. Assume there are m input units, n hidden units, and p output units.

Step 1. Apply the input vector $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^t$ to the input units.

Step 2. Calculate the net-input values to the hidden layer units:

$$net_j^h = \sum_{i=1}^{m} W_{ji}^h x_i + \theta_j^h.$$

Step 3. Calculate the outputs from the hidden layer: $H_j = f_j^h(net_j^h)$.

Step 4. Calculate the net-input values to the output units:

$$net_k^o = \sum_{j=1}^{n} W_{kj}^o H_j + \theta_k^o.$$

Step 5. Calculate the outputs of the output units: $O_k = f_k^o(net_k^o)$.

Step 6. Calculate the error term for the output units: $\delta_k^o = (y_k - o_k) f_k^o(net_k^o)$.

Step 7. Calculate the error term for the hidden units:

$$\delta_j^h = f_j^h(net_j^h) \sum_k \delta_k^o w_{kj}^o.$$

Step 8. Update weights on the output layer:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta\delta_k^o H_j + \alpha\Delta w_{kj}^o(t-1).$$

Step 9. Update weights on the hidden layer:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \eta\delta_j^h x_i + \alpha\Delta w_{ji}^h(t-1).$$

References

[1] D. Cohn, L. Atlas and R. Ladner, Improving generalization with active learning, Machine Learning 15(1994)201–221.

[2] P.A. Devijer and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice-Hall, London, 1982.

[3] R.C. Eberhart, The role of genetic algorithms in neural network query-based learning and explanation facilities, in: *COGANN-92*, 1992, pp. 169–182.

[4] S. Fahlman, An empirical study of learning speed in a back-propagation network, Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1988.

[5] S. Fahlman, Faster-learning variations on back-propagation: An empirical study, in: *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, Los Altos, CA, 1988.

[6] J.A. Freeman and D.M. Skapura, *Neural Networks: Algorithms, Application, and Programming Technology*, Addison-Wesley, 1992.

[7] R.F. Gorman and T.J. Sejnowski, Analysis of hidden units in a layered network trained to classify sonar targets, Neural Networks 1(1988)75–89.

[8] J-N. Hwang, S-O. Choi and R.J. Marks, Query-based learning applied to partially trained multilayer perceptrons, IEEE Transaction on Neural Networks 2(1991)131–136.

[9] R. Kerber, Chimerge: Discretization of numeric attributes, in: *11th National Conference on Artificial Intelligence*, 1992, pp. 123–128.

[10] D. Kulkarni and H.A. Simon, Experimentation in machine discovery, in: *Computational Models of Scientific Discovery and Theory Formulation*, J. Shranger and P. Langley, eds., Morgan Kaufmann, San Mateo, CA, 1990, pp. 255–273.

[11] P. Langley, G.L. Bradshaw and H.A. Simon, Rediscovering chemistry with the bacon system, in: *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., Morgan Kaufmann, Los Altos, CA, 1983, pp. 307–330.

[12] D.B. Lenat, The role of heuristics in learning by discovery: Three case studies, in: *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., Morgan Kaufmann, Los Altos, CA, 1983, pp. 243–306.

[13] R. Lippman, An introduction to computing with neural nets, IEEE ASSP Magazine, April 1987, pp. 4–22.

[14] T.M. Mitchell, P.E. Utgoff and R. Banerji, Learning by experimentation: Acquiring and refining problem-solving heuristics, in: *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., Morgan Kaufmann, Los Altos, CA, 1983, pp. 163–190.

[15] B. Nordhausen and P. Langley, An integrated approach to empirical discovery, in: *Computational Models of Scientific Discovery and Theory Formulation*, J. Shranger and P. Langley, eds., Morgan Kaufmann, San Mateo, CA, 1990, pp. 97–128.

[16] S. Piramuthu, N. Raman and M.J. Shaw, Learning-based scheduling in a manufacturing flexible flow line, Technical Report, Beckman Institute, University of Illinois at Urbana-Champaign, 1993.

[17] L. Rendell and H. Cho, Empirical learning as a function of concept character, Machine Learning 5(1990)267–298.

[18] D. Rumelhart, G. Hinton and J. McClelland, A general framework for parallel distributed processing, in: *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol. 1, The MIT Press, Cambridge, MA, 1987, Chap. 2.

[19] S.C. Park, N. Raman and M.J. Shaw, Intelligent scheduling with machine learning capabilities: The induction of scheduling knowledge, IIE Transaction 24(1992)156–168.

[20] P.H. Winston, Learning structural descriptions from examples, in: *The Psychology of Computer Vision*, P.H. Winston, ed., McGraw-Hill, New York, 1975.

[21] Y. Yoon, T. Fuimaraes and G. Swales, Integrating artificial networks with rule-based expert systems, Decision Support Systems 11(1994)497–507.

[22] R-T. Zhang and G. Veenker, Neural networks that teach themselves through genetic discovery of novel examples, in: *1991 IEEE International Joint Conference on Neural Networks*, Vol. 1, IEEE, New York, 1991, pp. 690–695.