신경망을 이용한 클래스 간 메소드 위치 결정 메커니즘

정 영 애[†] · 박 용 범[††]

요 약

클래스의 속성과 메소드 참조관계를 고려한 응집도 측정 방법은 다양한 형태로 연구되어 왔다. 이러한 응집도의 측정 방법은 일반적으로 하나의 클래스에서 이루어진다. 단일 클래스에서 두 개의 클래스로 참조 범위를 확장하면 두 클래스 사이에 발생할 수 있는 참조관계를 알 수 있다. 본 논문에서는 메소드의 위치 결정을 위하여 신경망 학습모델을 제안하였다. 신경망은 입출력 패턴에 대한 반복학습 후, 학습 패턴에 포함되지 않았던 입력 패턴의 목적 값을 예측하고, 일반화(generalization)하는데 효과적이다. 두 개의 클래스 안의 속성과 메소드를 5개 이하로 제한하였고, 학습 벡터는 30개의 이진 값으로 구성된 입력 벡터와 메소드 위치 결정 값인 이진 목적 값으로 생성되었다. 제안된 신경망은 교차검증에서 약 95%, 테스트 데이터에 대해서는 약88%의 예측율을 보였다.

키워드 : 메소드 위치 결정, 응집도, 리팩토링, 신경망, 리질리언트 프로파게이션

A Mechanism to Determine Method Location among Classes using Neural Network

Young A. Jung[†] · Young B. Park[††]

ABSTRACT

There have been various cohesion measurements studied considering reference relation among attributes and methods in a class. Generally, these cohesion measurement are carried out in one class. If the range of reference relation considered are extended from one class to two classes, we could find out the reference relation between two classes. In this paper, we proposed a neural network to determine the method location. Neural network is effective to predict output value from input data not to be included in training and generalize after training input and output pattern repeatedly. Learning vector is generated with 30-dimensional input vector and one target binary values of method location in a constraint that there are two classes which have less than or equal to 5 attributes and methods. The result of the proposed neural network is about 95% in cross-validation and 88% in testing.

Key Words : Decision of Method Location, Cohesion, Refactoring, Neural Network, RPROP(Resilient Back Propagation)

1. 서 론

객체지향 개발방법은 분석, 설계, 프로그래밍, 테스팅, 유지보수에 이르는 소프트웨어 개발 전 과정에 걸쳐 동일한 방법론과 표현기법이 적용될 수 있다는 장점을 가진다[8].

또한 객체지향 개발방법을 기초로 하여 1980년대 초기부터 소프트웨어의 품질과 생산성, 프로그램의 복잡도 등을 측정하기 위한 방법들이 제안되었다. 독립적 모듈은 소프트웨어의 복잡도를 감소시켜 소프트웨어의 품질과 신뢰성을 향상시킬 수 있다. 신뢰성과 유지보수성이 좋은 소프트웨어를 생산하기 위하여 추상 데이터 타입에서의 응집도를 측정하기 위한 여러 연구들이 제안되었다[8]. 응집도란 모듈을 구성하는 요소들 간의 연관정도를 표현한 소프트웨어의 속성이며 소프트웨어의 응집도가 높을수록 소프트웨어의 가독성과 유지보수성이 높아진다[17]. 이러한 연구들은 클래스의 응집도를 측정하는데 주로 클래스 안의 인스턴스 변수들과 메소드 간의 상호작용을 측정하기 위하여 클래스 구성요소들의 연결성 정도를 측정한다.

객체지향 패러다임에서 객체는 속성(attribute)과 동작(behavior), 객체 사이의 관계(relationship)등으로 표현되는 클래스로 구현된다. 클래스는 클래스 간 속성과 메소드를 참조하는 관계를 가진다[3]. 일반적으로 능동적인 행위의 개념인 메소드는 다른 클래스의 속성이나 메소드를 참조하게 된다. 본 논문에서는 두 클래스 간의 참조관계를 가지는 메소드의 위치를 결정하기 위하여 리팩토링의 무브 메소드 요인을 정의한다. 정의된 세 가지 요인들에 의해 구성된 두 클래스 간의 참조 집합들의 속성을 가지는 실험데이터를 신

※ 이 연구는 2005학년도 단국대학교 대학연구비의 지원에 의해 수행되었음.
† 준 회 원 : 단국대학교 대학원 전자계산학과
†† 종신회원 : 단국대학교 전자컴퓨터학부 교수
논문접수 : 2006년 8월 9일, 심사완료 : 2006년 9월 11일

경망 학습 모델에 적용하여 학습시킨 후 테스트 하여 메소드 위치 예측을 위한 신경망 학습 모델의 응용을 제안한다.

이 연구에서 논의하고자 하는 사항은 다음과 같다. 첫째, 신경망 모델과 같은 기계 학습 알고리즘이 일반적으로 프로그래머의 직관적인 수동분석에 의해 이루어지는 리팩토링 분야에 적용될 때 발생될 수 있는 문제에 대하여 논하고자 한다. 둘째, 클래스의 응집도 측정에 사용되는 참조 집합으로 표현되는 무브 메소드 요인 세 가지에 대한 각 속성값을 신경망 학습 모델에 적용하는 것이다. 셋째, 임의의 데이터가 아닌 실제 리팩토링이 필요한 데이터에 대해 제안된 예측모델을 테스트하여 그 실용성을 파악하고자 한다.

논문의 구성은 2장에서 본 논문의 기초가 되는 리팩토링 기법과 응집도 척도, 신경망 등의 관련연구에 대해 살펴본다. 3장에서는 메소드 위치 결정 요인과 참조집합에 대한 정의과정을 논한다. 4장에서는 신경망 학습 모델에 적용할 데이터의 준비과정을, 5장에서는 제안된 방법을 적용한 실험 결과를 보인다. 마지막으로 6장에서 제안한 방법에 의한 예측에 있어 해결해야 하는 문제점을 지적하고, 앞으로 연구방향에 대하여 논한다.

2. 관련 연구

2.1 클래스 구성요소를 고려한 응집도 매트릭스

객체지향 시스템의 기본단위인 클래스는 객체의 특징을 속성(attribute)과 행위(behavior)로 표현한 단위이다. 클래스 구성요소들 간의 연관성을 나타낸 소프트웨어의 속성을 응집도[19]라 하며 응집도에 의해 표현되는 클래스의 독립성은 소프트웨어의 복잡도를 감소시키며 소프트웨어의 품질과 신뢰성 향상의 핵심으로 평가되어 왔다. 소프트웨어의 응집도가 높을수록 소프트웨어의 가독성이나 유지보수성이 향상된다. 이러한 응집도의 척도를 구하기 위한 다양한 연구는 모듈이나 컴포넌트까지 그 영역이 확대되었다[1].

클래스 안에 존재하는 메소드의 수를 중심으로 한 매트릭스로는 WMC(Weighted Methods per Class)와 RFC(Response For a Class)가 있다. WMC는 클래스당 가중 메소드의 수를 나타낸 것으로 클래스에 정의된 모든 메소드에 대한 복잡도의 합으로 표현된다. 클래스에 선언된 메소드가 많을수록 이 값은 증가하며, 클래스의 개발이나 유지보수에 많은 노력이 필요함을 의미한다. CBO(Coupling Between Object classes)는 상속성이 없는 클래스 간의 결합도를 나타낸다. 하나의 클래스가 다른 클래스의 속성이나 메소드를 사용하여 결합한 수를 측정한다[5]. 이 값이 클수록 복잡도가 증가하며 모듈의 독립성 저하로 인하여 재사용과 변경이 어려워진다.

RFC는 클래스가 호출하는 메소드의 수로 클래스에 대한 반응도를 나타낸다. 호출되는 메소드가 많아질수록 객체의 복잡도가 증가하여 유지보수에 더 많은 노력이 필요하게 된다[5].

클래스 내부의 메소드 응집 결여도를 나타내는 LCOM(Lack of Cohesion in Methods)은 같은 메소드에 의해 공용되는 인스턴스의 수와 공용되지 않는 인스턴스의 수에 대한 차이다[21]. 메소드들이 같은 인스턴스를 많이 사용할수록 클래스의 응집도가 향상된다. 이와 유사한 개념으로 TCC(Tight Class Cohesion)와LCC(Loose Class Cohesion)가 있다[12]. 하나 이상의 공통 인스턴스 변수를 사용하는 두 메소드를 직접 연결되었다고 하고, 다른 직접 연결된 메소드들에 의해 연결된 메소드는 간접 연결되었다고 한다. NP(C)는 객체(클래스) 내의 직접 또는 간접적으로 연결될 수 있는 최대 경우의 수이고, NDC(C)는 직접 연결수를 NIC(C)는 간접 연결수를 의미한다. TCC는 NP(C)에 대한 NDC(C)의 비율로 LCC는 NP(C)에 대한 NDC(C)와 NIC(C)의 합에 대한 비율로 나타내며 그 값이 클수록 응집도가 높아진다. 그러나 같은 값을 가지는 경우에 메소드와 인스턴스 변수들 간의 연결 패턴의 차이를 구분하지 못한다는 단점이 있다[2].

CBO(Coupling Between Object classes)는 상속성이 없는 클래스 간의 결합도를 나타낸다. 하나의 클래스가 다른 클래스의 속성이나 메소드를 사용하여 결합한 수를 측정한다. 이 값이 클수록 복잡도가 증가하며 모듈의 독립성 저하로 인하여 재사용과 변경이 어려워진다.

2.2 리팩토링과 무브 메소드

리팩토링은 프로그램의 행위(behavior)를 보전하면서 프로그램의 구조 및 성능을 향상시키는 좋은 방법이다[2]. 프로그램 개발자는 리팩토링을 사용하여 프로그램을 이해하기 쉽고 변화를 포용할 수 있도록 프로그램을 단계적으로 개선할 수 있다. 리팩토링은 프로그램의 가독성을 높일 뿐만 아니라 개발 속도를 높일 수 있도록 효율적이고 통제된 방법을 제공하여 소프트웨어의 가치를 높인다는 장점을 가진다[2].

리팩토링은 1990년 초의 개발을 시작[22, 23, 24]으로 하여 그 후에도 소프트웨어 개발의 주된 분야로 지속적인 연구가 있었다[17]. 초기의 리팩토링은 대부분 수동분석에 의존하였으나, 후에 자동화하려는 시도는 매우 다양했다. 리팩토링 자동화는 리팩토링 후보영역을 정의하는 분야[15, 20, 25]와 개발자가 정의한 후보영역에 리팩토링을 자동으로 적용하는 분야[13, 14]로 양분되어 있다.

리팩토링의 후보 영역을 정의 하는 분야의 연구로는 특정 리팩토링 기법에 대한 불변점 패턴 매칭(invariants pattern matching)을 사용하여 리팩토링이 필요한 부분을 찾아 나가는 방법[15, 25], 요구사항 변경에 따른 자바 프로그램의 후보영역(candidate spot)을 원하는 디자인 패턴 구조로 자동 변환하기 위하여 추론(inference)규칙을 사용하는 방법이 제안되었다[20].

개발자가 정의한 후보영역에 자동으로 리팩토링을 적용하는 분야에서는, 가중치 종속 그래프(weighted dependence graph)를 사용하여 객체지향 프레임 워크에서 메소드를 자동으로 리팩토링하는 방법[13]과 프로그램 슬라이싱(Program slicing)을 사용한 방법[14]등이 있다. 더불어 응집도 매트릭스에 의해 측정된 거리를 시각화(visualization)하는 툴을 개발하여 리팩토링을 돕는 연구도 제안되어 있다[10].

2.3 신경망

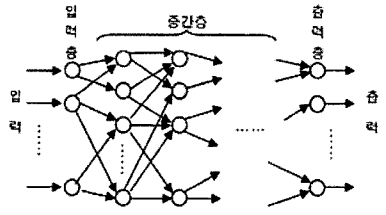
뇌의 학습 기능에 대한 메커니즘은 완전히 밝혀지지는

않았지만, 생리학뿐만 아니라 공학 분야에서도 이 학습 원리에 대한 많은 연구가 시도되었다. 1960년에 로잔브로트에 의해 제안된 퍼셉트론으로 시작되었으며 민스키(Minsky)와 페퍼트(Papert)에 의해 퍼셉트론의 학습식별 능력의 한계가 밝혀졌다. 이로 인하여 관련연구에 대한 침체기를 거쳤다[4]. 그러나 1986년 룸멜하트 등에 의해 백프로파게이션(Back propagation) 학습 알고리즘이 제안되어 다시 신경망에 대한 관심이 높아져 학습 알고리즘의 성능 향상을 위한 연구와 응용이 활발하게 이루어지고 있다[7, 18].

신경망의 원리는 여러 계층으로 결합한 계층형의 네트워크에서 어떤 형태의 계산처리를 받은 신호가 다음 층에 전해지며 이 신호는 또 다른 층에 차례로 전해져서 마지막 층에서 어떤 값으로 출력된다. 이때 각 뉴런 간에 가변의 결합강도가 있고, 결합강도를 적절히 변경함으로써 네트워크의 여러 가지의 입출력관계를 실현할 수 있다. (그림 1)은 다층 구조의 신경망 네트워크 구조이다.

네트워크에 목표 입출력 데이터를 반복해서 네트워크에 제시하고 네트워크는 데이터가 제시될 때마다 결합강도를 조금씩 조절해 가며 될 수 있는 대로 그 입출력관계를 학습한다. 다층 신경망에서 학습은 출력 뉴런에 대하여만 행해지며 입력 층과 중간 층 사이의 결합은 랜덤하게 정해지지만 여기서의 결합도 학습에 의해 효율적인 결합관계를 나타낼 수 있도록 한 것이 백프로파게이션 학습 알고리즘이다.

신경망은 입출력 패턴 대한 반복적인 학습 수행 한 후 학습 패턴에 포함되지 않았던 입력 패턴의 목적 값을 예측하고, 일반화(generalization)하는데 효과적인 학습 알고리즘[4]이다. 실제 수동 분석에 의해 결정된 메소드의 위치와 그 결정을 하기 위해 고려된 참조 관계 데이터를 이용하여 신경망에 대한 학습을 수행한 후 실제로 고려되지 않았던 환경의 적용요인 값을 학습모델에 적용하여 대상 메소드의 위치를 예측하고자 한다. 본 논문에서는 여러 신경망 중 역전파 알고리즘의 속도를 개선한 RPROP(Resilient Back Propagation) 학습 알고리즘을 선택하였다[18].



(그림 1) 계층형 네트워크

3. 메소드 위치 결정 요인 정의

3.1 메소드 위치결정 요인을 위한 용어 정의

두 클래스 간에 참조관계를 가지는 메소드의 위치를 결정하기 위한 요인을 정의하기 위한 용어들은 다음과 같다.

- 소스 클래스(Source Class) : 무브 메소드를 적용할 메소드가 포함되어 있는 클래스
- 타겟 클래스(Target Class) : 무브 메소드를 적용할 메소드가 이동하게 될 클래스
- 속성(attribute) : 무브 메소드를 적용할 메소드가 참조하는 소스클래스의 필드 (타겟 클래스의 필드를 참조하기 위하여 get/set 메소드를 사용하는 것은 타깃 클래스의 필드를 참조하는 것으로 간주함)
- 대상 메소드 : 무브 메소드 기법을 적용할 메소드
- 일반 메소드(general method) : 타겟 클래스의 필드를 참조하기 위한 get/set 메소드를 제외한 모든 메소드

3.2 무브 메소드에 기초한 메소드 위치 결정 요인

클래스 간의 참조관계가 형성된 경우, 무브 메소드의 적용여부에 대한 확신을 가지는 것은 쉽지 않다. 파울러는 무브 메소드 적용요인을 "메소드가 자신이 정의된 클래스보다 다른 클래스의 기능을 더 많이 사용하고 있는 경우" 정의하고 있다[8]. 메소드 위치 결정 요인은 무브 메소드에 대한 파울러의 기본정의에 기초하였으며, 수동 분석시 보편적으로 적용되는 상황들을 분석하여 반영하였다.

보편적으로 적용되는 상황으로는 대상 메소드가 타겟 클래스의 필드를 소스클래스의 필드보다 많이 사용하거나, 공동으로 일하는 부분이 너무 많아서 단단히 결합되어 있는 경우 등이 있다. 이런 경우 무브 메소드를 적용함으로써 클래스의 역할을 좀 더 명확하게 구분할 수 있게 된다.

메소드의 위치를 결정하기 위한 세 가지 요인[6]은 다음과 같고, 각 요인별로 소스 클래스의 속성이나 메소드를 참조하는 경우인 직접 참조와 타겟 클래스의 속성이나 메소드를 참조하는 경우인 간접 참조 관계를 가지게 된다. 그러므로 참조관계는 각 요인 별로 두 개씩의 참조관계가 성립할 수 있어 총 6가지의 참조관계로 표현된다.

다음은 메소드 위치를 결정하기 위한 세 가지 요인이다.

- 메소드 위치 결정 요인 1 : 대상 메소드가 참조하는 소스클래스의 속성의 개수와 타겟 클래스의 필드 개수의 차이를 비교한다. 일반적으로 다른 클래스의 속성에 접근하기 위해서는 간접접근자인 get/set 메소드를 통하게 되므로 간접 접근자를 통한 타겟 클래스의 속성에 대한 참조도 간접적인 필드참조로 간주한다. 따라서 결정 요인1은 대상 메소드의 속성에 대한 직접 또는 간접적인 참조관계이다.
- 메소드 위치 결정 요인 2 : 대상 메소드의 일반 메소드에 대한 직접 또는 간접적인 메소드 참조횟수의 차를 비교한다. 대상 메소드에서 소스클래스의 메소드보다 타겟 클래스의 메소드에 대한 호출이 많으면 메소드를 타겟 클래스로 옮겨야 한다.
- 메소드 위치 결정 요인 3 : 대상 메소드가 일반 메소드에 의해 직접 또는 간접적으로 참조되는 횟수의 차를 비교한다. 타겟 클래스에 의해 호출되는 횟수가 많은 경우 타겟 클래스와 강하게 결합하여 있으므로 무브 메소드를 수행해야 한다.

3.3 메소드 위치결정 요인을 위한 참조 집합

메소드 위치 결정을 위한 참조 집합은 응집도 매트릭스를

위해 제안된 참조 그래프[9, 11]를 기반으로 하여 정의되었다. 참조 집합은 두 클래스에 속한 속성과 메소드들의 참조 관계를 잘 표현할 수 있는 방법으로 두 클래스의 관계를 시각화하는데 사용되었다.

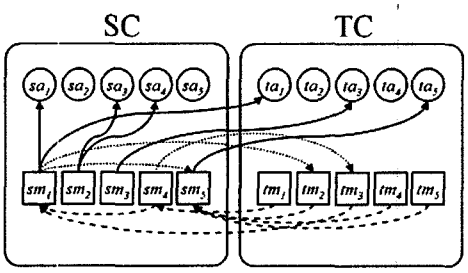
참조그래프는 소스클래스와 타겟 클래스에 속한 속성과 메소드들은 노드로, 참조관계를 나타나는 방향성 있는 에지로 구성된다.

• 참조 그래프의 정의 :

$SC=\{sa_1, sa_2, \cdots, sa_i, sm_1, sm_2, \cdots, sm_j\}$가 소스 클래스에서 정의된 데이터 요소와 메소드들의 집합이고, $TC=\{ta_1, ta_2, \cdots, ta_i, tm_1, tm_2, \cdots, tm_j\}$는 타겟 클래스에서 정의된 데이터 요소와 메소드들의 집합이라 하자. sm(sm∈SC)이 sa(sm∈SC)을 사용하면 sm이 sa를 참조한다고 하고, sm → sa 로 나타낸다. 메소드 참조 그래프 G(V, E)는 방향성을 가지는 그래프이며, V={SC∪TC}이고, $E=\{(sa,sm,ta,tm) \mid sm \xrightarrow{ad} sa, sm \xrightarrow{ai} ta, sm \xrightarrow{md} sm', sm \xrightarrow{mi} tm, sm \xleftarrow{rd} sm', sm \xleftarrow{ri} tm\}$이다. 여기에서, sm과 sm'은 SC에 존재하는 서로 다른 메소드이다.

위치 결정의 대상이 되는 메소드가 SC의 속성이나 메소드와의 참조관계는 ad, md, rd이며, 여기에서 d는 직접(direct)의 의미이며, TC의 구성요소와의 참조관계는 ai, mi, ri로 나타내며 여기에서 i는 간접(indirect)를 의미한다.

(그림 2)는 여섯 가지의 참조관계를 가지는 경우이며, 속성에 관한 참조관계인 ad, ai는 실선으로, 메소드를 참조하는 경우는 둥근 점선으로, 다른 메소드에 의해 참조되는 경우인 rd, ri는 파선으로 표현한다.

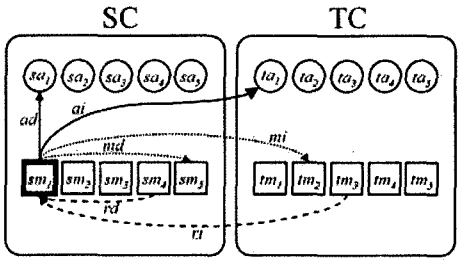


(그림 2) 참조 그래프의 예

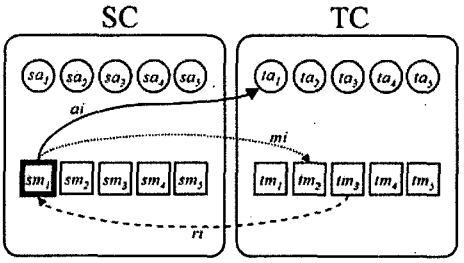
• 메소드 위치 결정 참조 집합의 정의 :

무브 메소드를 적용할 대상 메소드인 sm(sm∈SC)이 가지는 참조 관계를 나타내는 "*ad, md, ai, mi, rd, ri*"들의 집합으로 정의되는 메소드 위치 결정 참조 집합은 $Ref_{sm} = \{ad_1, \cdots, ad_i, ai_1, \cdots, ai_j, md_1, \cdots, md_k, mi_1, \cdots m_{il}, rd_1, \cdots, rd_m, ri_1, \cdots, r_{in}\}$ 이며, 여기에서 *i, j, k, l, m, n*은 sm이 각 속성들과의 여섯 가지 각각의 참조관계에 대한 개수이다.

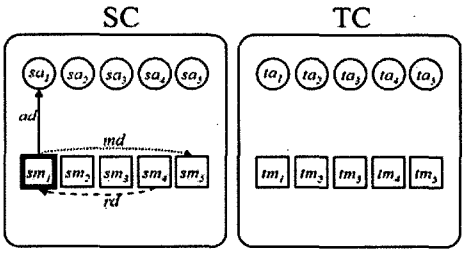
(그림 3)은 굵은 실선으로 표현된 대상 메소드 sm_l대하여 SC와 TC참조관계를 하나씩 가지는 경우에 나타나는 메소드 결정 참조 집합의 예이다. (그림 4)는 sm_l이 가질 수 있는 간접적인 참조 관계만을 가지는 경우에 나타나는 경우이고, (그림 5)는 직접적인 참조 관계만을 가지는 경우이다. 여기에서, (그림 4)와 같은 경우는 간접적인 경우만 있다 하더라도 두 클래스 간의 참조 관계가 존재한다. 그렇지만 (그림 5)의 경우에는 직접적인 참조만 있기 때문에 SC와 TC 사이의 참조 관계는 없지만, 본 논문에서는 간접적인 참조 관계가 없는 경우라고 하더라도 직접적인 참조관계가 존재하면 본 논문에서 제안한 신경망의 입력 데이터로 사용되었다.



(그림 3) 참조 집합의 예1



(그림 4) 참조 집합의 예2



(그림 5) 참조 집합의 예3

4. 신경망 적용을 위한 데이터 준비

앞서 정의된 메소드의 위치를 결정짓는 적용요인들의 검증을 위해 전처리 과정을 통해 준비된 데이터에 대해 시뮬레이션을 수행하였다. 데이터는 적용요인1~적용요인3이 가지는 두 가지씩의 경우에 해당되는 여섯 가지의 입력 데이터와 하나의 디지트(digit)로 표현되는 목적 데이터로 구성된다. 여섯 가지의 입력 데이터들의 속성 값의 범위는 실제 상황에서 다양할 수 있겠지만 본 논문에서는 0~5까지의 속성 값을 가지는 경우로 제한하였다.

학습 데이터는 각 입력 데이터의 속성 값을 0을 포함하는 자연수로 대치하고 digit-to-binary 벡터를 생성시켰다. 하나의 적용요인 속성 값은 다섯 개의 비트로 코드화 되었으며, 참조관계가 전혀 없는 경우는 0으로 코드화한다.

입력 데이터와 목적 값이 같은 패턴이 여러 번 나타난 경우는 학습에 중복적용이 되어도 학습효과에는 미치는 영향은 한 번을 적용하는 것과 다르지 않으므로, 이러한 데이터의 사용은 한번으로 제한하였다.

학습에 사용한 30×1의 구조를 가지는 입력 데이터의 수와 메소드의 위치 값은 출력 데이터의 수는 280개이다. 여기에서 입력 데이터는 하나의 메소드가 가질 수 있는 참조 관계의 수이며, 출력 데이터는 리팩토링 기법을 이용한 수동 분석 결과 값이다. 테스트 데이터의 수는 전체 데이터의 20%, 30%, 40%에 해당되는 데이터들을 사용하였다. 20%의 경우는 5가지, 30%의 경우는 3가지, 40%의 경우는 2가지의 경우로 구분하였다. 이때, 테스트 데이터에 해당되는 데이터들은 무작위 추출하였고, 각 경우의 테스트 데이터들의 중복은 없도록 하였다. 또한, 준비된 전체 데이터의 80%는 10-원 교차검증에 사용되었고, 나머지 20%는 테스트를 위하여 사용되었다.

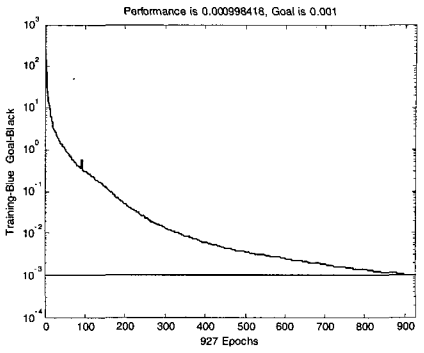
5. 실험 및 결과

학습할 데이터의 전처리 과정을 통해 준비된 학습용 데이터를 신경망에 적용시켰다. 학습에 이용된 알고리즘은 역전파 알고리즘의 학습속도를 개선한 RPROP(Resilient back PROPagation)이다. 학습율은 0.01, 최대 학습 반복수 10,000, 최소 학습 율의 변화는 0.000001, 최대 가중치 변화는 50번, 가중치 변화 증가비율은 1.2, 가중치 변화 감소비율은 0.5이다. 학습에 사용된 시스템은 IBM PC Pentium IV 3.0GHz이다.

신경망 학습 구조에서 중간층의 뉴런 수는 실험을 통해 선택하였다. 신경망의 중간층 뉴런 수를 결정하기 위하여 중간층 뉴런의 수를 12개~24개까지 변화시켜 학습시켰고 각 학습의 경우에 대하여 학습율을 0.01~0.05까지 변화시켜 학습하였다. 그 결과, 중간층의 뉴런 수가 12개이면서 학습율이 0.01 또는 0.05인 경우, 뉴런 수가 14개이면서 학습율이 0.05인 경우, 뉴런 수가 16개이면서 학습율이 0.03인 경우가 학습효과가 다른 경우에 비해 비교적 높았다. 실험에서 뉴런 수가 12개이면서 학습율이 0.01인 경우에 가장 좋았기 때문에, 신경망 구조의 중간층 뉴런 수는 12개로, 학습율은 0.01로 채택하였다. (그림 6)은 뉴런 수를 12개, 학습율을 0.01로 조절했을 경우의 그래프이다. 일반적으로 학습 결과에 따른 그래프는 학습의 일반화를 측정할 수 있는 근거가 된다.

중간 층 뉴런의 수가 12개이고 학습율이 0.01인 학습 모델로 준비된 데이터로부터 20%의 테스트 데이터에 대한 실험을 하였다. 학습 데이터 대비 테스트 데이터의 비율이 80% : 20%의 비율로는 5가지 경우로 실험을 한 결과는 95% 이상의 예측율을 보였다.

<표 1>은 신경망 구조를 31×12×1로, 학습율은 0.01로 선택한 경우에 10-원 교차 검증의 실험 결과를 보여주고 있다. 학습된 신경망에서 학습의 경우에는 100%, 교차검증에서는 95.18±2.52%의 학습 예측율, 그리고 테스트 데이터에 대해서는 88.46±3.65%의 예측율을 보였다.



(그림 6) neuron : 12개, 학습율: 0.01인 경우의 학습 그래프

〈표 1〉 학습 및 교차검증예측결과(임계값: 0.9)

case	구 분	학습 예측율
1	학 습	100.00±0.00
	교차검증	98.22±2.18
	테 스 트	92.50±4.50
2	학 습	100.00±0.00
	교차검증	95.51±3.52
	테 스 트	86.96±6.92
3	학 습	100.00±0.00
	교차검증	95.18±7.64
	테 스 트	92.14±8.23
4	학 습	100.00±0.00
	교차검증	94.66±5.51
	테 스 트	88.21±6.40
5	학 습	100.00±0.00
	교차검증	92.31±11.87
	테 스 트	82.50±6.38
평 균	학 습	100.00±0.00
	교차검증	95.18±2.52
	테 스 트	88.46±3.65

6. 결론 및 향후 과제

본 논문에서는 신경망을 객체지향 프로그램의 메소드의 위치를 결정하는데 적용하는 방법론을 제안하였다. 제안하고 있는 신경망은 메소드의 위치 결정에 좋은 예측결과를 보이는 학습 모델임을 실험을 통하여 알 수 있었다.

신경망과 같은 학습 모델의 응용은 기존 데이터를 통해 예측결과를 예측하는 모델링이 가능하다는 장점을 가진다. 소프트웨어 공학적 측면에서 소프트웨어의 안정성을 높이기 위한 메소드의 위치결정뿐만 아니라 소프트웨어의 안정성에 영향을 미치는 요인들을 정의하고 제안된 모델에 적용하여 메소드뿐만 아니라, 속성의 위치 결정까지도 예측하는 실험도 필요하다. 또한, 소프트웨어 개발자들에 의해 수행되는 수동적 분석을 통하여 수정을 하는 것보다 여러 학습 모델의 예측결과를 확보하여 그 결과를 이용하는 것이 보다 신뢰할 수 있을 것으로 기대된다. 앞으로 소프트웨어의 안정성과 관련된 여러 기법들에서 다양한 요인을 추출하고, 그에 따른 다양한 데이터를 준비하여 복합적인 요인들에 의해 소프트웨어의 안정성을 보장할 수 있는 연구가 필요하다.

참 고 문 헌

[1] 고병선, 박재년, 컴포넌트의 응집성 측정, 정보처리학회 논문지 제9-D권 제 4호, 613-618

[2] 박지훈, 자바 디자인 패턴과 리팩토링, 한빛미디어㈜, 2003

[3] 윤청, 성공적인 소프트웨어 개발 방법론, 생능출판사, 1996

[4] 김응수, 뇌의 정보 시스템, 생능출판사

[5] 김성애, 최완규, 이성주, 객체지향 패러다임에서 저해요인에 기반한 응집도 척도, 한국정보처리학회 논문지 제7권 제11호, 2000

[6] 정영애, 박용범, 로지스틱 분석을 이용한 메소드 위치 결정 방법, 한국정보처리학회 논문지 제 12-D권 제 7호, pp.1017-1022

[7] Aristoklis D. Anastasiadis, George D. Magoulas, Michael N. Vrahatis, "New globally convergent training sckeme based on the resilient propagation algorithm," Nerocomputing 64, 2005, pp.253-270

[8] Bindu, Mehra, "A Critique of Cohesion Measures in the Object-Oriented paradigm," Masters Thesis, Department of Computer Science, Michigan Technological university, 1997

[9] Embley, d. W. and woodfield, S. N., "Accessing the quality of abstract data types written in Ada," Proc of Pheonix Conf. on Computers & Comm., pp.205-213, 1987.

[10] F. Simon, F. Steinbrkner, and C. Lewerentz, "Metrics based refactoring", in Proc. European Conf. Software Maintenance and Reengineering IEEE, pp.30-38,Computer Society, 2001.

[11] HeungSeok Chae, YongRae Kwon, "A Cohesion Measure for classes in Object-Oriented Systems," Proceeding of the 5th International symposium of Software Metrics, 1998.

[12] James M. Bieman, Byung-Kyoo Kang, "Cohesion and reuse in an object-oriented paradigm," Proc. ACM Symposium on Software Reusability (SSR-95), pp.259-262, 1995.

[13] Katsuhisa Maruyama, Kenichi Shima, "Automatic Method refactoring using weighted dependence graphs," Proceedings of the 21st international confer-ence on Software engineering, 1999

[14] Katsuhisa Maruyama, "Automated Method extraction refactoring by using block-based slicing", Proceedings of the 2001 symposium on Software reusability: putting software reuse in context, Vol 26, pp.236-245, 2001.

[15] Kuyul Noh, Changki Kim, Jonggul Park and Jaeha Song, "The Evaluation of Daikon: utilization of Daikon in the POI Data Inspection System", 4WD Team Master of Software Engineering Program School of Computer Science, Carnegie Mellon University, 2002.

[16] L. M. Otto et al., "Developing Measures of Class Cohesion for Object-Oriented Software.," 7th Annual Oregon Workshop on Software Metrics, 1995.

[17] Martin Fowler, etc, 'Refactoring Improving the Design of Existing Code', Addison Wesley, 1999.

[18] Martin Riedmiller, Heinrich Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," proceedings of the International Conference on Neural Networks, San Francisco, 1993, pp.586-591.

[19] Samadzadeh, M. H. and Khan, S. J., "Stability Coupling and Cohesion of Object-Oriented Software Systems," PROC, 22nd Ann. ACM Computer Science conf., pp.312-319, 1994.

[20] Sang-Uk Jeon, Joon-Sang Lee, and Doo-Hwan Bae, "An Automated Refactoring Approach To Design Pattern-Based Program Transformations in Java Programs", In Proceedings of 9th Asia-Pacific Software Engineering Conference, Gold Coast in Austrailia, 2002.

[21] Shyam R. Chidamber, Chris f. kemerer, "Towards a Metrics Suite for Object-Oriented Design," In Proc. OOPSLA, '91, ACM, pp.197-211, 1991.

[22] William G. Griswold, 'Program Restructuring as an Aid to Software Maintenance PhD Thesis', Dept. of Computer Science & Engineering, University of Washington, 1991.

[23] William F. Opdyke and Ralph E.Johnson, Refactoring: An aid in designing application Frameworks and evolving object-oriented systems, In Proceedings of SOOPPA '90: Symposium on Object-Oriented Programming Emphasizing Practical Applications, Sep., 1990.

[24] W.F. Opdyke, 'Refactoring object-oriented frameworks Ph.D.thesis', Computer Sciences Department, University of Illinois at Urbana-Champaign, 1992.

[25] Yoshio Kataoka, Michael D. Ernst, William G. Griswold, David Notkin, "Automated support for program refactoring using Invariants", Proceedings of the IEEE International Conference on Software Maintenance(ICSM'01), pp.736-743, 2001.

정 영 애

e-mail : yajung@dankook.ac.kr

1995년 호서대학교 전자계산학과(학사)

2000년 호서대학교 전자계산학과(이학석사)

2003년 단국대학교 전자계산학과 박사 수료

1994년~1998년 미래 산업(주) 소프트웨어 개발팀 연구원

관심분야 : 소프트웨어 품질평가, Information Architecture, 패턴인식, 멀티미디어 콘텐츠

박 용 범

e-mail : ybpark@dankook.ac.kr

1985년 서강대학교 전자계산학과(학사)

1987년 N.Y. Polytechnic Univ. 전자계산학과(석사)

1991년 N.Y. Polytechnic Univ. 전자계산학과(박사)

1993년~현재 단국대학교 전자컴퓨터학부 컴퓨터과학전공 교수

관심분야 : Information Architecture, 패턴인식, 분산 에이전트