비정규화 데이타를 이용한 신경망 소프트웨어 신뢰성 예측

이 상 운†

요 약

소프트웨어 신뢰성을 예측할 때, 소프트웨어 시험 중에 시험 종료시간과 소프트웨어에 얼마나 많은 결함이 잔존하는지(데이터의 최대 값)를 판단할 수 없으므로 데이타의 최대 값을 추정하여 신경망을 훈련시키고 얻은 신뢰성 예측치는 부정확해질 수 있다. 본 논문에서는 신뢰성 예측에 있어서 신경망에 정규화하지 않은 데이타를 사용하는 방법을 제시한다. 이 방법은 신경망 훈련시 데이타의 최대 값을 추정할 필요가 없어 정규화 없이 신경망을 사용 가능하며, 신뢰성 예측 결과도 추정된 최대 값과 실제 획득될 최대 값과의 차이(추정 오차)만큼 정확해질 수 있다. 또한 비정규화된 데이타를 사용하는 방법이 데이터의 최대 값을 알고 있다고 가정한 상태의 정규화된 방법보다 예측 정확성이 좋음을 보였다. 따라서, 이 모델을 비정규화된 데이타를 이용하여 신뢰성 예측에 유용하게 사용할 수 있다.

Neural Network for Software Reliability Prediction with Unnormalized Data

Sang-Un Lee†

ABSTRACT

When we predict of software reliability, we can't know the testing stopping time and how many faults be residues in software (the maximum value of data) during the software testing process, therefore we assume the maximum value and the training result can be maccuracy In this paper, we present neural network approach for software reliability prediction with unnormalized (actual or original collected) data. This approach is not consider the maximum value of data and possible use the network without normalizing but the predictive accuracy is better Also, the unnormalized method shows better predictive accuracy than the normalized method given by maximum value Therefore, we can make the best use of this model in software reliability prediction using unnormalized data

1. 서 론

일반적으로 소프트웨어의 품질을 정량적으로 표현하는 속성중의 하나가 소프트웨어 신뢰성이다[11, 14]. 소프트웨어 신뢰성은 주어진 환경하에서 주어진 시간동안 소프트웨어가 고장없이 작동할 확률로 정의된다[4, 11] 소프트웨어 신뢰성을 평가하기 위해 고장 발생 현상이 주어진 분포를 따른다는 가정하에 유도된 수학적 모델에 관찰된 고장 데이타를 이용해 추정된 모수를 적용한 통계적 모델이 일반적으로 사용되고 있다. 주어진 소프트웨어에 대해 가장 적합한 통계적 신뢰성 평가 모델 선택 문제는 소프트웨어 인도 일자나 자원을 할당하는 의사결정 문제가 모델의 예측 결과에 크게 의존하기 때문에 소프트웨어 신뢰성 추정 및 예측에 있어서 매우 중요한 요소이다. 그러나 통계적 모델 즉, 소프트웨어 신뢰성 성장 모델(SRGMs : software re-

† 정 회 원 : 경성대학교 대학원 전자계산학과
논문접수 : 1999년 10월 18일, 심사완료 : 2000년 5월 8일

liability growth models)이 많이 제안되었지만 모든 소프트웨어의 다양한 환경에 적합한 모델이 아직 연구되지 못한 실정이다[6, 14].

최근들어 모델의 모수 추정과 모델 자체 학습으로 미래의 결과를 예측하는데 신경망(NNs : Neural Networks)이 적용되고 있다[2, 6, 15]. NNs(또는 Artificial Neural Networks, Neural Nets, Connectionist)은 현재까지 이해된 단순한 뇌 신경계가 어떻게 작동을 하는가를 이상적으로 표현한 수학적 모델로 단지 입력된 고장 데이타에 적합하도록 학습을 통해 자율적으로 망의 내부 모델을 개발하고 미래의 고장 발생 과정에 대한 예측을 할 수 있다. 즉, 고장 발생 이력의 복잡성에 적합하도록 모델의 복잡성을 자율적으로 조절하기 때문에 널리 사용되고 있는 통계적 모델보다 정확한 결과를 얻을 수 있다[5, 8-10, 14].

소프트웨어 신뢰성 예측 관련 대표적인 신경망 연구로는 Karunanithi et al.[8, 9]과 Park et al.[13]이 있다. 대부분의 신경망 연구에는 주어진 신경망의 입·출력 뉴런인 시그모이덜(Sigmoidal) 뉴런의 활성화 함수(Logistic 또는 Tanh 함수)가 취할 수 있는 범위에 일치하도록 수집된 데이타를 [0, 1] 또는 [-1, 1] 사이의 값으로 한정시킨 정규화(Normalized)된 데이타로 변환시켜 사용하고 있다. 소프트웨어 시험의 경우, 시험 종료시 수집된 데이타(발견된 고장 수)의 최대 값을 반드시 알고 있어야만 정규화-역정규화시 데이타의 정확성에 대한 손실 없이 변환이 가능하다. 최대 값을 모를 경우 단지 추정된 값을 사용해야하며, 모델의 예측 정확성에 나쁜 결과를 초래할 수 있다. 본 논문은 이러한 문제점을 해결하고자 실제적으로 관찰 또는 수집된 데이타를 정규화 하지 않은 원 데이타(Original Data)를 사용한 신경망을 활용할 수 있는 소프트웨어 신뢰성 예측 모델을 제안하고자 한다.

2장에서는 관련 연구 및 문제점을, 3장에서는 데이타 정규화시 발생되는 문제를 해결할 수 있는 신경망을 제안하며, 4장에서는 정규화한 데이타와 정규화하지 않은 원 데이타를 적용시 모델의 예측결과를 상호 비교함으로서 제안된 모델의 적합성을 살펴보고자 한다.

2. 관련 연구 및 문제점

신경망의 중요한 특징중 한가지는 환경 변화에 따라 자체적인 학습능력을 갖고 있으며, 어떤 점에서는 성능을 향상시키기 위해 학습을 수행한다는 것이다. 소프트웨어 신뢰성 분야의 경우 미래에 발생되는 고장 수를 예측하는데 가장 널리 알려진 망으로는 FFNs(FeedForward Networks)와 PRNs(Partial Recurrent Networks)인 Jordan과 Elman 망이 있다. Karunanithi et al.[8, 9]는 신경망을 활용하여 소프트웨어 신뢰성 예측을 모델링하였으며, 소프트웨어 신뢰성 예측에 FFNs과 PRNs이 적용될 수 있음을 보였다.

데이타 쌍들 $(t_i, m_i), i=1, 2, \cdots, n$이 소프트웨어 시험 결과 관찰되었다고 가정하자. 여기서 m_i는 i번째 시험 시간인 t_i까지 발견된 누적 고장 수를 의미한다. 이들 데이타 쌍들 (t_i, m_i)을 고장 수 데이타라 칭한다. Karunanithi et al.[8, 9]는 훈련(Training)에 사용될 수 있는 데이타들로 (t_i, m_i), $i=1, 2, \cdots, l$을 가정하고, 미래의 d시점 m_{l+d}, $d=1, 2, \cdots$에 대한 예측을 하기 위해 2가지의 훈련 제도(Training Regimes)를 고려하였고, FFNs, Jordan 및 Elman 망에 적용하였다.

- 일반 훈련제도(Generalization training regime) : 입력(Input) t_i는 목표(Target) m_i에 관련
- 예측 훈련제도(Prediction training regime) : 입력 t_{i-1}는 목표 m_i에 관련

다양한 소프트웨어에서 수집된 고장 데이타를 이용 신경망 모델의 예측 정확성이 평가되고 일반적으로 널리 적용되고 있는 통계적 SRGM들과 비교되었다. 그 결과, 신경망은 다른 여러 가지 데이타에 대해 적합하였으며, 특히 장기 예측 정확성(Long-term predictive accuracy) 측면에서 통계적 모델들 보다 좋은 결과를 얻었다. 그러나 신경망 모델을 적용하는데 있어서 사용되는 입력-출력 데이타를 어떻게 표현할 것인가에 대한 해답은 명확히 얻지 못하였다. 따라서, 입력과 출력 값의 최대 값을 알고 있다는 가정하에 신경망의 은닉층과 출력층 뉴런의 활성화함수로 로지스틱 함수를 사용하였다.

데이타 표현(Data Representation)은 주어진 문제에 대한 입력·출력 변수 값을 어떻게 표현할 것인가의 문제로 신경망을 활용한 소프트웨어 신뢰성 성장 모델에서 중요한 문제중 하나이다. 대부분의 신경망은 은닉층(Hidden Layer) 또는 출력층(Output Layer) 뉴런의 활성화함수(Activation Function)로 로지스틱 함수를 가진 시그모이덜 뉴런 또는 단순히 선형 함수를 가진

뉴런을 사용한다. 로지스틱 활성화함수는 [0, 1] 또는 [-1, 1] 범위의 값만을 취하나 선형함수의 경우 표현되는 데이타 범위의 제한 영역이 한정되지 않아 모든 값을 표현할 수 있다 은닉층과 출력층 모두에 로지스틱 함수를 사용할 경우, 주어진 문제의 입력-출력 변수 값은 신경망의 뉴런에 사용되는 활성화함수가 취할 수 있는 값의 범위에 일치하도록 정규화 되어야만 한다.

일반적으로 입·출력 데이타의 범위가 명확히 밝혀진 문제에 대해서는 데이타를 정규화시키는데 문제가 발생되지 않는다 그러나, 미래의 고장발생 피정을 예측하는데 신경망을 사용하기 위해서는 현재까지 관찰된 고장 발생 데이타를 이용해야만 하며, 시험 종료 시점까지 발견된 누적 고장 수(즉, 정규화 데이타의 최대 값)을 모르는 상태이다. 이 경우, 데이타를 정규화하기 위해서는 소프트웨어 신뢰성 모델의 입력-출력으로 일반적으로 사용되는 전체 고장 이력 또는 시험 종료 시간 관찰 값을 갖고 있거나 예측해야만 한다는 문제점을 내포하고 있다[8,9]. 시험을 수행하는 도중에 소프트웨어 내에 남아 있는 모든 결함 수 또는 시험 종료 시간을 정확히 알기란 현실적으로 불가능하며, 대부분의 통계적 모델들이 이들 값을 추정하여 사용하고 있다. 따라서, 정규화된 데이타를 사용한 모델을 적용시 실제로 관찰된 데이타를 적용할 때보다 정확하지 못한 결과를 예측하며, 이로 인해 시험 종료 후 양도시점 또는 자원할당 의사결정에 사용되는 모델의 출력 값에 신뢰를 할 수 없게 되는 문제점을 갖고 있다.

또한, 신경망의 입·출력 변수 값의 범위가 [0, 1] 이내이어야만 한다는 것은 상식적으로 잘못된 생각이다. 비록 입·출력 값의 정규화가 때로는 이득이 있을지 몰라도 사실 그와 같은 요구조건이 명확히 기술된 것도 없다. 따라서 입력-출력 값을 정규화시켜 데이타를 구간 [0, 1]로 한정하는 것은 잘못된 판단이다. 만약 출력 뉴런의 활성화 함수가 범위 [0, 1]로 제한된다면 목표값(Target Value)들은 명백히 이 범위 이내로 정규화 되어야만 한다 그러니 관찰된 데이타를 출력 뉴런의 활성화함수에 일치하도록 하는 것보다는 관찰된 데이타의 분포에 적합하도록 출력 뉴런의 활성화 함수를 선택하는 것이 보다 좋은 선택 방법이 될 수 있다. 따라서, 3장에서는 이러한 기법을 구현하기 위한 모델을 제시하고자 한다.

3. 정규화하지 않은 데이타를 이용한 신경망 모델

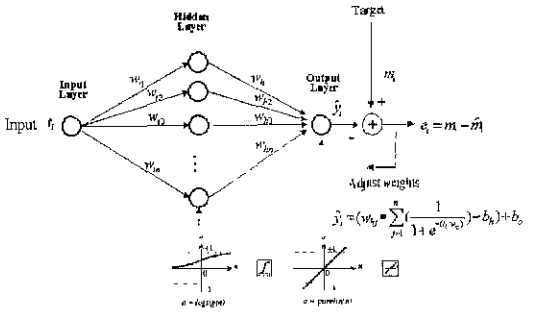
소프트웨어 시험을 수행하는 목적은 소프트웨어 내에 잔존하는 결함을 찾아내기 위한 과정으로 차후 운영단계에서 발생되는 결함의 영향을 최소화하기 위함이다. 따라서, 시험을 수행하는 중에는 소프트웨어에 잔존하는 총 결함 수를 정확히 판단할 수가 없으며, 미래의 고장 발생수를 예측하기 위해 신경망에 사용되는 관찰된 데이타의 입-출력 값을 [0, 1]로 정규화할 경우 최대 값을 결정할 수가 없는 문제점이 발생한다.

이 문제점을 해결할 수 있는 방법은 시험이 언제 종료될 것인가 또는 소프트웨어에 잔존하는 총 결함수가 얼마인가를 전혀 고려하지 않은 상태에서 단지 발생된 고장 데이타를 직접 신경망에 입력하고 신경망 학습결과 획득된 출력도 소프트웨어 시험시 관찰된 고장 데이타 또는 미래의 시점에서 관찰될 고장 데이타 범위로 예측하는 방법이 최적의 신경망 예측 정확성을 얻을 수 있다

일반적인 다층 신경망인 FFN이나 PRN에 대해 관찰된 데이타를 정규화하지 않고 신경망에 직접 적용할 수 있는 방법은 은닉층 뉴런의 활성화 함수로는 로지스틱 함수를, 출력층의 뉴런에는 선형함수를 사용하면 정규화 하지 않은 데이타가 은닉층을 거치면서 입력되는 데이타의 분포를 충분히 표현하고 출력층을 통해 원 데이타 범위내의 결과를 표현할 수 있다 그러나 출력층 뉴런에 로지스틱 함수를 사용하면 출력되는 값이 [0, 1] 범위로 한정되므로 사용이 불가하다.

Cybernko[2]는 선형함수를 이용하여 모든 분제에 대한 보편적 근사 이론(Universal Approximation Theorem)을 증명해 보였으며, 선형함수로 모든 함수를 표현 가능하기 때문에 이와 같은 방법을 사용해도 타당한 근거가 되기 때문이다. Karunanithi et al [8,9]는 출력층 뉴런의 활성화 함수로 로지스틱 함수를 사용하였다. 이 경우 입력·출력 변수 값의 변환이 필요하나 본 논문의 접근방법은 이따한 데이타 변환도 필요하지 않다. 비정규화 데이타를 사용할 수 있는 신경망을 (그림 1)에 표현하였다

미래의 고장 수를 예측하기 위한 알고리즘 및 해를 구하는 방법은 다음과 같다. t_i시점까지 시험하여 수집



(그림 1) 비정규화 데이터를 사용하는 FFN

된 데이터 $(t_i, m_i), i = 1, 2, \cdots, l$에 대해 신경망의 입력으로 시험시간 t_i를, 신경망의 목표 출력 값으로 누적 고장수 m_i로 설정하여 신경망을 훈련시킨다 훈련이 완료된 신경망에 대해 미래의 다음 단계 시험시간 t_{l+1}를 입력하였을 경우 훈련된 신경망의 출력으로 다음 단계의 누적 고장 수를 예측하는 알고리즘으로 신경망의 구성 및 훈련은 MATLAB 5.2 Neural Network Toolbox Ver. 3.0을 이용하였다

정규화하지 않은 원 데이타를 사용하는 경우의 장점은 다음과 같다. (1) 데이타를 정규화하기 위해서는 소프트웨어의 전체 고장이력을 갖고 있거나 시험 종료시점에서 발견된 누적 결함 수를 추정해야만 한다. 그 결과 예측 결과는 부적절해질 수 있다 ; (2) 정규화 방법의 경우, 신경망을 훈련시키기 전에 원 데이타를 정규화 하여야하며, 훈련시킨 후 출력되는 예측 오차 또는 누적 결함 수는 원 데이타 범위로 다시 변환시키는 과정에서 불필요한 시간이 소요된다 ; (3) 정규화된 데이타는 적은 데이타 범위에서의 예측오류율이 적을 지라도 원 데이타인 큰 데이타 범위로 변환시 오류 변화 효과는 위험한 예측 문제점을 가져올 수 있다 ; (4) 임의의 값을 가진 데이타 범위를 정규화시켜 신경망을 훈련한 결과로 미래의 누적 고장발생 과정을 예측할 경우, 새로 입·출력되는 데이타가 기존 정규화된 데이타 범위를 초과할 경우 초과되는 부분이 잘려나간 상태의 값으로 원래의 누적 고장 수 값으로 변환시 예측 결과에 오류를 유발시킬 수 있다.

4. 예측력 실험 및 결과 분석

본 장에서는 고장 수 데이타에 대해 정규화된 데이타와 정규화 되지 않은 데이타 상호간 비교 분석을 통해 제안된 신경망 모델의 예측 성능을 판단하고자 한다. Karunanithi et al.[8, 9]는 소프트웨어 신뢰성 예측 분야에 FFNs와 Jordan 망이 적합함을 보였다. 본 논문에서는 단지 1개인 은닉층을 가진 FFN에 한정하여 비교 분석하고자 한다. 예측력을 분석 및 평가하기 위해 2종류의 고장 수 데이타를 선택하였으며, 이들 데이타는 Karunanithi et al.[9]의 Data2와 Data14이며 <표 1>에 기술하였다.

〈표 1〉 선택된 데이타 특성 및 고장 데이타

데이터	참고문헌	LOC	누적 고장 수	데이터 크기	적용분야
Data 2	[12]	21,700	136	25	Realtime Control
Data 14	[15]	not known	266	46	Realtime Control

Data2			Data14					
시험 시간	발견 고장수	누적 고장수	시험 시간	발견 고장수	누적 고장수	시험 시간	발견 고장수	누적 고장수
1	27	27	1	2	2	26	7	184
2	16	43	2	0	2	27	0	184
3	11	54	3	30	32	28	22	206
4	10	64	4	13	45	29	2	208
5	11	75	5	13	58	30	5	213
6	7	82	6	3	61	31	12	225
7	2	84	7	17	78	32	14	239
8	5	89	8	2	80	33	5	244
9	3	92	9	2	82	34	2	246
10	1	93	10	20	102	35	0	246
11	4	97	11	13	115	36	7	253
12	7	104	12	3	118	37	3	256
13	2	106	13	3	121	38	0	256
14	5	111	14	4	125	39	0	256
15	5	116	15	4	129	40	0	256
16	6	122	16	0	129	41	0	256
17	0	122	17	0	129	42	5	261
18	5	127	18	0	129	43	2	263
19	1	128	19	0	129	44	3	266
20	1	129	20	0	129	45	0	266
21	2	131	21	0	129	46	0	266
22	1	132	22	30	159			
23	2	134	23	15	174			
24	1	135	24	2	176			
25	1	136	25	1	177			

FFN의 경우, 신경망 훈련제도로는 2장에서 기술한 Karunanithi et al.[9]의 "일반적 훈련제도"에 대해 고려하였다. 신경망의 적절한 구조를 선택하기 위해 본 연구에서는 은닉층이 1개인 FFN에 대해 시행착오법(Trial-and-Error)을 사용하고자 한다. 따라서 예측하고자 하는

시점에서 은닉 뉴런 수가 1개인 FFN을 구성하고 최적 시점 종료 기법(Early Stopping 또는 Optimal Stopping)을 사용하여 오차가 최소가 되는 시점에서 신경망의 훈련을 멈추고 예측 오차를 계산한다. 은닉 뉴런 수를 1개에서 50개까지 1개 단위로 변화시켜 가면서 예측 오류를 계산한다. 이 50개의 예측 오류 값 중 가장 최소의 예측 오류를 가진 뉴런이 해당 시점에서의 신경망의 최적의 뉴런수로 결정된다. 데이타 (t_i, m_i), $i=1, 2, \cdots, l$ 개에 대해 $i-1$개의 관찰된 데이타는 신경망 훈련 데이타에, 나머지 1개는 검증 데이타로 사용된다. 이들 l개의 관찰된 데이타로부터 $l+1$번째 데이타가 예측을 위한 시험 데이타로 사용된다. 즉, 바로 다음 단계에 대한 예측(One-step ahead prediction)을 수행한다. 훈련과 검증 데이타의 비율에 따라 신경망의 예측 성능에 영향을 미친다. 그러나 최적의 검증 데이타 비율은 주어진 문제에 의존할 수밖에 없으며, Bös[1]는 최적의 검증 데이타 비율을 찾는 것은 매우 어려우며 단지 적은 비율의 검증 데이타로도 좋은 결과를 얻는데 충분함을 제안하였다 따라서, 고장 수 데이타의 다음 단계 예측에 사용되는 훈련 데이타의 개수는 극히 작은 관계로 훈련과 검증 데이타의 비율을 $l-1:1$로 선정하였으나 다른 데이타 분할 비율도 적용 가능하다. 신경망의 훈련과 검증에 필요한 최소한의 데이타 크기 l 은 전체 데이타 크기 n의 약 20%로 설정하고 l을 1개씩 증가시킨다. 시험 데이타에 대해 예측 상대오차(Relative Prediction Error)가 계산되며, 다음과 같이 정의된다.

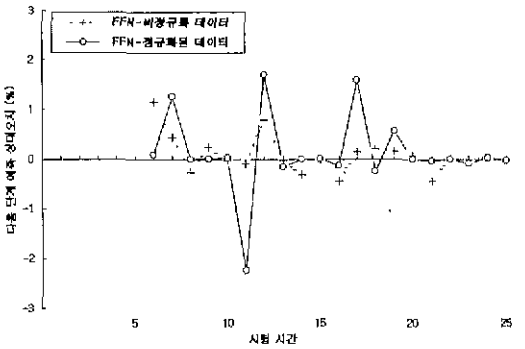
$$e_{l,j}=\frac{(l+j)\text{번째 예측된 값}-(l+j)\text{번째 실제 값}}{(l+j)\text{번째 실제 값}}\times 100.$$

정규화된 데이타와 정규화 되지 않은 데이타의 예측력을 평가하기 위해 다음 단계 예측 상대오차의 절대값(Next-step absolute relative prediction error)의 평균인 $\overline{e}_{l,r}=\sum_{j=1}^{r}|e_{l,j}|/r$, $r=1$을 사용한다[13]. 즉, $e_{l,1}=\overline{e}_{l,1}$. 그러나, 다른 r의 값이 고려될 수도 있다.

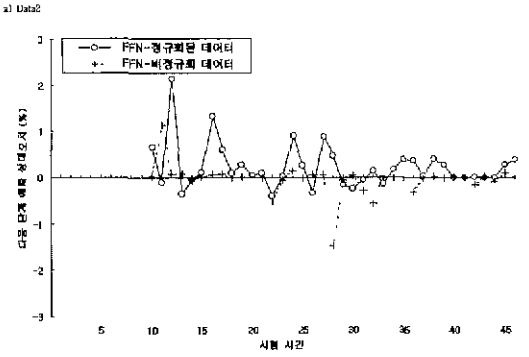
주어진 데이타로부터 신경망의 적절한 구조와 가중치 값을 정확히 알 수 없기 때문에 대부분의 신경망 훈련방법은 신경망의 가중치들의 초기 값을 랜덤하게 설정한다. 따라서 신경망을 훈련시킬 때마다 랜덤하게 설정된 초기 가중치 차이로 인해 신경망의 예측력에 차이가 발생한다. 따라서 20회의 훈련을 통해 평균값을 취한 것으로 예측 오차를 구하고자 한다. 선택한 모델이 주어진 데이타에 적합한 모델인지를 평가하기 위해 만약 몇 개의 모델이 동일한 예측 오차를 나타내면 통계적 관점에서 보다 단순한 구조가 보다 좋은 모델로 선정될 수 있다. 따라서 단순성(보다 적은 은닉 뉴런 수)과 보다 좋은 예측 오차를 모델 선택 기준으로 선정하였다. 주어진 데이타에 대해 Karunanithi et al.[8, 9]이 적용한 정규화시킨 데이타를 이용한 신경망과 제안된 정규화 시키지 않은 원 데이타를 이용하는 신경망의 예측 정확성을 시험의 각 시간별로 비교한 결과는 (그림 2)에, 다음단계의 전체 평균 예측 오차 $\overline{e}_{l,1}$는 <표 2>에 표기하였다 (그림 2)와 <표 2>에서 Karunanithi et

〈표 2〉 다음단계의 전체 평균 오차 $\overline{e}_{l,1}$

데이타	다음단계의 전체 평균 오차 $\overline{e}_{l,1}$	
	비정규화 데이타	정규화된 데이타
Data2	0.2416	0.4088
Data14	0.1503	0.3334



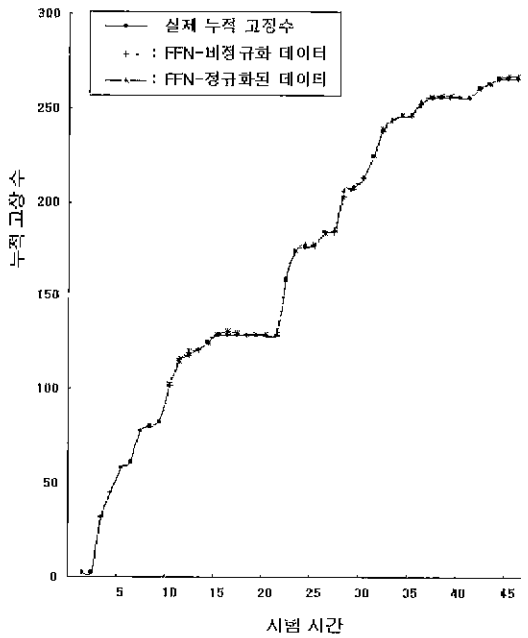
(a) Data2



(b) Data14

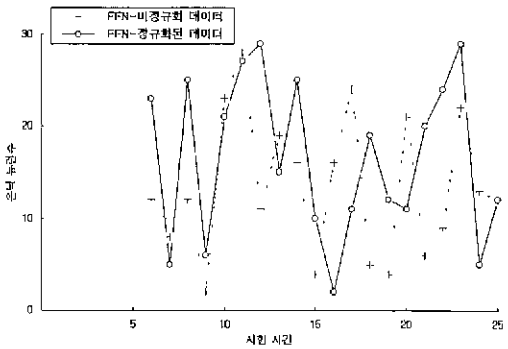
(그림 2) 다음 단계 예측 상대오차

al.[8, 9]의 정규화된 데이타를 이용하는 신경망 보다 정규화 하지 않은 데이타를 사용하는 제안된 신경망이 보다 좋은 예측 결과를 얻을 수 있음을 알 수 있다. 또한 Data14의 누적 고장수 에 대한 예측 결과를 (그림 3)에 제시하였다 (Data2도 유사한 경향을 보임) 각 예측 단계별로 적합한 은닉 뉴런 수는 (그림 4)에 표기하였다. 따라서, 은닉 뉴런 수는 데이타의 수에 따라 민동이 심하며, 데이타 표현 방법(정규화 유무)과는 무관함을 알 수 있다

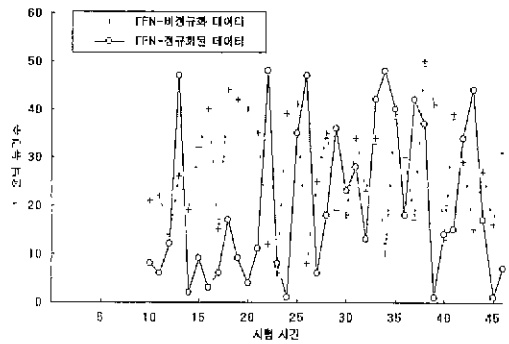


(그림 3) 예측 누적 고장 수(Data14)

원 데이타를 직접 적용하는 것이 예측력이 훨씬 좋음을 알 수 있었다 따라서 소프트웨어 신뢰성 예측 경우와 같이 데이타의 최대 값을 모를 경우, 추정된 최대 값을 사용하는 정규화 방법보다 제안된 모델이 월등히 좋은 예측 정확성을 나타낼 수 있다



(a) Data2



(b) Data14

(그림 4) 예측 단계별 최적의 은닉 뉴런 수

5. 결론 및 향후 과제

본 논문에서는 소프트웨어 시험시 관찰된 고장 데이타를 정규화 하지 않은 원 데이타를 신경망에 적용하여 소프트웨어 신뢰성을 예측하는 분야에 관해 연구하였다. 망 구조로는 FFN을 선택하였으며, 적절한 은닉 뉴런 수를 결정하기 위해 관찰된 데이타를 훈련과 검증 데이타로 분류하고 관찰되지 않은 다음 단계 데이타를 시험 데이타로 하여 예측 오차를 측정하는 방법으로 최적시점 종료기법을 적용하였다 실험을 통해 살펴본 바와 같이 최대 값을 알고 있다고 가정 한 상태에서의 정규화 데이타의 결과보다 정규화하지 않은

소프트웨어 신뢰성 예측 분야에 신경망을 적용시 수집된 고장 데이타를 정규화하지 않을 경우 이점은 다음과 같이 결론지을 수 있다 , (1) 정규화하지 않은 데이타 처리 방법은 누적 고장 수와 시험 완료시간의 최대 값을 사전에 추정하지 않아도 되며, 단지 관찰된 데이타만을 사용해 신경망을 훈련시켜 예측력을 판단할 수 있다 ; (2) 원 데이타의 정규화 또는 정규화된 신경망의 출력 값을 원 데이타로의 변환과정은 귀찮은 일로 판단된다. 그러나 정규화 하지 않은 데이타(원 데이타)를 사용하는 제안된 방법은 이 작업이 필요 없이 보다 간단하게 적용할 수 있다.

본 논문에서는 관찰된 /개의 데이타로부터 훈련 데

이타로 7-1개를, 나머지 1개를 검증 데이터로 선택하였다. 그러나 훈련과 검증 데이터를 구분하는 비율에 따라 신경망의 예측력에 영향을 미칠 수 있으며, 훈련 제도가 변화해도 영향을 받을 수 있다. 따라서, 추후로는 이 부분에 대한 연구가 수행될 것이다.

참 고 문 헌

[1] S. Bös, "How to Partition Examples between Cross-Validation Set, and Training Set?," Lab. for Information Representation, RIKEN, 1996

[2] G. Cybenko, "Approximation by Super-positions of A Sigmoidal Function," Mathematics of Control, Signals and Systems, Vol.2, pp 303-314, 1989.

[3] J. L Elman, "Finding Structure in Time," Cognitive Science, pp.179-211, 1990

[4] A. L. Goel, "Software Reliability Models Assumptions, Limitation, and Applicability," IEEE Trans. on Software Eng Vol SE-11, No.12, pp.1411-1423, 1985.

[5] L. Holmström, P. Koistinen, J Laaksonen, and E. Oja, "Neural and Statistical Classifiers-Taxonomy and Two Case Studies," IEEE Trans on Neural Networks, Vol.8, No.1, pp.5-17, 1997.

[6] R. H. Hou and S Y Kuo, "Applying Various Learning Curves to Hypergeometric Distribution Software Reliability Growth Model," IEEE. 1994.

[7] M L. Jordan, "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine," Proc 8th Annual Cognitive Science, pp.531-546, 1986.

[8] N. Karunanithi, D Whitley, and Y. K. Malaiya, "Prediction of Software Reliability Using Connectionist Models," IEEE Trans. on Software Eng., Vol. 18, No.7, pp.563-574, July 1992.

[9] N. Karunanithi, D. Whitley and Y. K. Malaiya, "Using Neural Networks in Reliability Prediction," IEEE Software, pp.53-59, 1997.

[10] T. M Khoshgoftaar, E. B. Allen, J P. Hudepohl, and S. J. Aud, "Application of Neural Networks to Software Quality Modeling of a very Large Telecommunications Systems," IEEE Trans. on Neural Networks, Vol.8, No.4, pp.902-909, 1997.

[11] M R Lyu, "*Handbook of Software Reliability Engineering.*" IEEE Computer Society Press, 1996.

[12] J. D Musa, "Software Reliability Data," Technical Report, Data and Analysis Center for Software, Rome Air Development Center, Griffins AFB, New York, 1979.

[13] J. Y. Park, S. U. Lee, and J. H. Park. "Neural Network Modeling for Software Reliability Prediction from Failure Time Data," Journal of Electrical Eng. and Information Science, Vol.4, No.4, pp.533-538, 1999

[14] F. Popentiu and D. N. Boros, "Software Reliability Growth Supermodels," Microelectron. Reliab. Vol. 36, No 4, pp.485-491. 1996.

[15] Y. Thoma et al., "Parameter Estimation of the Hyper-Geometric Distribution Model for Real Test/Debug Data," Dept. Computer Science, Tokyo Inst. Tech., Tech. REP. 901002, 1990.

[16] R. J Williams and D. Zipser, "A Learning Algorithm for Continuous Running Fully Recurrent Neural Networks," Neural Computation, Vol.1, pp. 270-280, 1989.

이 상 운

e-mail : sangun_lee@hanmail.net

1983년~1987년 한국항공대학교 항공전자 공학과(학사)

1995년~1997년 경상대학교 전자계산학과(석사)

1998년~현재 경상대학교 전자계산학과 박사과정

관심분야 : 소프트웨어 공학(소프트웨어 시험 및 품질보증, 소프트웨어 신뢰성), 신경망, 퍼지