論文2003-40CI-3-3

개념학습을 위한 논리적 진화방식

(Logical Evolution for Concept Learning)

朴 明 銖 *, 崔 鎭 榮 **

(Myoung Soo Park and Jin Young Choi)

요 약

이 논문에서는, 이진 논리 함수(binary logic function)로 표현되는 개념들에 대한 새로운 학습방법인 논리적 진화방식(Logical Evolution)을 제안하였다. 그리고 이 방법을 통해 기존 귀납학습의 문제점들을 해결하고자 시도하였다. 사용하는 특징이 사전지식의 영향을 적게 받도록, 학습과정에서 얻어진 정보를 이용하여 특징을 생성하고 동시에 이를 이용하여 학습한다. 그리고 전체 자료가 아니라 개별 자료를 이용하여 특징생성 및 학습을 수행한다. 그 결과 새로운 문제가 주어지거나 입출력이 변경되는 경우에도, 이전의 특징을 재사용할 수 있으며 경우에 따라서는 보다 효율적인 학습이 가능하다. 논리적 진화방식은 5가지 연산으로 구성되며, 이러한 연산들은 특징생성 및 학습 과정에서 논리적 평가방식(logical evaluation)에 의해 적절하게 선택되고 실행된다. 제안된 방법의 성능을 평가하기 위해서 MONK 문제와 새로 정의한 다른 문제를 이용하였다.

Abstract

In this paper we present Logical Evolution method which is a new learning algorithm for the concepts expressed as binary logic function. We try to solve some problems of Inductive Learning algorithms through Logical Evolution. First, to be less affected from limited prior knowledge, it generates features using the gained informations during learning process and learns the concepts with these features. Second, the learning is done using not the whole example set but the individual example, so even if new problem or new input-output variables are given, it can use the previously generated features. In some cases these old features can make the learning process more efficient. Logical Evolution method consists of 5 operations which are selected and performed by the logical evaluation procedure for feature generation and learning process. To evaluate the performance of the present algorithm, we make experiments on MONK data set and a newly defined problem.

Keywords : Logical Evolution, binary logic concept, inductive learning, constructive induction, feature generation

1. 서 론

개념학습은 일종의 분류(classification) 문제이다. 주어지는 자료가 원하는 집합에 속하는지를 결정하는 함수를, 주어진 학습 자료로부터 결정하는 문제로 정의된다[1]. 이 때 이 함수를 개념(concept)라고 부른다. 개념학습(concept learning)은 매우 폭넓은 여러 가지 문제에 응용이 가능하다. 개념학습은 실제 상황에서 접하게

* 學生會員, 서울大學校 電氣工學部
(School of Electrical Engineering & Computer Science, Seoul National University)
** 正會員, 서울大學校 電氣工學部
(School of Electrical Engineering & Computer Science, Seoul National University)
接受日字:2000年8月11日, 수정완료일:2003年3月31日

되는 여러 가지 문제들의 추상적인 일반형(general form)으로 생각할 수 있다. 지식 발견 및 자료 수집(KD&D: knowledge discovery and data mining)[2] 분야의 요약(summarization), 예측(prediction) 등의 여러 가지 주제에서부터 유전 프로그래밍(Genetic Programming)에 의한 규칙 생성[3], 신경 회로망(neural network)의 학습[1, 4] 등 많은 종류의 학습은 개념 학습의 틀 위에서 다루어질 수 있다. 어떤 표현형(논리식, 수학식)으로 해를 표시하며, 해를 찾기 위해서 어떤 방법을 사용할 것인가에 차이가 있을 뿐이다.

이 논문에서 제안하는 논리적 진화방식(logical evolution)은, 이러한 개념학습 문제의 해결을 목적으로 한다. 개념학습 문제 중에 제일 간단한 것이 논리 함수(logic function)의 학습이다. 적은 수의 학습 자료를 사용하여 많은 자료에 적용 가능한, 즉 충분히 큰 일반화 성능(generalization performance)을 가지는 논리 함수 형태의 개념을 학습하는 문제이다. 논리 함수의 학습은 다른 문제들에 대해 비교적 단순하다. 반면 본질적인 면이 명확히 드러나고, 따라서 기존 알고리듬의 여러 가지 특징과 문제점들에 대한 보다 명확한 고찰이 가능하다. 일단 논리함수에 대한 해결이 이루어진 후에는 다치 논리(multi-valued logic)로의 확장, 실수 값 자료(real-valued data)를 이용하기 위한 양자화(quantization) 등에 관련된 차후 연구들을 통해, 실제적인 많은 문제들의 해결에 이용될 수 있다. 이 논문에서는 일단 논리적 개념, 그 중 제일 단순한 이진 논리 함수(binary logic function)의 학습에 초점을 맞춘다.

논리적 진화방식(logical evolution)은 이진 특징과 개념을 진화시켜 나가는 학습 방식이다. 학습된 이진함수는 망 구조(network structure)로 표현된다. 망 구조 내의 특징 노드들은 서로 결합되고 성장하면서 적합한 새로운 특징노드로 바뀌어지고, 원하는 개념을 기술하기에 부적합한 특징노드들은 삭제된다. 이를 위해 5가지 연산(creating, visualizing, deleting, specializing, fixing)과 연산들의 선택방법이 사용된다. 해가 저장되는 망 구조가 주어진 자료 각각에 대해 보이는 반응들에 따라 연산이 선택되어 망 구조를 변경한다. 이 때 망 구조 위에서 생성되고 변경된 특징들은, 학습하기를 원하는 개념의 일부, 즉 부분적인 조각 개념(partial piece of concept)들로 볼 수 있다. 그리고 이러한 부분적인 조각 개념들의 집합에 대응되는 망 구조가 최종적으로 학습된 개념에 해당된다.

논리적 진화방식은, 이전에 제안된 여러 귀납 학습 알고리듬들[5-7]과 여러 가지 면에서 차이가 있고 기존 귀납 학습 알고리듬의 여러 가지 문제점을 해결할 수 있다. 첫째로 특징 생성과 학습을 동시에 진행시켜, 사전적으로 특징을 선택할 경우에 발생하는 사전지식의 영향을 줄일 수 있다. 따라서 문제에 대한 해석 등 사전작업에서 오는 잘못된 해석과 시행착오의 영향이 적다. 유사한 목적을 가진 연구로 건설적 귀납(constructive induction)[8-11]이 있다. 둘째로, 이전의 학습경험을 다시 사용하여 학습할 수 있다. 어떤 문제를 해결한 경우에 얻어지는 특징 노드들은 다른 문제의 해결에 다시 사용할 수 있다. 이러한 것을 위해 특징 노드의 생성이 개별 자료에 의존하고 전체 자료 분포의 영향을 적게 받도록 하였다. 따라서 새로운 문제가 주어지는 경우에도 바로 활용할 수 있고, 이전의 학습 결과를 부분적으로 이용하여 더욱 빠른 속도로 학습할 수 있다.

논문의 구성은 다음과 같다. 우선 Ⅱ장에서 논리적 진화방식의 해를 저장하는 데에 사용되는 망 구조에 관해 정의하고 설명한다. 망을 이루는 노드와 연결에 대해 살펴보고, 노드가 가지는 성질들, 정확성(correctness), 특수성(speciality), 고정성(fixedness), 적합성(fitness)을 설명한다. Ⅲ장에서는 학습 알고리듬에 대해 설명한다. 우선 사용되는 5가지 연산(operation)들, 생성(creating), 가시화(visualizing), 고정화(fixing), 특수화(specializing), 제거(removing)를 정의한다. 이 연산들은 망 구조를 변경시키는 것으로 실질적으로 학습을 구현한다. 그리고 망과 노드들의 반응에 따라 적절한 연산들을 선택하기 위한, 논리적 평가방식을 제시한다. 설명된 알고리듬의 성능은 Ⅳ장에서 모의 실험을 통해 평가한다. 학습 성능의 비교 평가(benchmarking)를 위해, 널리 사용되는 MONK 자료[12]를 사용하여 다른 알고리듬과 비교한다. 그리고 새로 정의된 간단한 문제를 통해 알고리듬의 특징을 명확히 한다. Ⅴ장에서 결론과 추후과제에 대해 언급한다.

Ⅱ. 학습망 구조(learning network structure)

본 논문에서 제안하는 학습방법의 핵심은 학습패턴이 주어졌을 때 특수한 구조의 학습망을 변경함으로써 그 학습패턴을 기억시키는 것이다. 또한 추가적인 학습

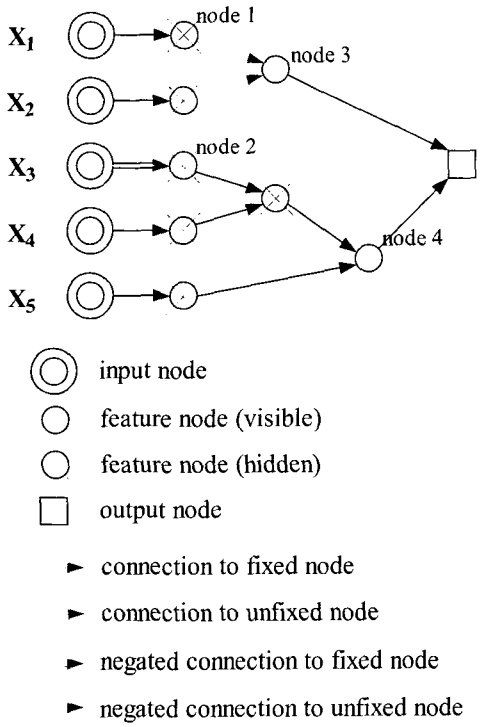


그림 1. 망 구조 및 망구성 기본 요소
Fig. 1. Network structure and basic components of network.

패턴이 주어지면 기존에 학습된 망의 자원(resource)을 최대한 활용하고 필요시 추가적인 자원을 생성하여 이미 기억된 내용의 손상이 없이 새로운 패턴을 기억시키는 것이다. 이 절에서는 제안되는 학습방법의 기본이 되는 학습망 구조에 대해 기술한다.

모든 이진 논리함수는, 입력에 해당하는 변수들과 그것의 부정들에 대한 논리곱을 취하여 만들어진 항에 대한 논리합으로 표현될 수 있다. 따라서 이러한 표현에 해당하는 망 구조를 사용하면 모든 논리함수를 나타낼 수 있다. 본 논문에서 사용된 망 구조는 <그림 1>과 같은 형태를 취한다.

<그림 1>에서 망은 세 종류의 노드(node)들로 구성된다. 노드에는 입력노드(input node)와 특징노드(feature node), 출력노드(output node)가 있다. 입력노드의 값은 외부에서 입력되는 값에 의해 결정된다. 그리고 특징노드의 값은 연결된 노드들(입력노드 혹은 다른 특징노드)의 값에 논리곱(logical and ; $\wedge$)을 취하여 결정된다. 출력노드의 값은 연결된 특징노드들의 값에 논리합(logical or ; $\vee$)을 취하여 결정된다. 단, 특징노드의 경우는 입력노드와 직접 연결될 수 있고, 입력노드의 값을 부정(negation ; $\bar{x}$)한 값을 가질 수 있다. 특징노드 중에서 출력노드와 연결된 것을 보이는(visible) 특징노드라고 하고, 연결되지 않은 것을 숨겨진(hidden) 특징노드라고 한다. 숨겨진 특징노드는 추가적인 패턴을 학습할 때 활용될 수 있다. 이러한 정의에 따르면 <그림 1>의 망은

$$f(x_1, x_2, x_3, x_4, x_5) = (x_1 \wedge x_2) \vee (\bar{x_3} \wedge x_4 \wedge x_5) \quad (1)$$

라는 이진 함수를 표현하고 있다. 특징노드는 이진함수 표현에서 항(term)에 대응한다. 즉, $(x_1 \wedge x_2)$, $(\bar{x_3} \wedge x_4 \wedge x_5)$ 등에 해당한다.

일반적으로 논리함수는 입력에 해당하는 변수들과 그것의 부정들에 대한 논리곱을 취하여 만들어진 항에 대한 논리합으로 표현될 수 있다. 그런데 그러한 표현 방법은 유일하지 않다. 예를 들면 위의 (1)의 함수의 경우

$$f(x_1, x_2, x_3, x_4, x_5) = (x_1 \wedge x_2) \vee (\bar{x_1} \wedge \bar{x_3} \wedge x_4 \wedge x_5) \vee (\bar{x_2} \wedge \bar{x_3} \wedge x_4 \wedge x_5) \quad (2)$$

로도 표현할 수 있다. 따라서 이러한 경우에 대응하는 <그림 2> 또한 <그림 1>의 구조와 같은 함수에 해당하는 것이다. 그러므로 주어진 학습패턴에 대해 <그림 1>과 같이 학습되거나 또는 <그림 2>와 같이 학습되어도 틀린 것은 아니다. 이러한 경우는 매우 많이 존재하며 학습방법, 학습되는 패턴의 순서에 따라 달라질 수 있다.

이러한 경우 중 일부는 다른 함수의 학습에 이용하기에 보다 유용할 수 있다. 위에서 든 예에서 두 가지

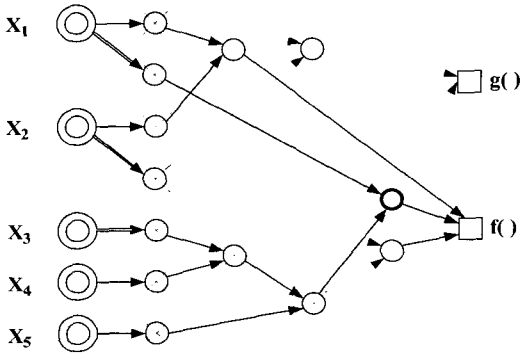


그림 2. f(·)에 대한 다른 망 구조 및 g(·)에 대한 추가 학습구조
Fig. 2. network structure for f(·) and g(·).

의 경우 중, $(\overline{x_1} \wedge \overline{x_3} \wedge x_4 \wedge x_5)$ 또는 $\overline{x_2} \wedge \overline{x_3} \wedge x_4 \wedge x_5$을 포함하는 다른 함수, 예를 들어 $g(x_1, x_2, x_3) = (x_1 \wedge \overline{x_2}) \vee (\overline{x_1} \wedge \overline{x_3} \wedge x_4 \wedge x_5)$을 학습할 때에 이전의 노드들을 이용하고자 한다면 보다 도움이 되는 쪽은 <그림 2>와 같은 경우가 될 것이다. <그림 2>에서 보듯이 이전의 자원을 활용하여 g의 함수를 용이하게 학습할 수 있음을 알 수 있다.

본 논문에서는 이전의 학습된 망 구조를 재활용하여 이후의 학습을 효과적으로 수행하는 학습알고리듬을 제안하는 것이다. 그러기 위해서는 전체자료의 분포에 의존적인 기존의 학습 알고리듬들은 곤란하다. 이러한 알고리듬에서는 특정한 문제를 풀기 위해 주어지는 자료에 최적화된 하나의 구조가 결정되기 때문이다. 그러한 구조는 다른 문제를 푸는 데에 적합하지 않는다. 예를 들어, 어떤 문제 1을 ID3[5]로 푸는 데 있어서 최적화된 트리 구조는 다른 자료가 추출된 문제 2를 푸는 데 있어서는 적합하지 않다. 우선, 트리 구조의 구축과정에서 전체 자료집합을 반복적으로 참조하기 때문에, 다른 자료집합을 이용하는 문제 2의 경우에는 문제 1의 학습결과를 사용할 여지가 없다. 즉 재활용 가능성을 고려하지 않고 있기 때문에 활용하기 곤란하다. 또한 각각의 해는 서로 다른 자료구조, 즉 개별적인 트리 구조에 저장된다. 같은 트리 구조에 문제 2를 풀게 된다면 문제 1의 결과는 손상된다. 부분적으로 이전의 결과를 공유하기 곤란한 것이다. 이러한 문제점은 자료전체의 분포에 따라 그것에 최적화된 형태로 해가 결정되기 때문에 발생하는 것으로 볼 수 있다. 그러한 점은 개별적인 문제를 푸는 데는 장점이 되지만, 서로 다른 문제를 푸는 데 있어서 발생할 수 있는 공통된 특징들을 생성하고 교환하는 데에는 단점이 될 수 있는 것이다.

따라서 개별적인 문제에 대해 최적화된 구조라는 일반적인 목표는 본 연구의 알고리듬에서 고려하지 않고, 여기서는 단순히 주어진 자료에 대해 일관된 (consistent) 함수를 학습하는 것만을 고려한다. 대신, 이전의 학습된 망 구조를 재활용하여 새로운 문제를 빠른 속도로 학습할 수 있도록 하는 새로운 접근 방식을 구체화하는 것에 초점을 맞춘다. 즉 개별적인 문제에 대한 일반화능력의 향상보다, 여러 문제를 학습함에 따라 학습을 보다 쉽게 할 수 있는 새로운 방식을 시도하고자 한다.

Ⅲ. 논리적 진화방식 (Logical Evolution)

1. 알고리듬 개요

논리적 진화방식에 의한 학습은, 기본적으로 1) 이전에 유용했던 적이 있고, 2) 보다 구체적인 (정확한 표현식이 될 가능성이 높은) 특징들을 이용한다는 원칙에 기반하고 있다. 이것은 사람의 학습에 대한 관찰에서 비롯한 것이다. 새로운 상황이 주어졌을 때, 사람은 모든 상황을 처음부터 배워나가지는 않는다. 이전의 상황 중에 유용했고 현재의 상황과 유사한, 즉 현재를 정확하게 표현할 수 있는 것이 있다면, 이것을 이용하여 보다 빠른 속도로 학습해 나간다. 그럼으로써 보다 효율적인 학습이 가능할 수 있다. 논리적 진화방식은 이와 같은 학습 원칙을 적용하고자 하는 새로운 시도이다.

이러한 학습 원칙을 구현하기 위해, 논리적 진화방식에서는 위에서 설명한 망 구조를 이용한다. 그것에 해당하는 표현식이 주어진 자료를 잘 기술할 수 있도록 함과 동시에, 유용하고 구체적인 특징에 해당하는 노드들을 많이 생성시키고 재활용하도록 한다. 따라서 학습과정은 곧 망 구조를 변경하는 방법이 된다.

논리적 진화방식은 아래에서 설명될 기본적인 5가지 연산을 통하여 이루어진다. 자료가 주어지고 그 자료에 대해 노드의 값이 결정되면, 이 값을 토대로 적절한 연산을 선택하여 적용함으로써 망 구조를 변경시킨다.

2. 특징노드의 성질

먼저 학습 알고리듬에서 사용될 망 노드들의 성질들을 정의한다.

정의 3.1 [정확성(correctness)] : 특정 입력패턴에 대해 특징 노드의 출력이, 출력 노드의 원하는 출력 (desired output)과 같으면 그 특징노드는 정확성은 CORRECT하다고 하고, 그렇지 않으면 INCORRECT하다고 정의한다.

이 정확성은 특정패턴에 대한 특징노드의 유용성을 판단하기 위한 것이다. 이의 활용에 대해서는 뒤의 알고리듬 설명에서 상세히 기술한다. <그림 1>의 예에서 $(x_1, x_2, x_3, x_4, x_5) = (T, T, F, T, F)$가 주어지고 이에 대해 원하는 출력 값 $f_{desire} = F$ 라고 하자. 여기서 T는 True를 F는 False를 나타낸다. 이 경우 특징노드 4의

경우만 망의 원하는 출력 F와 같으므로 특징노드 4의 정확성은 CORRECT라는 값을 가지고 나머지는 INCORRECT이다.

정의 3.2 [고정성(fixedness)] : 원하는 망의 출력이 T인 최소한 하나의 입력에 대해, 출력을 T로 만드는 데에 이용된 적이 있다면 FIXED, 그렇지 않으면 UNFIXED 되어있다고 정의한다.

본 연구에서 제시하는 알고리듬에서는 원하는 출력이 T인 패턴에 대해 정확하게 기억하도록 한다. 그러기 위해 망 구조를 수정해야 하는데, 수정이 필요한 경우 중에 다음과 같은 경우가 있다. 원하는 출력이 F인 패턴에 대해서 T를 출력하는 고정된 특징노드는 잘못된 출력을 유발한다. 이를 방지하기 위해 새로운 특징노드를 생성하여, 잘못된 출력을 내는 특징노드와 F를 출력하는 다른 특징노드와 논리곱으로 결합함으로써 F를 출력하도록 한다. 이 특징노드는 원하는 출력이 T인 패턴에 대해 유용성이 검증되지 않았기 때문에 UNFIXED상태로 설정한다. 이 특징노드는 이후에 원하는 출력이 T인 패턴에 대해 올바로 반응을 하면 유용성이 검증되어 FIXED로 된다. 그러나 원하는 출력이 F인 패턴에 대해 다시 한번 틀리면 (T를 출력하면) 삭제된다.

정의 3.3 [특수성(speciality)] : 특징노드에 대응하는 항(term)에 포함되는 입력변수의 수를 특수성으로 정의한다. 변수의 수가 클수록 더 특수한 경우를 나타낸다.

<그림 1>에서 특징노드 4에 해당하는 항은 $\overline{x_3} \wedge x_4 \wedge x_5$이고, 여기에 사용된 입력변수는 3개 (x_3, x_4, x_5)이다. 따라서 특수성은 3이 된다. 특수성이 커진다는 것은 True를 내는 패턴수가 적어진다는 것을 의미한다. 특수성의 활용에 대해서는 아래의 책임성의 정의 및 학습 알고리듬 설명 시 상세히 기술된다.

정의 3.4 [책임성(responsibility)] : 특징노드의 출력이 출력노드의 출력과 같고, 출력노드와 연결되어 있으면 책임이 있다(respobsible)고 하고 그렇지 않으면 책임이 없다고 한다. 책임이 있는 특징노드 중 특수성이 가장 큰 노드가 가장 큰 책임이 있는 노드가 된다.

책임성은 출력에 가장 밀접한 역할을 하는 노드를 의미한다. 이의 활용에 대해서는 알고리듬 설명에서 상세히 기술된다. 이러한 성질들은 개별적인 자료 하나 하나가 주어짐에 따라 그 결과에 의해 바로 계산된다. 또한 이러한 특성 중 어느 것도 자료집합 전체의 분포에 의해 결정되지 않는다는 사실이다. 즉 특징노드의 특징들은 개별적인 자료에 따라 변형되지만, 전체의 분포에 직접적으로 의존하지는 않는다.

3. 학습 연산 (Learning operation)

학습에 사용되는 연산은 다음과 같이 크게 세 가지 방식으로 동작한다.

① 하나의 특징노드를 선택하여 그것을 제거하거나, 다른 특징노드들을 생성하는데 이용하는 것이다. 이때 선택되는 특징노드들은 현재 망의 입출력 결과에 대해, 제일 큰 책임을 가지는 노드들이다. 즉, 앞서 정의된 책임성이 제일 큰 특징노드가 선택되어진다. 여기에 해당되는 연산으로 제거(removing)와 특수화(specializing) 연산이 있다. 이 연산은 망의 올바른 출력이 F일 때 틀린 출력, 즉 T가 출력될 때 이를 수정하기 위해 행해지는 연산이다.

② 개별적인 특징노드를 선택하지 않고, 망에 새로운 특징노드를 추가하는 것이다. 여기에 해당되는 연산으로는 가시화(visualizing)와 생성(creating)이 있다. 이 연산은 망의 올바른 출력이 T일 때 틀린 출력, 즉 F가 출력될 때 이를 수정하기 위해 행해지는 연산이다.

③ 단순히 선택된 노드의 성질만을 변경하는 것이다. 이 때 선택되는 특징노드는, 첫 번째 경우와 마찬가지로 현재 망의 입출력 결과에 대해 제일 큰 책임을 가지는 노드들이다. 여기에 해당되는 연산으로 고정화(fixing)가 있다. 이 연산은 망의 올바른 출력이 T일 때 망이 맞게 반응을 할 때 특징노드들의 유용성을 확인하기 위해 행해지는 연산이다.

상술한 5개의 기본 연산을 상세히 설명하면 다음과 같다.

연산 3.1 [제거(removing)] : 원하는 출력이 F일 때 INCORRECT하고 책임이 가장 큰 노드가 UNFIXED 상태인 경우 망으로부터 제거한다.

원하는 출력이 F 일 때 INCORRECT(T를 출력하는)인 책임있는 특징노드가 있으면 망의 출력은 A가 되어 틀린 반응을 하게 된다. 이때 이 노드가 UNFIXED이면 망으로부터 제거하여 망의 출력이 T가 되는데 기여하지 않도록 한다. 이때 책임있는 모드가 여러 개일 경우 모두 제거해야 올바른 출력을 내게 될 것이다. 그러나, 여기서는 책임이 가장 큰 노드만을 제거한다. 이러한 방식을 취하는 것은, 책임이 적은 노드는 다른 패턴에 의해 학습에 사용될 수 있기 때문이다. 그럼으로써 옳은 출력을 내는 데 기여할 수 있다. 그러나 이 노드들이 다른 패턴에 의해 변하지 않고 틀린 반응을 하게 되면, 나중에 다시 연산이 적용되어 변경된다. 이는, 모든 패턴에 대해서 망이 올바른 출력을 내어야 학습이 종료되기 때문이다.

제거연산의 구체적인 예는 <그림 3>과 같다. <그림 3>에서 처음에 망 구조에 주어진 입력이 $(x_1, x_2, x_3, x_4,) = (T, F, T, T)$이고 이 입력에 대해 원하는 출력이 F라고 하자. 이때 노드 1은 INCORRECT하고 책임있는 특징노드이다. 이 노드는 UNFIXED이므로 단순히 망에서 제거하여 잘못된 반응을 하는 것을 방지한다. 그런데 이 노드가 FIXED이면 유용성이 검증된 노드이므로 제거를 하지 못한다. 이 경우에 다음의 특수화 연산을 하여 틀린 반응 (망의 출력이 T)을 하는데 기여를 못하도록 한다.

연산 3.2 [특수화(specializing)] : 원하는 출력이 F일 때 INCORRECT하고 책임이 가장 큰 특징노드가 FIXED 상태인 경우 출력노드로부터 분리시켜 숨겨진 (hidden) 상태로 바꾼다. 그리고 이 노드와 CORRECT 하고 특수성이 가장 큰 FIXED 특징노드를 논리곱으로 결합하여 새로운 노드를 생성시켜 출력노드와 연결한다. 새로 생성된 노드는 은닉시킨 노드에 비해 특수성이 커짐으로 이 연산을 특수화라고 정의한다.

여기서 책임이 가장 큰 노드만 특수화를 적용하는 것은 제거의 경우에서와 같은 이유이다. 새로 생성된 특징노드는 F를 출력하게 되어 CORRECT하고 책임있는 특징노드가 된다. 구체적인 예가 <그림 4>에 주어져 있다. <그림 3>와 마찬가지로 노드 1이 INCORRECT하고 책임있는 특징노드이다. 이번에는 FIXED이므로 제거를 할 수가 없다. 따라서 이 노드를 보이지 않는(hidden) 노드로 바꾼다(즉, 출력으로부터 분리한다). 그리고 CORRECT하고 특수성이 가장 크고 FIXED인 노드 2를 선택한다. 노드 2와 노드 1에 논리곱을 하는 새로운 노드 3을 생성하고, 출력노드에 연결한다. 그러면, 현재 주어진 학습패턴에 대해 노드 1은 T를 출력하고 노드 3은 F를 출력하게 되어, 노드 3은 노드 1에 비해 적은 수의 패턴에 대해서만 T를 출력하게 된다. 이 경우 노드 3은 노드 1에 비해 더 적은 즉, 더 특수한 경우의 패턴에 대해 T를 출력한다고 해서 더 특수해졌다고 한다. 정리하면, 주어진 입력패턴에 대해서 F가 옳은 출력인데, 지나치게 일반적인 노드가 있어서 이 입력에 대해서 T를 출력하도록 하는 경우에, 이 잘못된 노드를 수정하기 위해 특수화가 이용되는 것이다.

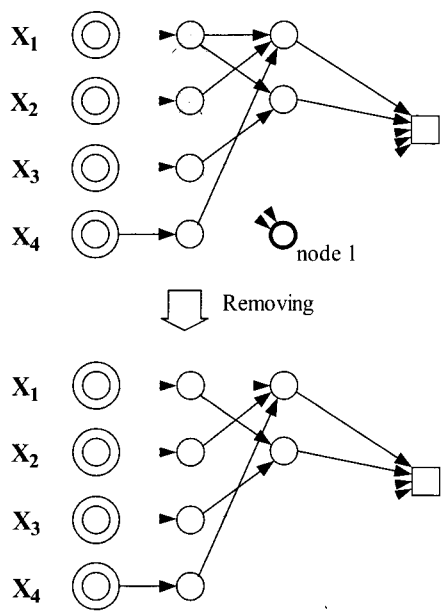


그림 3. 제거 연산을 적용한 망 구조의 예
Fig. 3. Example of network change by Removing.

선택된 노드에 대해 특수화가 적용될 것인지 혹은 제거가 적용될 것인지를 결정하는 기준은 고정성이다. 고정성이 FIXED로 된 특징노드는 적어도 하나 이상의 입력에 대해, 올바른 출력을 내기 위해 이용되는 것이므로 제거하지 않는다. 이러한 노드에 대해서는 특수화를 적용한다. 반대로 고정성이 UNFIXED인 경우는 아

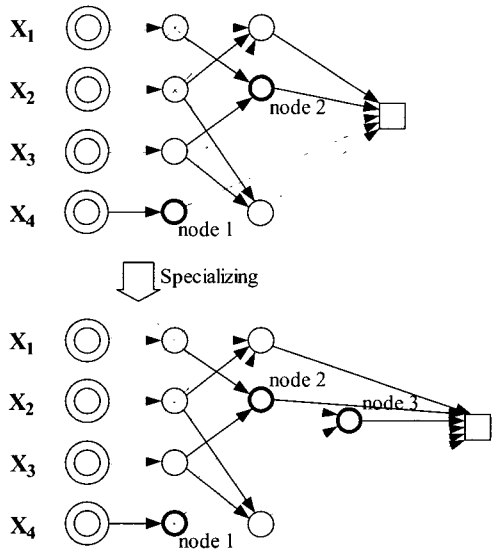


그림 4. 특수화를 적용한 망 구조의 예
Fig. 4. Example of network change by Specializing.

직까지 올바른 출력을 내는 데에 이용된 것이 없으므로, 망이 불필요하게 복잡해지지 않도록 제거 연산을 적용한다.

이상에서 설명된 제거와 특수화와 다른, 두 번째 방식의 연산에 속하는 것은 가시화와 생성이다. 앞서서 언급했듯이 망의 올바른 출력이 T인데 망은 틀린 출력, 즉 F를 출력할 때 이를 수정하기 위해 행해지는 연산이다. 이 경우는, 보이는 특징노드가 모두 F를 출력하고 있다는 것을 의미한다. 따라서 망이 T를 출력하도록 하기 위해 T를 출력하는 보이는 특징노드를 만들어 주는 연산들이다.

연산 3.3 [가시화(visualizing)] : 원하는 출력이 T일 때 망이 틀린 반응(즉, 출력이 F)을 하면 CORRECT하고 특수성이 가장 큰 숨겨진 특징노드를 보이는 노드로 바꾸고 출력노드에 연결한다.

이 경우는 CORRECT하고 책임있는 노드가 없다는 의미이다. 따라서 가시화된 특징노드는 CORRECT하고 책임있는 노드가 되어 망이 올바른 출력을 내도록 해주는 것이다. <그림 6>은 가시화의 예를 보여주고 있다.

그런데 CORRECT한 노드가 없는 경우는 가시화를 적용할 수가 없다. 이 경우에 다음의 생성 연산을 적용한다.

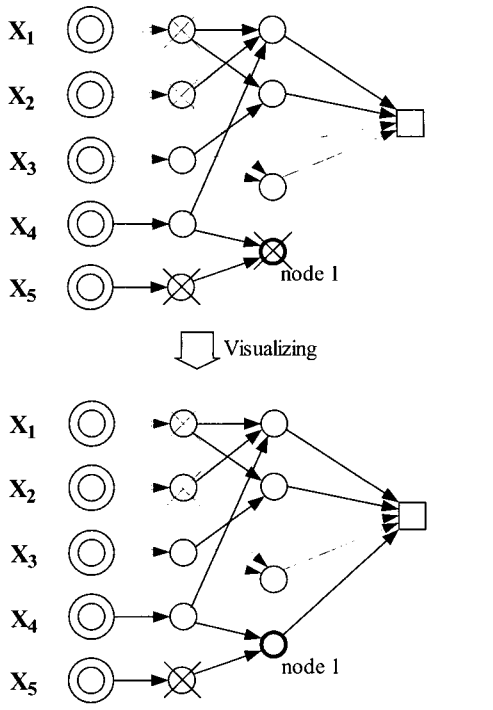


그림 5. 가시화를 적용한 망 구조의 예
Fig. 5. Example of network change by Visualizing.

연산 3.4 [생성(creating)] : 원하는 출력이 T일 때 망이 틀린 반응을 하는데 CORRECT 한 특징노드가 망에 없는 경우, 입력노드와 직접 연결되는 새로운 특징노드를 망에 추가 생성한다. 새로운 특징노드의 특수성은 1을 가지게 되고 CORRECT하도록 입력노드와 연결한다.

새로이 생성된 노드가 CORRECT하기 위해서는 1)입력노드의 값이 F이면 부정된 연결(negated connection)을 하고, 2)입력노드의 값이 T이면 정 연결을 하면 된다. 구체적인 예는 <그림 6>에서 보는 바와 같다. 그림에서와 같이 망에 아무 자원이 없을 경우에 생성이 일어난다. 이것은 학습을 위한 입력이 (x_1, x_2, x_3, x_4) = (T, F, T, T)인 경우이다. 그리고 다시 (x_1, x_2, x_3, x_4) = (F, T, F, F)가 입력으로 주어지면, 각 노드는 모두 F가 되어 CORRECT한 노드가 없으므로 생성이 한번 더 일어나게 된다. 이에 대한 예는 <그림 6>에서 보여진다. 이러한 경우를 제외하고는 생성 연산이 이루어지지 않는다.

앞서의 연산에 대한 정의에서, 특수화 시에 추가되는 특징노드는 주어진 패턴에 대해 F를 출력하도록 만들

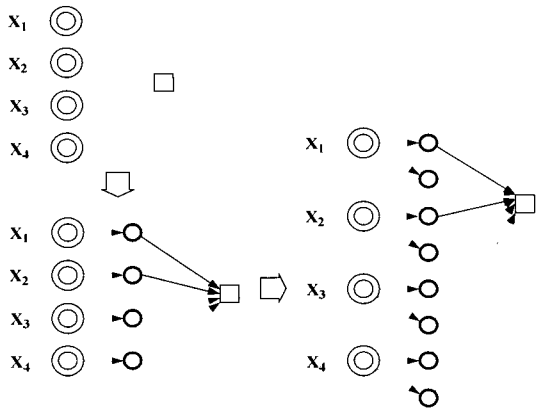


그림 6. 생성 연산을 적용한 망 구조의 예
Fig. 6. Example of network change by Creating.

어진다. 따라서 이 노드는 T를 출력하는 패턴에 대해서는 검증이 되어 있지 않다. 이 패턴에 대한 원하는 출력이 F이고 이 노드가 제일 책임이 크면 틀린 출력을 유발하므로 연산 3.1에 의해 삭제된다. 하지만 원하는 출력이 T인 패턴에 대해 이 특징노드가 T를 출력한 적이 있고, 이때 제일 책임이 큰 노드였다면, 삭제한다는 것은 그 패턴에 대한 기억을 잃어버리게 하는 것이다. 따라서 삭제되지 못하도록 FIXED 상태로 변경하는 것이다. FIXED 상태인 노드는 삭제가 불가능하므로 연산 3.2와 같이 특수화를 하게 되는 것이다. 이것을 위해 고정화가 이용되는데, 고정화는 다음과 같이 정의된다.

연산 3.5 [고정화(fixing)] : UNFIXED상태인 특징노드가 원하는 출력이 T인 패턴에 대해 CORRECT한 출력(T)을 하고 책임이 제일 크면 고정성(fixedness)을 FIXED로 변경한다.

각각의 연산은 결국 개별적인 입력에 대해, 학습자료에서 요구되는 출력을 낼 수 있는 항들을 생성을 통해 망에 만들어나가고, 특수화 및 삭제를 통해 요구되는 출력을 내는 데에 방해가 되는 것은 제거하거나 수정하는 것이다. 이 과정에서 이전에 유용했던 자원, 즉 노드를 가시화(visualizing) 함으로써 경제적으로 망이 구성되도록 한다.

4. 학습 절차(Learning Procedure)

이상에서 정의된 연산들은, 순차적으로 주어지는 학습자료 각각에 대해서 선택되고 실행된다. 제일 처음에

표 1. 논리적 진화방식 알고리듬
Table 1. Logical Evolution algorithm.

I. 원하는 출력이 T인 학습패턴에 대해
 1. 망이 틀린 반응 (F 출력)을 하면
 - CORRECT한 **숨겨진** 특징노드가 **있으면 가시화** 연산을 적용한다.
 - CORRECT한 **숨겨진** 특징노드가 **없으면 생성** 연산을 적용한다.
 2. 망이 옳은 반응 (T 출력)을 하면
 - CORRECT하고 UNFIXED노드중 책임이 가장 큰 노드에 대해 **고정화** 연산을 적용한다.
II. 원하는 출력이 F인 학습패턴에 대해
 1. 망이 틀린 반응 (T 출력)을 하면
 - INCORRECT하고 책임이 가장 큰 노드가 **UNFIXED** 상태이면 **제거** 연산을 적용한다.
 - INCORRECT하고 책임이 가장 큰 노드가 **FIXED** 상태이면 **특수화** 연산을 적용한다.
 2. 망이 옳은 반응 (F 출력)을 하면
 - 아무 연산도 하지 않는다.

는 입력노드와 출력노드만 존재한다. 그리고 첫 번째 입력패턴이 주어지면 생성 연산이 적용된다. 그 다음 입력패턴부터는 망의 상태 즉 노드의 값들이 결정되고, 이 값을 이용해서 상술한 바와 같이 필요한 연산이 적용된다. 그리고 모든 입력에 대해 아무런 연산이 수행되지 않으면 학습을 끝낸다. 어떤 문제에 대해 이미 학습된 기존의 망에 새로운 문제의 패턴을 학습할 수도 있다. 다만, 이때는 새로운 입력노드나 출력노드가 추가될 필요가 있다.

학습은 기본적으로 주어진 학습패턴에 대해 망이 잘못된 반응을 보일 경우 이를 수정하기 위해 망 구조를 변경하는 것이다. 학습 절차를 위에서 기술한 연산들을 이용하여 정리하면 다음 <표 1>과 같다.

<표 1>의 절차는 각각의 학습패턴에 대해 적용되며 모든 패턴에 대해 연산이 이루어지지 않으면 학습을 종료한다. 단, 효율적인 학습을 위해, 특수화 혹은 생성 연산을 적용한 후에는 원하는 출력이 T인 패턴을 선택하여, 특수화 혹은 생성에 의해 추가된 특징노드의 고정화를 조기에 유도하는 것이 바람직하다.

5. 장·단점 고찰

(1) 망 자원의 재활용

이상에서 기술한 알고리듬은, 기본적으로 주어진 학습자료의 입출력관계를 만족하는 함수를 학습할 수 있다. 즉 그러한 함수에 대응하는 망 구조를 생성할 수

있다. 또한 생성된 망 구조 내의 특징노드는 다른 문제를 푸는 데에 바로 재 이용될 수 있다. 특히 공통된 특징을 포함하는 문제를 해결하는 경우라면, 과거의 문제에 대한 특징노드를 재 이용하여 보다 빠른 속도로 적절한 특징노드를 생성하여 학습할 수 있다. 과거의 망 구조에서 얻어진 특징노드들을 가시화와 특수화를 통해, 새로운 망 구조 내의 특징노드를 빠르게 생성하기 위해 이용하는 것이다. 이러한 연산들은 모두 전체 자료를 함께 이용하지 않는다. 즉 현재 망 구조의 변경에 있어서 중요한 것은 개별적인 학습 자료 내의 입력에 대한 값이다. 따라서 이전의 학습에서 생성된 특징 노드를 현재의 문제를 푸는 과정에서 생성되는 것과 구분하지 않고 바로 이용할 수 있다. 또한 이를 통해, 하나의 망 구조 내에 여러 개의 이진 함수를 동시에 저장할 수 있고, 과거의 학습결과에 영향을 주지 않고 새로운 학습결과를 얻을 수 있다. 이는, 새로운 문제를 풀 때마다 학습자료 전체와 새로운 자료구조를 이용해야 하는 기존의 학습 알고리듬에서는 시도되지 않는 특징이다. 이에 대한 실험적 예시를 IV절의 모의실험에서 보인다.

(2) 학습의 일반화 및 망구조 최적화

각각의 문제를 해결하는 과정에서 학습된 망 구조는, 다른 알고리듬의 자료구조, 예를 들자면 ID3에서 생성되는 트리 구조등과 비교할 때, 크기 최적화와 일반화의 측면이 고려되지 않은 것이다. 따라서 개별적인 문제 각각에 대해서는 지나치게 복잡한 망 구조가 학습될 수도 있고, 일반화 성능이 떨어질 가능성도 있다.

일반화와 크기의 최적화에 대해서는 다음과 같은 점을 고려할 필요가 있다. 논리적 진화방식을 통해 학습되는 망 구조는 여러 가지 문제를 해결하는 데에 이용하는 특징노드를 모두 가지고 있다. 그리고 그들 중 일부는 두 가지 이상의 문제에 동시에 이용되기도 한다. 따라서 개별적인 문제가 순차적으로 주어질 경우, 개별적인 문제 모두에 대해 최적화된 크기를 가지도록 할 수는 없다. 그것은 개별적인 문제에 대한 학습 자료가 모두 동시에 주어져야만 가능한 것이다. 마찬가지로 일반화의 경우에 있어서도, 학습자료를 통해 얻어지는 일반화능력은 학습자료 내의 분포가 이후에 학습결과를 적용할 분포에 가깝다는 가정을 근거로 하고 있다. 그러나, 문제가 순차적으로 주어질 경우에는 이러한 분포를 미리 알고 학습할 수 없다. 모든 문제와 그것을 해결하기 위한 학습자료가 한꺼번에 주어지기 전에는 가능하지 않다. 이러한 두 가지 점을 고려하여, 논리적 진화방식을 개발하는 데 있어서는 일단 일반화와 크기의 최적화를 목표로 하지 않았다. 주어진 학습자료의 입출력관계를 만족시키는 함수를 학습하는 것만을 목표로 하였다.

다만, IV절의 실제적인 실험(MONK 문제)에서는 논리적 진화방식에 의한 학습 결과가 어느 정도 이상의 일반화 성능을 보이는 것을 확인할 수 있었다. 실제로 특징 노드는 고정성과 특수성을 통해 확률적인 정보를 축적해가게 되리라고 생각되는데 이에 대해서 차후에 다루어볼 필요가 있다.

(3) 학습의 완전성

제안된 알고리듬이 학습자료의 입출력관계를 만족하는 함수를 완전히 학습할 수 있다는 것을 장점으로 볼 수 있다. 이에 대해 증명하면 다음과 같다. 학습이 끝나고 나서, 망 구조가 학습자료 내의 입출력관계를 만족하지 않는다고 가정하면, 두 가지 종류의 학습 자료가 최소한 하나 이상 존재해야 한다. 첫째는, 원하는 출력이 T이고 망의 출력이 F인 자료이다. 둘째는, 원하는 출력이 F이고 망의 출력이 T인 자료이다. 그러나 이러한 자료가 존재한다면 첫째의 경우 생성 혹은 가시화가 동작하며, 원하는 출력이 T인 자료의 수를 하나 이상 증가시킨다. 따라서 학습이 끝났다면 (즉 아무런 연산도 더 이상 수행되지 않았다면) 이러한 자료는 존재할 수 없다. 학습이 끝났다면, 원하는 출력이 T인 모든 자료에 대한 망의 출력은 T가 된다. 둘째의 경우도 마찬가지로 이러한 자료가 존재한다면 특수화 또는 삭제가 동작하여, 원하는 출력이 F인 자료의 수를 하나 이상 감소시킨다. 따라서 학습이 끝났다면 이러한 자료는 존재할 수 없다. 결과적으로 학습이 끝난 뒤의 망 구조는 학습자료 내의 입출력관계를 만족하는 것이 된다.

IV. 모의 실험

이제까지 설명된 논리적 진화방식의 학습 성능을 평가하기 위해, 두 종류의 실험이 수행되었다. 첫째는 MONK 자료로, 기계학습(machine learning) 분야에서 알고리듬의 성능 평가를 위해 일반적으로 사용되는 것들 중 하나다. 이 자료를 학습시켜 개별적인 문제에 대한 학습 성능을 관찰한다. 둘째는 새로 정의한 자료집합이다. 여러 가지 문제를, 공통된 특징 노드를 이용하여 해결할 때, 특징노드의 재사용에 의해 학습 성능이

향상되는 것을 관찰한다.

1. MONK 자료

MONK 자료[12] 학습 알고리듬의 성능 비교평가(benchmarking)를 위해 정의되고 이용된다. 3가지 개념을 각각 학습하는 문제들로 구성되며, 각각의 문제는 이진 분류 문제로 다음과 같이 기술되는 개념을 학습하는 것이다.

#1 : (head_shape=body_shape) ∨ (jacket_color=RED)
#2 : exactly two of the six attributes have their first value.
#3 : (jacket_color=green ∧ holding=sword)
∨ (jacket_cloor=blue ∧ body_shape=octagon)
where value of head_shape∈{round, square, octagon}
value of body_shape∈{round, square, octagon}
value of is_smiling∈{yes, no}
value of jacket_color∈{red, yellow, green, blue}
value of holding∈{sword, balloon, flag}
value of has_tie∈{yes, no}

3가지 문제는 독특한 특징을 가지고 있다. #1은 논리합으로만 이루어진 간단한 개념이다. #2는 부호 문제(parity problem)이고, #3은 단순히 논리합과 논리곱으로 이루어져 있지만 잡음이 포함된 자료를 통해 학습하는 경우이다. 일단 주어지는 특징들이 가질 수 있는 값을 모두 고려하면 전체 가능한 경우는 432(=3×3×2×4×3×2)가지다. 전체 자료집합은 432개의 모든 경우를 예로 포함하고, 이 중 일부 자료(#1을 위해서 124개, #2를 위해서 169개, #3을 위해서 122개)가 선택되어 학습 자료로 사용된다. 그런데 논리적 진화방식에 사용가능한 입력은 이진 변수뿐이므로, 입력을 17개의 이진 변수로 변형하여 사용한다. 예를 들어 round, shape, octagon이라는 값을 가질 수 있는 holding이라는 변수는 holding_is_sword, holding_is_balloon, holding_is_flag이라는 3개의 이진 변수로 바꾸어 사용한다.

학습이 종료된 이후 망 구조에 학습된 개념을, 총 432개의 전체 자료를 이용하여 평가한다. 전체 예 중 학습된 개념에 의해 정확히 분류된 예들의 비율을 기준으로 사용하며 <표 2>에 주어져 있다. 다른 학습 알고리듬에 관련된 자료는[12]에서 인용되었다.

논리적 진화방식에 의해 학습된 망 구조의 분류 결과는 #1, #2에 대해서는 각각 100%로 좋은 성능을 보

표 2. MONK 자료 학습 결과
Table 2. Learning result on Monk data.

learning algorithm	#1	#2	#3
ID3	98.6	67.9	94.4
CN2	100.0	69.0	89.1
Back Propagation	100.0	100	93.1
Back Propagation (with weight decay)	100.0	100	97.2
AQ17-DCI	100.0	100	94.2
AQ17-HCI	100.0	93.1	100.0
AQ15-GA	100.0	86.8	100.0
Logical Evolution	100.0	100	98.3

인다. #3에 대해서는 98.3%로, 100%를 가지는 AQ17-HCI, AQ15-GA에 비해 약간 떨어지는 성능을 보인다. 그러나 AQ17-HCI와 AQ15-GA의 경우는 #2에 대해 각각 98.1%와 86.8%로 논리적 진화방식에 비해 좋지 않은 결과를 보이고 있다.

#3의 경우를 좀더 살펴보면, 잡음(noise)이 포함된 자료에 대한 논리적 진화방식의 독특한 대응 방식이 나타난다. #3의 학습 자료는 5%의 잡음(잘못된 출력을 준 경우)을 포함하고 이것은 전체 자료집합에서 1.4%에 해당한다. 따라서 자료의 타당성에 대한 내부적인 판단을 포함하지 않은 알고리듬에 의해 가능한 최고의 결과는 98.6%이다. 논리적 진화 방식에는 이러한 판단이 포함되어 있지 않고, 따라서 실제로 얻어진 98.3%라는 결과는 이러한 경우 얻어질 수 있는 최고 결과에 가깝다. 이 때, 이러한 결과는 잡음이 포함된 자료의 영향을 고립시킴으로서 가능해진다. 다른 자료에 의한 특징 노드와 논리적으로 맞지 않는 새로운 자료에 대해서는 그것을 아주 특별하고 예외적인 경우로 취급하도록 새로운 특징 노드를 만들어 고립시킨다. 이후 잘못된 자료의 영향은 이 특징 노드로 집중된다. 따라서 잘못된 예에서 얻어진 특징 노드는 올바른 예에서 얻어진 특징 노드에 대해 큰 영향을 주지 않게 된다.

2. 공통된 특징을 가지는 두 문제에 대한 학습

이 실험의 목적은 하나의 망 구조에 논리적 진화방식을 사용하여 두 가지 개념을 학습할 때, 이전에 생성된 특징의 재사용 가능성(reusability)과 그 효과를 관찰하는 것이다. 사용되는 두 가지 개념은 다음과 같이 간단히 정의되었다.

표 3. 필요한 기본연산 횟수
Table 3. Number of necessary operations.

	경우 1	경우 2
#1	125	125
#2	134	21
#1+ #2	259	146

$\#1 : (x_1 = T \wedge (x_3 = T \wedge x_5 = T)) \vee (x_2 = T \wedge x_4 = T)$

$\#2 : (x_1 = T \wedge (x_2 = T \wedge x_4 = T)) \vee (x_3 = T \wedge x_5 = T)$

두 가지 개념은 3개의 공통된 항 $x_1 = T$, $x_2 = T \wedge x_4 = T$, $x_3 = T \wedge x_5 = T$ 들로 이루어져 있다. 이러한 항은 개념을 학습하기 위한 특징에 해당된다. 따라서 두 개념 중 하나의 개념의 해결 과정에서 생성된 특징 노드를 다른 개념의 해결 과정에서 이용할 수 있다면, 특징 노드의 재생성에 필요한 부가적인 연산이 절약된다. 그러므로 이러한 특징이 재사용 가능한지의 여부는, 다음 두 가지 방식으로 학습할 때 필요한 연산의 수를 비교하여 확인할 수 있다. 첫 번째 경우는 각각의 개념에 대해 하나씩의 망, 즉 전체 두 개의 망을 사용하는 것이다. 두 번째 경우는 두 가지 개념을 공통된 특징노드를 포함하는 하나의 망을 이용하여 학습하는 것이다. 결과는 <표 3>에 주어져 있다. <표 3>에서 수치는 기본연산의 발생 횟수를 의미한다.

두 개의 망에 학습하는 경우 전체 연산횟수는 259번이고, 하나의 망에 학습하는 경우는 146번이다. #1을 풀 때 생성된 특징이 #2의 학습에 재 사용되어 #2의 학습에 필요한 연산 횟수가 1/6에 가깝게 감소하고, #1과 #2 모두를 푸는데 필요한 학습 횟수는 거의 반에 가깝게 감소하는 것을 확인할 수 있다. 이 결과는 생성된 특징의 재사용이 분명히 논리적 진화 망에서 일어나며, 이에 따라 필요한 연산이 크게 줄어드는 것을 보여준다.

이상의 결과에서, 논리적 진화방식에서는 특징의 재사용이 발생하며, 하나의 망에 여러 가지 문제를 해결할 때에 보다 효과적임을 관찰할 수 있다.

V. 결 론

이 논문에서는 새로운 방식의 귀납학습을 위한 논리적 진화방식을 제안하였다. 학습 과정에서 주어지는 자료에 따라, 망에 포함된 특징 노드들이 새로 생성되거나 변형되면서 학습이 이루어진다. 이 학습은 5가지의 기본 연산에 의해 이루어지며 목표로 하는 개념의 학습이 이루어진다. 학습 알고리듬에서는 각각 학습패턴에 대해 적용되는 논리적 평가방식으로 인해, 이전의 귀납학습 알고리듬이 가지는 일부 문제점들이 해결되었다. 첫째로, 문제에 포함된 정보를 학습 과정에서 추출하고 이용하므로, 보다 쉽게 적합한 특징을 생성하고 이용할 수 있다. 따라서 사전지식에 의해 특징을 미리 선택하는 경우의 영향이 제거되고 시행착오로 인한 잘못된 결과가 줄어든다. 둘째로, 특징의 생성, 선택과정에서 전체 자료집합에 강하게 의존하지 않는다. 따라서 새로운 문제가 주어지거나, 입출력이 변경되는 경우에도 이전의 학습 자원(즉, 특징 노드)을 바로 이용하여 빠르게 학습할 수 있다. 이 경우 이전의 특징노드는 이전의 학습 경험(experience)을 제공하여 학습을 돕는다. 따라서 기존의 귀납학습 알고리듬과 달리, 문제의 학습 결과를 축적하여 효율적으로 재 이용할 수 있다.

논리적 진화 방식은 아직 많은 추후 연구를 필요로 한다. 첫째로는 앞서 언급했던 다치 논리로의 확장, 실수 값 자료를 사용하기 위한 양자화가 연구되어야 한다. 그럼으로써 실제적인 많은 문제들에 적용이 가능해진다. 둘째로는 알고리듬 내부적인 문제다. 많은 문제가 학습됨에 따라 특징노드들의 특수성이 지나치게 커지는 경우가 발생할 수 잇다. 그러한 경우 현재 수준보다 일반화능력이 저하되고 해가 복잡해질 것이다. 이를 해결하기 위해 현재 단계의 알고리듬이 가지는 일반화능력을 분석하고 개선을 시도할 필요가 있다.

참 고 문 헌

[1] Mitchell, T., Machine Learning, MacGraw-Hill, 1997.

[2] Usama M. Fayyad, Gregory Piatesky-Shaprio, Padhraic Smyth and Ramasamy Uthurusamy, Advances in Knowledge Discovery and Data Mining, MIT Press, 1996.

[3] Koza, John R., Genetic Programming, MIT Press, 1992.

[4] Simon Haykin, Neural Networks, 2th ed, Prentice-Hall, 1999.

[5] Quinlan, J.R., "Discovering rules by induction from large collections of examples," in Expert systems in the micro electronic age, Edinburg

University Press, 1979.

[6] Clark, P. and Niblett, R., "The CN2 induction algorithm," Machine Learning, 3, pp. 261~284, 1989.

[7] Rumelhart, D.E., J.K. McClelland, eds., Parallel Distributed Processing : Exploration in the Microstructure of Cognition, vol.1, Cambridge, MA: MIT Press.

[8] Michalski, R.S., "A Review of Machine Learning Methods" in Machine Learning and Data Mining : Methods and Applications, John Wiley & Sons, 1996.

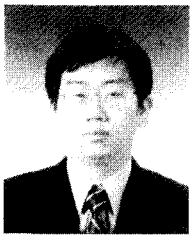
[9] Bleodorn, E. and Michalski, R.S., "Data-driven Constructive Induction in AQ17-DCI : A Method and Experiments," Reports of Machine Learning and Inference Laboratory, Center for Artificial Intelligence, George Mason University, 1991.

[10] Pagallo, G., and Haussler, D., "Boolean feature discovery in empirical learning," Machine Learning, 5., pp. 71~99, 1990.

[11] Yang, D-S., Rendell, L. A., and Blix, G., "A Scheme For Feature Construction and a Comparision of Empirical Learning Method," Proceedings of the 12th International Joint Conference on Artificial Intelligence, pp. 699~704, 1991.

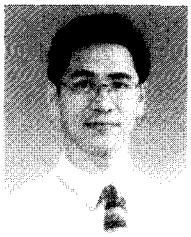
[12] Thrun, S.B. et al,, "The MONK's Problems : A Performance Comparison of Different Learning Algorithms," Technical Report CMU-CS-91:197, Carnegie Mellon University, Pittsburgh, 1991.

저 자 소 개



朴 明 銖(學生會員)

1998년 : 서울대학교 전기공학부 학사. 2000년 : 서울대학교 전기공학부 석사. 2000년~현재 : 서울대학교 전기공학부 박사과정 <주관심분야 : 신경회로망, 인공지능, 기계학습>



崔 鎭 榮(正會員)

1982년 : 서울대학교 제어계측공학과 학사. 1984년 : 서울대학교 제어계측공학과 석사. 1993년 : 서울대학교 제어계측공학과 박사, 1984~1994년 한국전자통신연구소 연구원, 1998년~1999년 : University of California, Riverside 객원교수. 1994년~현재 : 서울대학교 전기공학부 부교수 <주관심분야 : 적응제어, 신경회로망>