2024年 2月
博士學位論文

종속적 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형에 관한 연구

朝鮮大學校 大學院

電算統計學科

金 倫 秀

종속적 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형에 관한 연구

A Study on the Dependent Software Reliability Models and Deep Learning Software Reliability Models

2024年 2月 23日

朝鮮大學校 大學院

電算統計學科

金 倫 秀

종속적 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형에 관한 연구

指導敎授 張 仁 弘

이 論文을 理學 博士學位 申請 論文으로 提出함

2023年 10月

朝鮮大學校 大學院

電算統計學科

金 倫 秀

金倫秀의 博士學位論文을 認准함

敎 授 權 容 萬 (인)

敎 授 鄭 璣 文 (인)

敎 授 金 世 珍

敎 授 宋 光 胤 (인)

敎 授 張 仁 弘 (인)

2024年 1月

朝鮮大學校 大學院

목 차

표 목 차

그 림 목 차

ABSTRACT

A Study on the Dependent Software Reliability Models and Deep Learning Software Reliability Models

Kim, Youn Su
Advisor : Prof. Chang, In Hong, Ph.D.
Department of Computer Sceience and statistics,
Graduate School of Chosun University

As times have progressed, software has become a very important part of every field. From the smallest tasks to the most important ones, such as meetings, collaboration, and decision-making, software has made it easier for us to do things, and it's rare that we don't use it. In addition, we are now living in the era of the Fourth Industrial Revolution. The fourth industrial revolution utilizes artificial intelligence and big data to think and make decisions like humans using computer technology, and it has become very close to our lives. Software plays a huge role in this as well.

If the software you use is faulty and produces incorrect results, or the entire software fails and stops working, the damage will be enormous. In particular, Artificial Intelligence(AI) relies on real-time data, which greatly affects the speed of computation. Since the data is updated in real time, a once computational error that results in incorrect results can have a huge social and economic impact. So, the software reliability is so important. Software reliability is a field that measures how well software works and develops software reliability models based on software failure data. A software reliability model is a model that uses mathematical assumptions to predict the number of software failures, and the results of

the predictions can be used to help improve software reliability. Most software failures follow a Poisson distribution, and most software reliability models assume a Non-Homogeneous Poisson Process(NHPP), which assumes a different number of failures at different points in time, rather than a constant number of failures. Software reliability models have evolved to include more assumptions, such as software failures occurring independently of each other, testing effort, incomplete debugging, and uncertainty in the operating environment.

Many traditional software reliability models have the problem that they only fit well in special cases due to the mathematical addition of special assumptions. In addition, as the structure of software has become more complex and multiple pieces of software are composed together, the environment of software has become very diverse. It is not easy to find an assumption that fits the complexity of the structure because one assumption cannot be used to assume the same environment for all software. To solve this problem, we proposed a software reliability model using deep learning among machine learning methods that rely on given failure data.

In addition, models that rely only on data lack a logical basis because they do not make mathematical and statistical assumptions. To solve these problems, we proposed a deep learning software reliability model that replaces the activation function used in deep learning with a software reliability model. Among the activation functions utilized in deep learning, the sigmoid function is the most common structure of the software reliability model. We proposed a deep learning software reliability model that uses the software reliability model as the activation function because replacing this form with the existing software reliability model functions as an activation function with mathematical assumptions instead of a model that only depends on data.

The three proposed models demonstrated the superiority of the models through the criteria based on nine estimates and the criteria measure for

one prediction. In addition, the new software reliability model assuming dependent failures proposed an optimal release time considering the costs involved in software development and release based on the estimated number of failures. The software reliability model using deep learning and the deep learning software reliability model estimated and trained each parameter on 90% of the data set, calculated the predicted value on the remaining 10% of the data, and compared whether the actual value fell within the 95% confidence interval based on the predicted value.

Software is critical now and will be in the future. As software develops and evolves into more complex structures, there will continue to be a need for research on software reliability for complex structures. We believe that the three software reliability models proposed in this paper are well suited to the complex structure of current software, and if further research is conducted, it is possible to propose a very useful model in the field of software reliability.

제1장 서 론

제1절 연구 배경

소프트웨어는 어떤 분야를 막론하고 작은 부분부터 전체 시스템까지를 아우르는 프로그램으로 컴퓨터에게 인간이 원하는 작업을 할 수 있도록 지시하는 명령어의 집합체이다. 소프트웨어라는 단어는 John W. Tukey가 1957년도에 처음으로 사용한 것으로, “부드러운”의 의미를 가진 soft와 “제품”이라는 뜻을 가진 ware의 합성어이다. 따라서 컴퓨터를 포함하고 있는 시스템에서 특정 작업을 수행하도록 하는 프로그램의 집합이다(Moon, 2017).

과거 소프트웨어는 계산기와 같이 매우 작은 부분 또는 간단한 작업 등을 인간 대신 할 수 있도록 만들었다. 하지만 컴퓨터의 하드웨어가 발달함에 따라 연산 속도가 증가하고, 저장 공간도 기하급수적으로 늘어나면서 위의 능력들을 수행하는 것은 매우 간단한 일들로 발전했고, 해당 일들을 묶어서 한 번에 처리할 수 있도록 발전했다. 소프트웨어의 발전으로 스마트폰, 자동차 등과 같은 고성능 소프트웨어가 탑재된 기기, 게임 등의 여러 가지의 일을 컴퓨터가 동시에 처리할 수 있도록 여러 가지의 복잡한 명령어를 통해 구현시키는 능력까지를 아우르게 되었다. 소프트웨어는 점점 인간이 구현시키고자 하는 능력이 다양해지고 복잡해짐에 따라 알고리즘이나 구현시키고자 하는 코드가 매우 다양해졌으며 복잡해졌다. 또한, 연산을 최대한 시간을 단축시켜 연산할 수 있도록 코드를 구성하는 것 역시 매우 중요해졌다. 현재는 소프트웨어가 복잡해진 구조를 넘어서 전체를 관장하는 형태로 발전하여 어느 분야에서건 작은 일이라도 소프트웨어가 관여하지 않는 분야는 없다고 봐도 무방하다. 데이터를 저장하는 일, 간단한 문서작업 등 이런 모든 일들이 소프트웨어로부터 시작된다. 그만큼 우리는 소프트웨어에 매우 의존한 삶을 살고 있다.

현재 우리는 2016년 다보스 포럼에서 Klaus Schwab이 선언한 제 4차 산업혁명의 정중앙에 위치해 있다. 4차 산업혁명은 단순한 데이터가 아닌 규모가 크고(Volume), 다양하며(Variety), 처리 및 저장 속도가 빠른(Velocity) 빅데이터를 기반으로 이뤄지고 있다. 이는 컴퓨터 하드웨어의 성능과 구성되어 있는 소프트웨어의 연산 속도에 매우 민감하며, 컴퓨터의 성능과 소프트웨어의 빠른 연산 구조는 해당 산업의 흥망성쇠를 좌우한다. 컴퓨터 성능이 좋을수록, 알고리즘 성능이 빠르고 효율적일수록 좋은 결과를 보이기 때문에 현재 산업에서도 소프트웨어는 매우 중요한 부분을 차지하며, 이는 인간의 능력으로 컴퓨터 성능을 최대치로 이끌어 낼 수 있는 부분이기

때문에 중요하다. 따라서 시간이 지날수록, 사회가 발전할수록 더욱더 컴퓨터 및 소프트웨어에 의존할 수밖에 없다(Song, 2017; Jang과 Lee, 2019).

4차 산업혁명의 모든 기술은 컴퓨터 및 소프트웨어에 의존한다. 이 중 가장 대표적인 분야는 인공지능 분야이다. 인공지능은 인간처럼 생각할 수 있는 기계를 만드는 분야로, 인간이 사고하는 방식을 컴퓨터에 이입하여 인간이 사고하며 판단하는 것들을 컴퓨터 역시 알고리즘에 맞춰 사고하고 판단하도록 유도한다. 인간이 오감을 통해 보고, 맛보고, 느끼는 정보를 컴퓨터는 데이터를 통해 학습한다. 학습한 인공지능은 인간보다 빠른 연산 과정을 통해 최적의 판단을 도출한다. 따라서 인공지능은 컴퓨터의 성능도 중요하지만, 최적의 결과를 도출하기 위한 알고리즘이나 이를 구성하고 있는 여러 가지의 소프트웨어도 매우 중요하다. 인공지능을 구성하고 있는 소프트웨어가 코드에 의해, 외부의 충격에 의해, 여러 소프트웨어가 함께 작동하면서 발생하는 오류에 의해 고장 나게 된다면, 이는 매우 큰 문제를 일으킬 수 있다. 특히, 인공지능은 실시간 데이터에 의존하는데, 이는 연산 속도에 큰 영향을 미친다. 실시간으로 데이터가 업데이트되기 때문에 한 번의 연산 오류로 인해 잘못된 결과를 출력하게 된다면, 사회·경제적으로 큰 타격을 입힐 수 있다. 또한, 인공지능을 구성하고 있는 소프트웨어는 보이지 않는 문제를 야기할 수 있다. 코드나 여러 소프트웨어의 상호작용을 통해 발생하는 오류를 옳다고 판단하여 학습하는 경우, 예상할 수 없는 시스템 고장까지 발생할 수 있다. 해당 오류를 통해 학습한 결과가 옳은지 옳지 않은지를 판단하는 건 매우 어렵기 때문에 해당 인공지능으로 도출한 결과를 신뢰할 수 있는지도 따져봐야 한다.

따라서 인공지능의 기술은 매우 유망하며 대단한 기술이지만, 맹신할 수만은 없다. 2016년 미국의 한 쇼핑센터에서 경비 로봇이 16개월 된 아이를 넘어뜨려 다리를 다치게 하는 사고가 일어났고, 중국에서도 한 IT 전시회에서 교육용 로봇이 갑자기 전시장 유리를 깨뜨리고 그 파편으로 인해 방문객이 부상을 입는 사고가 발생했다. 2018년에는 미국 뉴저지 아마존 물류센터의 로봇이 갑자기 오류를 일으키며 곰 퇴치 스프레이 통을 찢어버리며 유독물질이 유출돼 무려 24명의 직원이 입원한 사고도 일어났다. 해당 사고 이후 아마존 직원 노동조합에서는 로봇이 인간 근로자에게 끼치는 위험에 대해 경고하는 성명을 발표했다.

이러한 문제를 안고 있는 현재의 불안정한 인공지능을 구성하고 있는 소프트웨어에 대해서도 신뢰할 수 있도록 신뢰성을 판단해야 하므로 소프트웨어 신뢰성은 매우 중요하며, 여러 방면으로 발생하는 소프트웨어의 고장을 줄이고 잘 관리하기

위해 소프트웨어 신뢰성에 관한 연구는 필연적이다. 소프트웨어 신뢰성은 소프트웨어가 얼마나 잘 작동하는지를 측정하는 분야로, 소프트웨어 고장데이터를 기반으로 소프트웨어 신뢰성 모형을 개발한다. 소프트웨어 신뢰성 모형은 소프트웨어의 신뢰성을 예측하는 방법 중 하나이다. 이에 대한 연구는 마코프 모형, 비동질 포아송 과정(Non-homogeneous poisson process; NHPP), 베이지안 방법 등 수학적·통계적인 가정을 기반으로 연구가 진행되고 있으며, 이 중 소프트웨어 고장은 포아송 분포를 따르기 때문에 현재 소프트웨어 신뢰성 모형에 관한 연구는 NHPP를 가정한 소프트웨어 신뢰성 모형을 바탕으로 연구가 이뤄지고 있다. NHPP는 소프트웨어 고장이 시간의 흐름에 따라 고정된 고장 및 결함을 가정하는 동질성 포아송 과정이 아닌 시간의 흐름에 따라 변화되는 고장 및 결함이 반영되는 과정을 의미한다. 여기서의 고장 및 결함은 평균값함수인 $m(t)$를 기반으로 구성된다.

초기 NHPP를 가정한 소프트웨어 신뢰성 모형은 Goel과 Okumoto(1979)로부터 지수분포를 통해 소프트웨어 고장이 독립적으로 발생할 것을 가정하면서 시작되었으며, 이를 기반으로 여러 가정이 추가된 소프트웨어 신뢰성 모형이 개발되었다. Yamada 외(1983), Ohba(1984)는 소프트웨어 누적 고장 수가 S자 형태로 증가함을 가정한 소프트웨어 신뢰성 모형을 제안하였다. 이를 더 확장하여 Yamada 외(1986)는 테스트 단계에서 투자된 여러 가지의 노력이 반영된 소프트웨어 신뢰성 모형을 제안했으며, 여러 가지의 노력은 레일리(Rayleigh) 분포를 가정하였다. 이 당시는 소프트웨어 출시(Release) 전 테스트 단계에서 검출된 결함 및 오류가 수정, 제거되고, 새로운 결함이 디버깅(Debugging) 과정에서 발생하지 않음을 가정한 모형이었다. 이를 개선하기 위해 현실적인 측면을 추가하여 테스트 단계에서 검출된 결함이 수정, 제거가 이뤄지지 않는 불완전 디버깅(Imperfect debugging)을 가정한 소프트웨어 신뢰성에 관한 연구가 진행되었다. Yamada 외(1992)는 결함탐지율 함수 $b(t)=b$로 일정한 불완전 디버깅을 가정한 소프트웨어 신뢰성 모형을 제안하였고, Zhang 외(2003)는 $b(t)$가 일정한 상수가 아닌 함수인 불완전 디버깅을 가정한 소프트웨어 신뢰성 모형을 제안하였다. Pham과 Zhang(1997)과 Pham 외(1999)는 지수함수 형태인 결함탐지율 함수와 소프트웨어 누적 고장 수가 변곡(Inflection) S자 형태인 소프트웨어 신뢰성 모형을 제안하였다. Pham과 Zhang(2003)은 소프트웨어의 테스트가 얼마나 충족되었는지를 나타내는 지표인 테스트 커버리지(Test coverage)를 고려한 소프트웨어 신뢰성 모형을 제안하였다. Kapur 외(2011)는 테스트 단계에서 발견된 결함 및 오류가 수정되는 것과 수정되지 않는 것을 함께 고려

한 모형으로 완전 디버깅과 불완전 디버깅을 동시에 고려한 모형을 제안하였으며, Roy 외(2014)는 테스트 단계에서 불완전 디버깅 및 오류가 발생하는 경우, 초기 단계에 급격히 증가하다가 일정한 형태로 결함이 검출되는 경우를 고려한 소프트웨어 신뢰성 모형을 제시하였다.

소프트웨어는 다양한 분야에서 활용되기 때문에 이를 개발하고 운용하는 환경은 분야마다 다르고 다양해졌다. 앞서 소개한 소프트웨어 신뢰성 모형은 다양한 운용환경을 고려하지 않고 소프트웨어 신뢰성 모형을 개발하였다면, 2000년대 이후에 개발된 소프트웨어 신뢰성 모형은 다양해진 운용환경을 반영한 소프트웨어 신뢰성 모형에 관한 연구를 시도하였다. Teng과 Pham(2006)은 운용환경의 불확실성을 고려한 일반화된 소프트웨어 신뢰성 모형을 제안하였고, Pham(2014)은 운용환경에 따라 단위 시간당 소프트웨어 결함탐지율의 불확실성을 포함한 소프트웨어 신뢰성 모형을 제안하였다. Chang 외(2014)는 운용환경의 불확실성을 고려한 결함탐지율 함수와 새로운 테스트 커버리지를 이용한 소프트웨어 신뢰성 모형을 제안했고, Song 외(2019a)는 테스트 커버리지와 운용환경의 불확실성에 대한 무작위성을 고려한 소프트웨어 신뢰성 모형을 제안하였으며, 이를 기반으로 민감도 분석을 통해 모수가 미치는 영향을 분석하였다.

소프트웨어 신뢰성을 유도하기 위한 고장검출률 함수에 다른 가정들을 추가한 소프트웨어 신뢰성 모형에 관한 연구가 진행되었다. Song 외(2017a)는 세 개의 모수를 포함하고 있는 결함탐지율 함수를 활용하여 불확실성 운용환경을 고려한 소프트웨어 신뢰성 모형을 제안하였다. Song 외(2018)는 운용환경에서 고장이 발생할 때, 고장 제거 확률에 영향을 받는 고장검출률 함수를 통해 소프트웨어 신뢰성 모형을 제안하였다. Song 외(2019b)는 운용환경 불확실성을 고려하며 고장검출률 함수가 변곡 계수를 갖는 소프트웨어 신뢰성 모형을 제안하였다. Raghuvanshi 외(2021)는 소프트웨어의 결함 탐지 및 최대 결함 수를 고려한 시간 가변 소프트웨어 신뢰성 모형을 제안하였다. 이는 모수의 평가를 위해 시간 가변 유전 알고리즘 과정이 구현되며, NHPP를 기반으로 소프트웨어에서 제거되지 않은 오류를 통합하였다. Das 외(2022)는 오류를 감지하는 순간만이 아닌 미리 지정된 디버깅 시간에 오류를 수정하는 주기적 디버깅에 적합하도록 각 오류의 오류 발생에 대한 NHPP 연속 시간 소프트웨어 신뢰성 모형을 제안하였다.

하지만 소프트웨어 고장은 독립적으로만 발생하지 않는다. 특히, 소프트웨어 구조가 복잡해지고 다양해지면서 여러 소프트웨어끼리 관계 맺어 작동하기 때문에 고장

이 발생했을 때 다른 소프트웨어의 고장에 영향을 미치지 않는다고 단정 지을 수 없다. 따라서 종속 고장이 발생함을 가정한 모형이 제안되었다. 여기서 종속 고장이란 하나의 고장이 다른 고장에 영향을 주거나 다른 기기의 고장확률을 높이는 데 영향을 주는 것을 의미한다(Li와 Pham, 2021). 종속 고장의 종류는 크게 두 가지가 존재한다. 공통원인 고장(Common cause failure)은 한 원인에 의해 여러 소프트웨어가 동시에 고장일 발생하는 것이고, 종속 고장(Dependent failure)은 일부 시스템에서 고장이 발생하여 다른 소프트웨어까지도 영향을 미치는 경우이다. Pham과 Pham(2000)은 불완전 디버깅을 가정한 소프트웨어 신뢰성 모형에서 소프트웨어 고장 수 함수인 $a(t)$, 고장검출률 함수인 $b(t)$가 종속적 관계를 갖는 것을 가정한 모형을 제안하였다. 또한, Lee 외(2020), Kim 외(2022)에서는 과거의 오류가 잘 고쳐지지 않았다면 소프트웨어의 고장에 계속적으로 영향을 미칠 것을 가정한 모형을 제시하였다. Lee 외(2022)는 위의 종속적 고장과 불확실성 운용환경을 동시에 고려한 소프트웨어 신뢰성 모형을 제안하였다. Song 외(2023)는 오픈소스 소프트웨어를 활용하는 유한 결함과 종속 결함 수를 고려한 소프트웨어 신뢰성 모형을 제안하였으며, 새로운 통합 적합도 척도를 활용하여 모형의 우수성을 입증하였다.

독립적·종속적 가정에 의해 연구된 여러 소프트웨어 신뢰성 모형은 연구가 발전하면서 특수한 경우에 적합한 모형이 많이 개발되었으며, 이를 일반화하는데 어려움이 발견되었다. 또한, 이는 누적된 고장데이터를 기반으로 개발한 모형이기 때문에, 즉각적인 고장에 대처하기에는 문제가 있을 수 있다. 수학적·통계적 모형은 고장의 추세에 따른 예측을 보이지만, 고장은 언제, 어떻게 발생할지 모르기 때문에 소프트웨어 신뢰성은 즉각적인 고장 발생에 대한 고민과 모수적인 관점이 아닌 비모수적 관점에 대한 논의가 꾸준히 이뤄지고 있다. Miyamoto 외(2022)는 오픈소스 소프트웨어를 통해 발생한 소프트웨어 고장에 관한 신뢰성을 심층신경망을 활용한 소프트웨어 신뢰성 모형을 제안하였다. 또한, Oveisi 외(2021), Raamesh 외(2022)는 소프트웨어의 고장 자체가 순차적 데이터 특성임을 이용하여 LSTM 딥러닝 기법을 활용한 소프트웨어 신뢰성 모형을 제안하였다. Kim 외(2023a)는 기존에 알려진 딥러닝 기법인 심층신경망, 순환신경망, 장단기 메모리, 게이트 순환 유닛을 활용하여 데이터에 의존하는 딥러닝을 활용한 소프트웨어 신뢰성 모형을 제안하였다. Wu 외(2021)는 소프트웨어 신뢰성 모형의 가중치를 심층신경망의 가중치를 통해 학습하는 구조를 제시하였으며, Batool와 Khan(2022)은 딥러닝 만이 아닌 머신러닝의 여러 기법까지 종합하여 소프트웨어 오류 및 결함을 예측하는 모형을 제안하

였다. Kim 외(2023b)는 기존 딥러닝 기법에서 활용되는 활성화함수를 소프트웨어 신뢰성 모형을 구성하는 고장검출률 함수인 $b(t)$를 활용하여 단순히 데이터에 의존하는 모형이 아닌 수학적 가정이 추가된 딥러닝 소프트웨어 신뢰성 모형을 제안하였다.

NHPP를 가정한 소프트웨어 신뢰성 모형에 관한 연구가 아닌 이외의 다른 방법을 기반으로 한 소프트웨어 신뢰성 연구도 시도되고 있다. Kim 외(2009), Inoue와 Yamada(2017)는 개발된 소프트웨어 신뢰성 모형에 포함된 여러 가지의 모수를 베이지안 기법을 기반으로 각 모수에 사전분포를 두고 새로운 소프트웨어 신뢰성 모형을 제안하였다. 최근에는 신뢰성 예측을 위해 가장 최적의 모형을 찾는 것이 중요한 관심사이다. 이를 활용하기 위해 퍼지이론(Fuzzy Theory)을 기반으로 많은 연구가 시행되고 있다. 퍼지이론은 모호한 대상을 다루는 논리로, 모호한 정도를 조절할 수 있는 집합에 대한 이론이다. 최적의 모형을 찾기 위해 퍼지이론을 기반으로 많은 연구가 시행되고 있다. Ogundoyin과 Kamil(2020), Sahu 외(2021), Rafi 외(2022)는 계층화 분석과정(Analytic hierarchy process; AHP)와 Hesitant 퍼지 집합(Hesitant fuzzy sets; HF), TOPSIS(Techniques for order of preference by similarity to ideal solution)의 결과를 조합하여 가장 최적의 신뢰성 모형을 찾았다.

NHPP나 비모수적 소프트웨어 신뢰성 모형이 아닌 다른 소프트웨어 신뢰성 모형에 관한 연구도 꾸준히 이뤄지고 있으며, 소프트웨어 신뢰성 모형을 기반으로 한 비용모형, 민감도 분석 등을 통해 다양한 소프트웨어 신뢰성에 관한 연구가 진행되고 있다.

제 2 절 연구 내용 및 방법

본 논문에서는 시대가 발전하면서 복잡해진 소프트웨어의 신뢰성 평가를 위한 새로운 소프트웨어 신뢰성 모형을 제안하고자 한다. 과거 NHPP 소프트웨어 신뢰성 모형은 소프트웨어 고장이 독립적으로 발생함을 가정하여 개발하였다. 하지만 소프트웨어가 발전하면서 복잡해지고, 여러 소프트웨어로 구성되면서부터 고장이 독립적으로 발생하는 것과 더불어 소프트웨어 고장이 다른 고장에 영향을 미치는 경우가 나타났다. 따라서 고장이 종속적으로 발생함을 가정한 새로운 소프트웨어 신뢰성 모형을 제안하고자 한다.

소프트웨어의 구조가 복잡해지고, 여러 소프트웨어가 함께 구성되면서 소프트웨어의 환경이 매우 다양해졌다. 소프트웨어는 각기 다른 환경에서 쓰이기 때문에 복잡해진 현재의 소프트웨어 구조에 맞는 가정을 찾기란 쉽지 않다. 또한, 기존 많은 소프트웨어 신뢰성 모형은 수학적인 여러 가정을 추가하였기 때문에 특수한 경우에만 잘 적합 하는 문제를 가지고 있다. 이는 특수한 경우가 아닌 일반적인 경우에 잘 적합 시킬 수 없는 문제를 함께 지니고 있어 이를 해결하기 위해 주어진 고장 데이터에 의존하는 형태인 기계학습 방법 중 딥러닝을 활용한 소프트웨어 신뢰성 모형을 제안하고자 한다. 이는 데이터에 의존하는 모형이기에 제안하는 소프트웨어 신뢰성 모형을 기반으로 특수한 경우의 소프트웨어 고장에 의존하지 않고, 소프트웨어에서 발생하는 고장을 모두 고려한 일반화된 소프트웨어 신뢰성 모형을 개발할 수 있다.

또한, 데이터에만 의존하는 모형은 수학적·통계적 가정을 하지 않아 논리적인 근거가 부족하다. 오롯이 데이터에만 의존해야 하므로 얻은 결과를 논리적 근거 없이 믿고 따를 수밖에 없다. 이러한 문제를 해결하고자 딥러닝에서 쓰이는 활성화함수를 소프트웨어 신뢰성 모형으로 대체한 딥러닝 소프트웨어 신뢰성 모형을 제안하고자 한다. 딥러닝에서 활용되는 활성화함수 중 시그모이드 함수는 소프트웨어 신뢰성 모형의 가장 일반적인 구조이다. 해당 형태를 기존의 소프트웨어 신뢰성 모형으로 대체하면 데이터에만 의존하는 모형이 아닌 수학적 가정이 추가된 활성화함수로서 기능하므로 소프트웨어 신뢰성 모형을 활성화함수로 하는 딥러닝 소프트웨어 신뢰성 모형을 제안하고자 한다.

총 세 가지의 제안하고자 하는 모형은 9개의 추정값을 기반으로 한 적합도 척도와 1개의 예측값에 대한 적합도 척도를 통해 모형의 우수성을 입증하고자 한다. 또

한, 종속 고장을 고려한 새로운 소프트웨어 신뢰성 모형은 추정한 고장 수를 기반으로 설치 비용, 테스트 비용 등 소프트웨어 개발 및 출시에 포함될 비용들을 고려하여 최적의 출시시점을 제안한다. 딥러닝을 활용한 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형은 전체 데이터 세트 중 90% 데이터를 통해 각각의 모수를 추정 및 학습하고, 남은 10% 데이터를 통해 예측값을 계산하여 실제값이 예측값을 기준으로 한 95% 신뢰구간 안에 포함되는지를 비교하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 소프트웨어 신뢰성에 대한 포괄적인 내용을 다룬다. 소프트웨어 신뢰성에 앞서 더 큰 범주인 신뢰성에 대한 이론과 신뢰성에서 쓰이는 여러 분포함수에 대한 소개와 함께 소프트웨어 신뢰성의 여러 모형을 소개한다. 3장에서는 딥러닝에 대한 이론적 배경을 소개한다. 4장은 2장, 3장에서 소개한 이론을 기반으로 제안하고자 하는 세 가지 모형을 소개하며, 5장에서는 기존에 개발된 NHPP 소프트웨어 신뢰성 모형과 제안하고자 하는 모형 간의 비교를 통해 제안하고자 하는 모형의 우수성을 입증하고자 한다. 마지막으로 6장에서는 결론 및 제언으로 마무리한다.

제2장 소프트웨어 신뢰성

제1절 신뢰성

1. 신뢰성 개념 및 역사

시스템이 복잡해지고 다양해지면서 신뢰성은 매우 중요한 개념으로 자리 잡았다. 시스템 이용자는 성능, 외관, 견고성, 수리 용이성 등 하는 여러 가지 요구 조건(Requirements)을 충족(Satisfaction)하는 것을 비교하며 선택한다. 이 모든 것을 직접 사용해보지 않는다면 정량적으로 비교하기란 쉽지 않다. 따라서 “얼마나 고장 나지 않고 오랫동안 사용할 수 있는가”라는 신뢰성을 수치화 정량화하여 시스템을 비교하여 올바른 선택 및 결정을 하는데 필요한 수단으로 활용한다. 신뢰성이란 제품이 주어진 사용 조건하에서 의도하는 기간 동안 정해진 기능을 성공적으로 수행하는(고장을 일으키지 않는) 능력을 의미한다. 국가 표준(Korea Standard; KS)에서는 시스템, 기기, 부품 등의 시간적 안정성을 나타내는 정도 또는 성질로 정의 내렸으며, 전자기기 신뢰성 자문 위원회(Advisory group on reliability of electronic equipment; AGREE)는 부분품, 장치기기 또는 시스템이 주어진 어떤 조건 아래서 일정 기간 중에 의도했던 기능을 수행하는 확률로 정의하였다. 또한, 신뢰성은 고유 신뢰성과 사용 신뢰성 두 가지 개념으로 나눠서 정의하기도 한다. 고유 신뢰성은 제품의 설계, 제조 및 시험 등의 과정을 거쳐서 형성되는 신뢰성으로 제조된 제품의 특성을 나타내게 되며 수명자료를 이용한 통계적 분석을 통하여 계량적으로 표시될 수 있는 고유성질이다. 사용 신뢰성은 제품의 사용(가동)상태에서의 제품의 신뢰성으로 예방보전이나 고장 발생 시의 수리 등을 포함한 사용의 용이성과 애프터서비스 등도 고려하는 광범위한 의미의 신뢰성을 의미한다(정해성 외, 2003, Liu와 Rho, 2013).

신뢰성은 고장 나지 않고 오랫동안 사용하고자 하는 욕구에서부터 시작되었으며 이는 매우 중요한 지표로 활용된다. 제품의 수명 관련 품질에 관한 추정을 위해, 새로운 고장의 원인이 무엇인지 알아내기 위해, 새로운 설계나 제조 공정의 비교 및 평가를 위해 신뢰성 분석이 이뤄지고 있으며 좋은 공정 과정이 이뤄지고 있는지 또는 좋은 제품을 판단하기 위해 신뢰성은 매우 중요하다. 예를 들어, 자주 여닫는 냉장고가 몇 년 동안 고장 없이 작동할 확률이나, 미사일과 같이 자주 사용하지 않지만 단 한 번의 작동이 실패하지 않고 정상 작동할 확률 등을 나타낼 때, 신

뢰성으로 판단한다. 신뢰성 향상에 관한 연구가 꾸준히 이뤄지고, 발전하게 되면 그만큼 제품에 대한 안정성, 가용성, 보안성 등이 함께 향상될 것이다. 따라서 신뢰성 향상에 관한 연구가 활발히 이뤄져야 하며 안정성, 가용성, 보안성 등과 함께 제품을 판단하는데 여러 정량적 근거가 함께 제시되는 연구가 진행되고 있다 (Sommerville, 2001).

신뢰성에 관한 연구는 1930년대 제품의 부품이나 시스템의 수명에 관한 연구를 진행하면서부터 시작되었으며 제2차 세계대전을 계기로 많이 발전했다. 1940년대에는 전쟁에 사용하기 위하여 만들어진 진공관으로부터 시작되었다. 제2차 세계대전 중에 극동 전략용인 군용기와 항공 통신기, 레이더 등 중요한 전자 장비에 이 진공관이 사용되었는데, 수송과 보관 시에 진공관의 고장으로 반 이상 무용지물이 되면서 고장 나는 무기를 구입하지 않겠다는 방침에 따라 신뢰성에 관한 연구가 착수되었다(정해성 외, 2003).

신뢰성이 발전하게 된 가장 결정적인 계기는 Sobel과 Epstein(1954)이 제안한 지수분포의 개발이었다. 지수분포는 고장률 계산이 쉬우며 설계 단계에서 데이터를 분석할 때, 지수분포로 표현하여 매우 단순화시키는 데 기인하였고, 수명자료를 분석하거나 가속 수명 설계 등 기본적인 수명분포로써 활용되기 때문에 신뢰성 이론에 매우 큰 영향을 미쳤다. 하지만 지수분포만으로 모든 현상을 표현하여 신뢰성을 판단할 수 없기 때문에 또 다른 분포인 감마분포, 와이블분포, 정규분포 등이 제안되었다. 전자기기 신뢰성 자문 위원회가 만들어진 이후 여러 연구기관에서 신뢰성 분야에 관한 연구 및 응용이 적극적으로 이루어졌다. 제품에 대한 신뢰성 추정, 신뢰성 및 가용도의 최적화, 고장 유형에 따른 시스템 영향 분석, 보전정책, 고장 자료 수집 및 분석, 네트워크의 신뢰성 계산 등 매우 광범위한 분야로 발전되었고, 신뢰성 기술은 공학, 의학 등 다양한 분야에서 많이 활용되고 있으며 특히, 하드웨어, 소프트웨어가 포함된 컴퓨터공학 분야에서 활발히 연구가 진행되고 있다(박동호 외, 2015).

2. 신뢰성의 분포함수

신뢰성 연구에 중요한 이론적 분야는 시스템 또는 제품의 노화(Aging)에 따른 성능 변화를 수치화하여 측정할 수 있도록 여러 형태의 함수를 정의하고 시간의 흐름에 따라 각 함수가 어떻게 변화하는지에 대한 성질을 규명하는 것이다. 이를 설명하기 위해 고장 시간이나 수명시간을 확률변수 T로 가정하여 확률밀도함수 $f(t)$를 통해 확률변수 T의 변화에 따라 고장의 수, 고장확률 등을 예측 및 추정한다. 확률밀도함수 $f(t)$와 누적분포함수 $F(t)$를 통해 특정 시간 t 이전에 고장 날 확률을 의미하는 수명분포함수를 정의하면 다음과 같다.

$$F(t)=\Pr[T<t]=\int_0^t f(s)ds$$

신뢰도함수 $R(t)$는 위의 수명분포함수 $F(t)$를 통해 시스템이 특정 시간 t보다 오랫동안 즉, t시간 이후에도 작동할 확률로 정의되며 수명분포함수 $F(t)$와는 반대되는 개념으로 다음과 같은 수식으로 표현된다.

$$\begin{aligned} R(t)=\Pr[T>t] &= \int_t^\infty f(s)ds \\ &= 1-\int_0^t f(s)ds \\ &= 1-F(t) \end{aligned}$$

순간고장률을 나타내는 고장강도함수 $\lambda(t)$는 위험률 함수라고도 하며, 이는 시점 t에서 순간 고장 날 수 있는 함수로 단위 시간당 조건부 평균 고장 비율(순간고장률)로 정의한다. 이를 통해 시점마다 변화하는 순간 고장률을 판단하여 추후에 발생할 고장 수를 판단할 수 있다. 고장강도함수 $\lambda(t)$는 다음과 같이 유도하여 표현한다.

$$\begin{aligned}\lim_{\Delta t\to 0}\frac{\Pr(t<T\le t+\Delta t|T>t)}{\Delta t} &= \lim_{\Delta t\to 0}\frac{\Pr(t<T\le t+\Delta t)}{\Pr(T>t)\cdot\Delta t}\\ &= \lim_{\Delta t\to 0}\frac{R(t)-R(t+\Delta t)}{R(t)\cdot\Delta t}\\ &= \frac{1}{R(t)}\lim_{\Delta t\to 0}\frac{R(t)-R(t+\Delta t)}{\Delta t}\\ &= \frac{1}{R(t)}(-R'(t))\\ &= \frac{f(t)}{R(t)}\\ &= \lambda(t)\end{aligned}$$

신뢰도함수 $R(t)$와 고장강도함수 $\lambda(t)$와의 관계는 일대일 대응 관계가 성립된다. 해당 관계는 신뢰도함수 $R(t)$에 대한 추정이 가능하면 고장강도함수 $\lambda(t)$에 대한 추정도 가능하게 되며 다음과 같은 식을 따르게 된다. 이러한 관계를 역공식 (Inversion formula)이라고 하며, 신뢰성 이론에서 매우 중요한 역할을 한다.

$$\lambda(t)=\frac{f(t)}{R(t)}=\frac{-R'(t)}{R(t)}$$

$$\int_0^t\lambda(s)ds=-\int_0^t\frac{R'(s)}{R(s)}ds=-\ln R(t)$$

$$R(t)=\exp\left\{-\int_0^t\lambda(s)ds\right\}$$

시스템 및 제품의 고장은 원인 또는 고장 시점에 따라 세 가지 경우로 나뉜다. 첫 번째, 제품에 내재하고 있는 설계나 공정에서의 결함에 의한 것으로 그 결함을 찾아내어 안정화 시킬 필요가 있는 시기로 설계 오류나 제조 오류 등의 원인으로 발생한다. 두 번째는 우발 고장으로 외부로부터 발생하는 부하로 생긴 고장으로 과중한 부하, 사용자의 과실 등의 이유로 발생한다. 세 번째는 마모고장으로 시간이 많이 흐르면서 발생하는 노화, 부식과 같은 고장이다. 이를 기준으로 총 5가지 형태의 수명주기에 따른 형태가 존재한다. 사용하는 제품 또는 시스템의 고장이 어느 위치에 발생하는지, 또는 어떠한 유형으로 고장이 발생하는지를 염두에 둔다면 보다 쉽게 대

처하여 큰 손실을 막을 수 있을 것이다(Wong, 1989; Hamada 외, 2008).

1. $\lambda(t)$가 증가하는 경우: 시간의 흐름에 따라 고장강도함수가 증가하며 전체적인 누적 고장 수의 증가율이 1보다 큰 형태(Increasing failure rate; IFR)이다.
2. $\lambda(t)$가 감소하는 경우: 시간의 흐름에 따라 고장강도함수가 감소하며 전체적인 누적 고장 수의 증가율이 1보다 작은 형태(Decreasing failure rate; DFR)이다.
3. $\lambda(t)$가 일정한 경우: 시간의 흐름에 따라 고장강도함수가 일정하며 전체적인 누적 고장 수의 증가율이 1인 형태(Constant failure rate; CFR)이다.
4. $\lambda(t)$가 욕조형 구조로 발생하는 경우: 초기에는 고장강도함수가 높다가 시간의 흐름에 따라 감소하며 다시 증가한 형태(Bathtub failure rate; BFR)이다.
5. $\lambda(t)$가 롤러코스터형 구조로 발생하는 경우: 욕조형과 비슷한 구조를 가지지만 일정 고장률이 유지되는 구간에 이전에 발견되지 않은 고장이 발견될 수 있다는 것을 가정한 형태(Rollercoaster shaped failure rate; RFR)이다.

시스템 및 제품은 확률변수 T에 대한 수명분포를 가정한다. 여기서 수명분포는 통계적 이론을 근거한 분포로 지수분포, 정규분포, 감마분포 등이 있다. 각각의 분포는 위치모수(Location parameter), 척도모수(Scale parameter), 형상모수(Shape parameter) 총 세 가지 모수가 있으며 포함하는 모수에 따라 분포가 다르게 정의된다. 위치모수는 분포의 위치나 이동에 영향을 미치는 모수이며 대푯값에 해당하는 모수이다. 척도모수는 흩어진 정도를 의미하며 산포도가 이에 해당하는 모수이다. 척도모수가 클수록 넓게 퍼져있으며, 작을수록 밀집되어 있다. 형상모수는 분포 모양을 결정하는 모수이다. 본 논문에서는 신뢰성에서 사용되는 분포에 대한 모수, 확률밀도함수 $f(t)$, 누적분포함수 $F(t)$, 고장강도함수 $\lambda(t)$ 등을 소개하고자 한다.

가. 지수분포

신뢰성 이론에서 가장 중요한 분포는 바로 지수분포이며, 이는 신뢰성 분야에서 통계학에서의 정규분포와 같이 많이 활용되는 분포이다. 지수분포는 포아송 분포와 매우 연관되어 있다(Davis, 1952; Barlow와 Proschan, 1965). 포아송 과정의 가정에 따라 무작위로 제품에 고장이 발생한다고 가정한다. 길이 t의 시간 간격에서 발생하는 고장 수 $X(t)$는 포아송 분포로 설명할 수 있으며, 여기서 λ는 고장 발생률을, 확률변수 T는 제품의 고장 시간을 의미한다.

$$\Pr(X(t)=n)=\frac{e^{-\lambda t}(\lambda t)^n}{n!},\ \lambda,t>0,\ n=0,1,2,\cdots$$

위의 가정하에 신뢰도함수 $R(t)$는 시점 t까지 고장 나지 않을 확률을 의미한다. 따라서 신뢰도함수 $R(t)$는 확률밀도함수 $f(t)$를 적분하여 계산하며 다음과 같다.

$$R(t)=e^{-\lambda t},\ F(t)=1-e^{-\lambda t}$$

확률밀도함수 $f(t)$는 신뢰도함수 $R(t)$를 미분하여 유도할 수 있으며 다음과 같이 표현된다.

$$f(t)=-\frac{dR(t)}{dt}=\lambda e^{-\lambda t},\ t\geq 0,\lambda>0$$

고장강도함수 $\lambda(t)$는 확률밀도함수 $f(t)$와 신뢰도함수 $R(t)$의 미분식을 이용하여 유도할 수 있으며 다음과 같이 표현된다.

$$\lambda(t)=\frac{-R'(t)}{R(t)}=\frac{f(t)}{R(t)}=\frac{\lambda e^{-\lambda t}}{e^{-\lambda t}}=\lambda$$

지수분포의 평균과 분산은 $E(T)=\frac{1}{\lambda}$, $Var(T)=\frac{1}{\lambda^2}$로 나타난다.

지수분포는 무기억성(Memoryless property)을 가지고 있는 유일한 연속확률분포이기 때문으로 알려져 있다. 지수분포의 무기억성은 아래와 같은 식을 따른다.

$$\Pr(T > s+t \mid T > t) = \Pr(T > s)$$

이는 확률변수 T가 특정 제품이 고장 날 때까지 걸리는 시간으로 가정하면 특정 제품이 시점 t까지 작동했다는 조건 하에 시점 $t+s$ 이후에도 잘 작동할 확률과 시점 s 이후까지 잘 작동할 확률이 같다는 것을 의미한다. 즉, 특정 제품이 출시되어 사용될 때, s시간 이후에 고장 날 확률과 t시간 동안 잘 작동한 다음, 그로부터 s시간 이후에 고장 날 확률은 같다는 것을 의미한다. 위와 같은 무기억성을 따르기 때문에 시간이 지나더라도 고장 날 확률이 증가하거나 감소하지 않고, 일정하게 유지됨을 가정하며 고장강도함수 $\lambda(t)$가 상수와 같은 CFR 형태를 가진다.

나. 와이블분포

와이블 분포는 스웨덴의 물리학자인 와이블(Weibull)로부터 1939년에 개발된 확률밀도함수로 신뢰성에서는 Weibull(1951)에 의해 논의되었다. 이는 지수분포를 일반화한 모형으로 일정하지 않은 고장강도함수를 포함한다. 특히, 와이블 분포는 고장강도함수의 증가와 감소를 모두 포함하며, 초기 고장(Alven, 1964)과 마모 고장(Lieblein과 Zelen, 1956)을 함께 설명하는데 성공적으로 활용되었다. 와이블 분포는 위치모수인 γ, 형상모수인 m, 척도모수인 η 총 3가지 모수를 포함한 확률밀도함수 $f(t)$로 다음과 같으며, 이에 따른 평균과 분산도 다음과 같다.

$$f(t) = \frac{m}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{m-1} e^{-\left(\frac{t-\gamma}{\eta}\right)^{m}}, \quad t \geq 0,$$

$$E(T) = \gamma + \eta\Gamma\left(\frac{m+1}{m}\right),$$

$$Var(T) = \eta^2\left[\Gamma\left(\frac{m+2}{m}\right) - \left(\Gamma\left(\frac{m+1}{m}\right)\right)^2\right]$$

일반적으로는 $\gamma = 0$을 가정하며 형상모수 m과 척도모수 η로 이뤄진 와이블 분포를 많이 활용하며 $m = 1$이면 와이블 분포는 고장률이 상수인 지수분포를 따르고, $m < 1$이면 감소함수인 감마분포, $m > 1$이면 증가함수를 따른다. $m = 2$인 경우는

고장강도함수 $\lambda(t)$의 기울기가 $2/\eta^2$인 선형 증가함수 형태인 레일리 분포를 따른다. 와이블분포는 모수에 따라 매우 다양한 형태를 띠고 있기 때문에 중요한 형식은 없다. 누적분포함수 $F(t)$, 고장강도함수 $\lambda(t)$, 신뢰도함수 $R(t)$는 다음과 같다.

$$R(t)=e^{-\left(\frac{t-\gamma}{\eta}\right)^m},$$

$$F(t)=1-e^{-\left(\frac{t-\gamma}{\eta}\right)^m},$$

$$\lambda(t)=\frac{m}{\eta}\left(\frac{t-\gamma}{\eta}\right)^{m-1}$$

다. 감마분포

감마분포는 지수분포의 확장된 형태로, 포아송 과정에서 n번째 사건이 발생할 때까지의 시간을 고려한 확률분포이다(Mann 외, 1974). 고장 사이의 시간 T가 지수분포를 갖는 경우, n번째 고장까지의 누적 시간인 $T=t_1+t_2+\cdots+t_n$은 척도모수 α와 형상모수 β를 갖는 감마분포를 따른다. 감마분포의 확률밀도함수 $f(t)$는 다음과 같으며, 평균과 분산은 $E(T)=\alpha\beta$, $Var(T)=\alpha\beta^2$이다.

$$f(t)=\frac{1}{\Gamma(\alpha)\beta^{\alpha}}t^{\alpha-1}e^{-\frac{1}{\beta}},\ t>0,\ \alpha,\beta>0$$

감마분포는 척도모수 α에 따라 형태가 결정되며, 감마분포는 모수 α가 1일 때는 지수분포와 동일한 확률밀도함수 $f(t)$를 가지고, 상수인 CFR을 따른다. $0<\alpha<1$인 경우는 DFR, $\alpha>1$인 경우는 IFR 형태를 가진다. 이에 대한 누적분포함수 $F(t)$, 고장강도함수 $\lambda(t)$, 신뢰도함수 $R(t)$는 다음과 같다.

$$R(t)=\sum_{k=0}^{\alpha-1}\frac{\left(\frac{t}{\beta}\right)^k e^{-\frac{t}{\beta}}}{k!},$$

$$F(t)=1-\sum_{k=0}^{\alpha-1}\frac{\left(\frac{t}{\beta}\right)^k e^{-\frac{t}{\beta}}}{k!},$$

$$\lambda(t)=\frac{t^{\alpha-1}e^{-\frac{t}{\beta}}}{\beta^{\alpha}\left[\Gamma(\alpha)-\Gamma\left(\alpha,\frac{t}{\beta}\right)\right]}$$

라. 정규분포

정규분포는 통계학에서 가장 많이 활용되는 분포로서 평균을 기준으로 대칭인 종 모양 분포이다. 이는 표본의 크기가 크면 정규분포를 따른다는 중심극한정리에 의해 광범위하게 응용되는 분포이다. 정규분포는 $(-\infty,\infty)$의 범위를 가지지만 결함 및 고장 관련 데이터는 0 이하의 값을 취급할 수 없으므로 정규분포를 활용하기 위해선 평균 μ가 충분히 큰 양수로 두고, 표준편차 σ를 작은 수로 하여 0보다 작은 확률에 대해선 무시할 수 있게 구성한다. 그렇지 않다면 0 이하를 절단한 후 스케일링을 다시 해야 한다(Johnson과 Kotz, 1970). 정규분포의 확률밀도함수 $f(t)$는 다음과 같으며 평균과 분산은 $E(T)=\mu$, $Var(T)=\sigma^2$을 따른다.

$$f(t)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(t-\mu)^2}{2\sigma^2}},\ -\infty\le t\le\infty$$

특히, $\mu=0$, $\sigma^2=1$인 경우는 표준정규분포라 부르며 다음과 같은 확률밀도함수 $\Phi(t)$를 따른다.

$$\Phi(t)=\frac{1}{\sqrt{2\pi}}e^{-\frac{t^2}{2}},\ -\infty\le t\le\infty$$

누적분포함수 $F(t)$, 고장강도함수 $\lambda(t)$, 신뢰도함수 $R(t)$는 다음과 같다.

$$F(t)=\Phi\left(\frac{t-\mu}{\sigma}\right),$$

$$R(t)=1-\Phi\left(\frac{t-\mu}{\sigma}\right),$$

$$\lambda(t)=\frac{\left(\frac{1}{\sigma}\right)\Phi\left(\frac{t-\mu}{\sigma}\right)}{1-\Phi\left(\frac{t-\mu}{\sigma}\right)}$$

로그정규분포는 정규분포의 아래쪽 꼬리가 왜곡된 분포로 0 이하의 문제를 해결하기 위해 활용된 확률밀도함수로 한쪽으로 치우친 분포에 적합한 형태를 띤다. Mann 외(1974)는 피로도가 누적된 파손모형으로 로그 정규분포를 활용하였다. 위의 정규분포에서 $T=\ln T^*$로 변형하여 활용하며 이에 따른 확률밀도함수 $f(t^*)$는 다음과 같다.

$$f(t^*)=\frac{1}{\sqrt{2\pi(\sigma t^*)^2}}e^{-\frac{(\ln t^*-\mu)^2}{2\sigma^2}},\ t^*>0$$

제2절 소프트웨어 신뢰성

오늘날 전 세계 수많은 사람은 컴퓨터 시스템의 영향을 직·간접적으로 받는다. 컴퓨터는 항공 교통 관제, 원자로, 항공기, 실시간 센서 데이터 등 다양한 분야에서 사용되어 수많은 사람들에게 영향을 미친다. 컴퓨터 작업의 기능이 더욱 중요해지고, 더욱 복잡하고, 중요한 응용 프로그램의 크기와 복잡도가 증가함에 따라 다양하고 복잡한 운용환경에서 컴퓨터 시스템의 신뢰성을 정량화하고 예측하는 방법을 모색할 필요성이 커졌다(Pham, 2006).

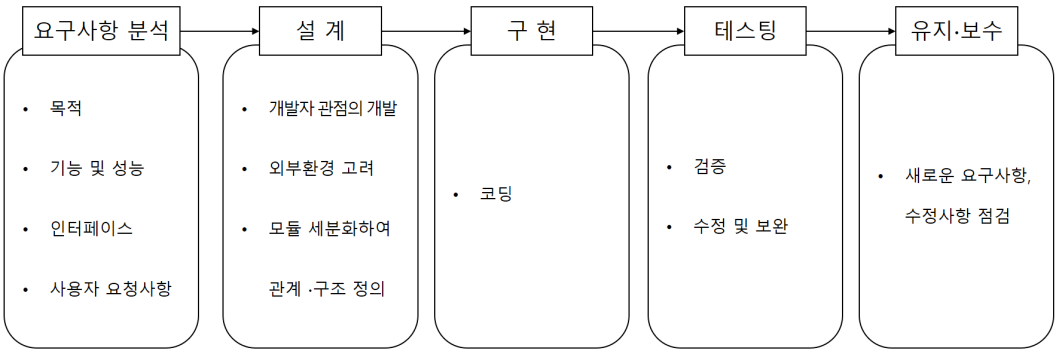
컴퓨터는 크게 하드웨어와 소프트웨어로 구성되어 있다. 하드웨어는 컴퓨터의 부품에 영향을 크게 받고, 대부분 성능은 컴퓨터 부품으로부터 이뤄진다. 소프트웨어는 부품으로 이뤄진 컴퓨터를 잘 작동시킬 수 있도록, 또 작동시킨 컴퓨터를 가장 효율적으로 사용할 수 있도록 만들어진 프로그램이다. 따라서 컴퓨터와 관련된 신뢰성에 관한 연구는 하드웨어와 소프트웨어로 나뉘어 연구가 진행되었고, 초창기에는 하드웨어 신뢰성에 관한 연구가 주를 이뤘다. 하지만 소프트웨어의 구성이 복잡해지고 활용도가 커지면서 소프트웨어 신뢰성에 관한 연구가 주를 이루고 있다. 하드웨어는 사용할수록 마모되고, 소진되며 고장률이 점점 증가하는 형태를 띠지만, 소프트웨어는 사용한다고 해서 마모되거나 고장률이 극적으로 증가하거나 감소하는 형태는 아니다. 즉 하드웨어의 고장은 물리적인 충격이나 스트레스, 노화가 주된 원인이라면 소프트웨어의 경우에는 설계 단계, 테스트 단계에서 발생한 결함이 소프트웨어 고장의 주요 원인이 되며 노화나 사용자의 실수 등은 고장의 원인이 되지 못한다. 또한, 하드웨어는 같은 제품이어도 신뢰성은 상호 독립적인 반면에 소프트웨어는 같은 버전이더라도 컴퓨터의 성능에 따라 소프트웨어의 성능은 매우 달라지는 종속적인 영향을 받기 때문에 소프트웨어 신뢰성을 예측하는 것은 매우 어려운 일이다. 따라서 소프트웨어 신뢰성에 관해 꾸준한 연구가 진행되고 있으며 많은 관련 소프트웨어 신뢰성 모형이 개발되고 있다(이성경, 1986).

소프트웨어 신뢰성은 제2장 제1절에서 소개한 전 분야에 걸친 신뢰성을 소프트웨어 분야로 함축한 분야로 소프트웨어가 주어진 사용 조건 하에서 의도하는 기간 동안 정해진 기능을 성공적으로 수행하는 능력을 의미한다. 소프트웨어가 다양한 분야에서 많이 활용될수록 소프트웨어 신뢰성은 매우 중요한 영역으로 성장했으며 소프트웨어 신뢰성을 예측하기 위해 많은 노력이 기울여졌다. 소프트웨어 신뢰성은 Jelinski와 Moranda(1972)로부터 소프트웨어 신뢰성 성장 모형에 관한 연구가 시작

되면서부터 1970년대에 활발히 연구가 이뤄졌다. 이러한 소프트웨어 신뢰성 성장 모형에 대한 연구는 모형과 모수 추정을 중심으로 하여 진행되고 있으며 이러한 연구 접근이 모두 확률적인 성질을 이용하여 추정과 예측을 하고 있다. 현재 소프트웨어 신뢰성에 관한 관심은 몇몇 기관에서 소프트웨어 인증 사업을 시작하면서 더욱 고조되었다. 특히 의료용 소프트웨어나 산업용 소프트웨어, 철도, 국방 관련 소프트웨어의 경우는 신뢰성이 상당히 중요한 요소이다. 그러나 신뢰성에 대한 평가는 상당히 이론적인 접근이 쉽지 않고, 측정 가능한 소프트웨어 고장에 관련된 데이터 수집이 쉽지 않아서 측정에 많은 어려움이 있다. IT 강국들은 우리보다 먼저 소프트웨어 품질에 상당한 관심을 가지고 신뢰성 향상을 위해서 품질을 평가하는데 많은 관심을 가지고 연구하였다. 우리나라에서도 한국정보통신기술협회나 산업자원부 기술표준원 등에서 소프트웨어의 품질 인증을 시작하여 제품의 품질 향상을 위한 노력에 최대의 힘을 기울이고 있다(Jung, 2006, Seo와 Kim, 2008).

소프트웨어는 시스템 소프트웨어, 실시간 소프트웨어, 비즈니스 소프트웨어, 공학 및 과학 소프트웨어, 임베디드 소프트웨어, 인공지능 소프트웨어 등 다양한 분야에 접목시킨 소프트웨어가 존재하며 매우 복잡한 형태로 개발된다. 이러한 복잡한 소프트웨어의 개발과정은 요구사항 분석, 설계, 구현, 테스팅, 유지보수 총 5단계로 구분된다. 요구사항 분석 단계에서는 해당 소프트웨어를 활용하고자 하는 사용자로부터 소프트웨어 개발 목적, 기능 및 성능, 인터페이스 요구 등 요구사항을 반영하는 단계이다. 요구사항을 사용자로부터 규명하는데 많은 노력과 시간이 필요하지만 추후에 개발 과정에서 발생할 고장에 대한 비용에 비해선 상당히 절약할 수 있는 단계이기 때문에 탄탄한 소프트웨어를 개발하기 위한 목적을 정확하게 하기 위해 매우 중요한 단계이다. 설계 단계는 요구사항 분석 단계에 수집한 여러 가지의 요구사항을 개발자의 관점에서 어떻게 반영하고 구현하는지를 설계하는 단계이다. 비용이나 개발하는데 필요한 시간, 신뢰성 등 많은 연구가 동반되며, 주어진 외부 상황을 모두 고려한 최적의 설계 방법을 선택해야 한다. 또한, 전체 시스템을 독립성이 유지될 수 있도록 여러 개의 모듈로 세분화하여 각 모듈 간의 관계와 구조를 정의하며 이를 기반으로 설계도를 작성한다. 구현 단계에서는 설계 단계에서 작성한 설계도를 프로그램 언어를 통해 코딩하는 과정이다. 구현하는 과정은 설계 단계에서 설계된 대로 기계적인 연산 과정으로 이뤄지기 때문에 잘못된 오류가 발생할 시, 대부분은 설계 단계에서 발생한 문제일 확률이 높다. 테스팅 단계에서는 구현된 소프트웨어가 설계대로 수행하는지를 검증하기 위한 단계이며 검증 단계에

서 발생한 오류를 수정 및 보완하는 단계이다. 테스팅은 품질보증에 대해 중요한 부분이며 앞선 단계에서 이뤄진 전 과정에 대한 최종 점검을 포함하고 있다. 마지막 유지보수 단계에서는 소프트웨어 작동 중 새로운 요구사항이나 수정 사항을 반영하며 수정 및 보완된 소프트웨어를 앞선 단계의 모든 과정을 재수행한다. 소프트웨어 고장은 테스팅 단계에서 발생하며, 꾸준한 모니터링을 통해 발견하고 관리한다. 발생하는 고장의 원인을 찾아 유지보수 단계에서 코드 수정 및 보완 과정을 거친다. 소프트웨어 신뢰성을 잘 관리한다면 유지보수 단계의 시행착오를 상당 부분 줄일 수 있다. 수정 및 보완이 완료된 소프트웨어는 출품 및 판매 단계를 거쳐 출시된다(Singpurwalla와 Wilson, 2012). <그림 2-1>은 소프트웨어 개발 단계에 대한 설명이다.



<그림 2-1> 소프트웨어 개발 단계

소프트웨어 신뢰성은 설계, 구현, 테스팅, 유지보수 등 개발과정에서 발생하는 소프트웨어 고장을 기반으로 판단하며 평가한다. 소프트웨어 신뢰성은 소프트웨어가 고장 나지 않고 얼마나 오래 사용할 수 있는가를 판단하기 때문에 출시 전까지 위의 과정을 반복하고 발생하는 고장을 관리·감독하며, 수정 및 보완 과정을 거쳐야 한다. 해당 과정을 거치면서 결함 없는 소프트웨어를 개발하는 것이 중요한 문제로 자리매김하고 있다. 만약 개발과정을 잘 거쳐서 결함 없는 소프트웨어를 출시하였다 하더라도 실제 운용환경은 테스트 환경과 다를 수 있기 때문에 이를 통해 즉각적으로 대처할 수 없는 결함이나 고장이 발생한다면 막대한 재정적 손실뿐만 아니라 소프트웨어를 출시한 기업의 이미지, 경영 등 다른 부분까지도 영향을 미칠 수 있다. 따라서 소프트웨어 고장을 예측하는 소프트웨어 신뢰성 모형에 관한 연구

가 지속해서 이루어지고 있다. 고장이나 결함을 잘 예측한다면 소프트웨어 신뢰성을 향상시키는 것뿐만 아니라 즉각적인 대응을 통해 여러 손실을 막을 수 있을 것이다. 또한, 예측 고장 수를 통해 소프트웨어 개발 및 테스트 단계에 투입되는 비용을 최소화하는 출시시점을 결정할 수 있다.

1. 소프트웨어 신뢰성 모형

소프트웨어는 모든 분야에서 아주 작은 부분부터 시작해서 전체를 관장하는 부분까지를 아우르고 있기 때문에 소프트웨어 고장이 발생한다면 큰 사회·경제적 문제를 야기 시킬 수 있다. 따라서 소프트웨어 고장에 대한 원인에 맞춰 소프트웨어를 수정 및 보완하여 개선해나가야 한다. 소프트웨어 신뢰성 모형은 이를 관리하여 소프트웨어 신뢰성을 높이기 위한 목적으로 개발한다. 여러 수학적·통계적 가정에 의해 소프트웨어 고장을 예측 및 추정하여 사전에 발생할 수 있는 문제를 차단하고자 한다.

소프트웨어 신뢰성 모형은 크게 모수적 방법과 비모수적 방법으로 개발된다. 모수적 방법은 수학적·통계적 가정을 기반으로 한 모수를 추정하여 소프트웨어 고장수를 예측한다. 이는 수학적·통계적 가정이 포함되기 때문에 논리적인 근거를 포함하고 있으며 종속 고장, 운용환경의 불확실성 등 복잡해지는 소프트웨어의 실제 상황을 고려한 모형 개발 방법이다. 비모수적 방법은 통계적 분포를 가정하지 않거나 모수에 대한 가정을 하지 않는 분포 무관인 방법으로 얻어진 데이터에 의존하여 논리적 근거를 데이터에서 얻으며, 이를 모형 개발에 활용한다.

먼저 모수적 방법을 통해 개발되는 소프트웨어 신뢰성 모형은 결정론적 방법과 확률론적 방법 두 가지로 나뉜다. 결정론적 방법은 프로그램의 고유 연산자 및 피연산자 수와 오류 수, 프로그램 안에 포함된 명령어 수 등을 연구하는 데 활용되며 대표적으로는 Halstead의 소프트웨어 매트릭스(Halstead, 1977)와 McCabe의 순환적 복잡성 매트릭스(McCabe, 1976)가 존재한다. 반면 확률론적 방법은 결함 발생과 결함 제거를 확률적 접근을 통해 문제를 해결한다. 오류 유입 모형, 고장률 모형, 곡선 적합 모형, 시계열 모형 등은 확률론적 모형의 대표적 예이다. 오류 유입 모형은 다단추출법(Multi-stage sampling technique)을 통해 소프트웨어 프로그램의 오류 수를 추정하는 모형으로 시스템 소프트웨어 테스트 단계 이후 잔여 오류의 양을 평가하는 데 사용할 수 있다. 오류 유입 모형은 Mills의 오류 시딩 모

형(Mills, 1972), Cai의 오류 시딩 모형(Cai, 1998), Hypergeometric 분포 모형(Tohma 외, 1991) 등이 있다. 여기서 시딩(Seeding)은 씨 뿌리는 의미를 내포하고 있으며 기존 코드에 시드 오류를 삽입하여 발견된 시드 오류 수를 기반으로 해당 소프트웨어의 실제 오류 수를 추정할 수 있다. Mills의 오류 시딩 모형은 소프트웨어에 시딩된 오류를 도입하여 소프트웨어의 오류 수를 추정하는 오류 시딩 방법을 제안하였다. 내재 오류와 유도 오류로 구성된 디버깅 데이터로부터 미지의 내재 오류 수를 추정할 수 있다. 내재 오류와 유도 오류가 모두 감지될 확률이 동일하다면, 제거된 오류 r에서 k개의 유도 오류가 발생할 확률은 다음과 같은 초기하분포를 따른다.

$$P(k;N,n_1,r)=\frac{{}_{n_1}C_k \cdot {}_{N}C_{r-k}}{{}_{N+n_1}C_r}$$

여기서 N은 총 내재 오류 수, n_1은 총 유도 오류 수, r은 디버깅 중에 제거된 총 오류 수, k는 제거된 오류 r에서 유도된 오류의 총 개수, $r-k$는 제거된 오류 r에서 내재 오류의 총 개수이며 이를 통해 $\hat{N}$은 최소정수함수(Least integer function)를 활용하여 $\hat{N}=\lfloor N_0 \rfloor+1$로 추정한다. N_0는 $\frac{n_1(r-k)}{k}-1$이다.

Cai의 오류 시딩 모형과 Hypergeometric 분포 모형은 Mills의 오류 시딩 모형으로부터 개발된 모형으로 Cai의 오류 시딩 모형은 소프트웨어에 남아있는 결함 수를 추정하는 데에 활용하였고, Hypergeometric 분포 모형은 초기하 분포를 기반으로 테스트 또는 디버깅 과정이 시작할 때, 프로그램에 초기에 발생하는 결함 수를 추정하는 데에 활용하였다.

고장률 모형은 고장 간격 동안 고장 시점에 고장률이 어떻게 변하는지에 대한 모형으로, 남아있는 결함 수의 변화에 따라 고장률도 함께 변하는 형태이다. 소프트웨어의 프로그램 결함 수는 셀 수 있는 이산형 변수(Discrete variable)이며 이산 확률변수를 갖는 확률질량함수(Probability mass function)를 가정한 여러 형태의 고장률 모형이 제안된다. 대표적인 모형으로는 Jelinski-Moranda(J-M) 모형(Jelinski와 Moranda, 1972)이 있다. J-M 모형은 아래와 같은 가정을 따른다.

1. 소프트웨어에는 잘 알려지지 않은 고정된 상수인 N개의 초기 결함이 포함되어 있다.
2. 소프트웨어의 각 결함은 독립적이며, 테스트 단계에서 결함을 일으킬 확률은 동일하다.
3. 고장 사이에 발생하는 시간 간격은 독립이다.
4. 발생한 고장은 확실히 제거되며, 고장을 유발하는 결함은 즉시 제거된다.
5. 결함을 제거하는 동안 새로운 고장은 발생하지 않는다.
6. 소프트웨어의 고장률은 하나의 상수로 일정하며, 소프트웨어의 잔여 결함 수에 비례한다.

이를 통해 i번째 고장 간격의 소프트웨어 고장률은 $\lambda(t_i)=\phi[N-(i-1)]$로 나타난다. i번째 고장 간격의 소프트웨어 고장률을 통해 계산하는 확률밀도함수와 누적분포함수, 신뢰도함수는 다음과 같다.

$$f(t_i)=\phi[N-(i-1)]e^{-\phi(N-(i-1))t_i},$$
$$F(t_i)=1-e^{-\phi[N-(i-1)]t_i},$$
$$R(t_i)=e^{-\phi(N-(i-1))t_i}$$

여기서 ϕ는 하나의 오류가 전체 소프트웨어에 미치는 비례 상수이며, N은 소프트웨어의 초기 결함 수, t_i는 $(i-1)$번째의 결함과 i번째의 결함 사이의 시간을 의미한다.

곡선 적합 모형은 가장 널리 사용되는 통계 분석 방법 중 하나인 독립변수와 종속변수의 관계를 분석하는 회귀분석을 활용하는 모형이다. 이는 소프트웨어의 오류 수를 종속변수로, 고장률, 복잡성, 시간, 환경 등 여러 조건 및 환경이 포함된 독립변수 간의 관계를 확인하여 가장 잘 맞는 모형을 정의하는 것을 목표로 한다. 곡선 적합 모형은 선형, 다항식, 비선형 곡선 적합 등 다양한 수학적 관계를 통해 소프트웨어 오류 수를 예측한다. 가장 기본적인 형태는 단순선형회귀 모형이며 다음의 수식과 같다.

$$y_t = \beta_0 + \beta_1 t + \epsilon$$

여기서 β_0는 절편, β_1은 기울기, ϵ은 오차항을 의미한다. 더 나아가서 하나의 독립변수보다는 여러 독립변수와 종속변수와 관계를 논하는 것이 여러 상황을 이해하는 데 쉬울 것이며, 이는 다항선형회귀분석으로 설명된다. 또한, 데이터 특성상 선형으로서 분리되는 경우보다 2차형, 3차형, 혹은 여러 형태의 곡선 등 다양한 비선형 곡선 형태를 띠는 경우가 많기 때문에 비선형 회귀곡선을 기반으로 곡선 적합 모형을 사용한다. <표 2-1>은 비선형 곡선 적합 모형에 대한 설명이다.

<표 2-1> 비선형 곡선 적합 모형

모형	수식
2차 모형	$y_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \epsilon$
3차 모형	$y_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \beta_3 t^3 + \epsilon$
성장 모형	$y_t = e^{\beta_0 + \beta_1 t} + \epsilon$
지수 모형	$y_t = \beta_0 e^{\beta_1 t} + \epsilon$
S 모형	$y_t = \beta_0 + \frac{\beta_1}{t} + \epsilon$
대수 모형	$y_t = \beta_0 + \beta_1 \ln t + \epsilon$
로지스틱 모형	$y_t = \frac{1}{k + \beta_0 \beta_1^t} + \epsilon$

시계열 모형은 시간의 흐름에 의해 변화를 보이는 현상을 관찰하여 현상을 대표할 수 있는 모형을 정의하거나 정의한 모형을 기반으로 예측하고자 하는 분석 모형이다(Yi 외, 2004). 시간의 흐름에 영향을 받는 주가 예측, 날씨 등의 분야에서 활용하며 소프트웨어 신뢰성을 측정하기 위한 방법 역시 시간의 흐름에 영향을 미치기 때문에 소프트웨어 신뢰성 연구에서도 활용된다. 시계열 모형을 활용하기 위해선 정상성을 가정해야 하고, 가정이 안 될시, 차분을 통해 정상성을 만족시켜 모형을 정의한다. 정상성은 추세나 계절성이 없는 자료를 의미하고, 차분은 시간의 흐름에 따라 나타나는 변화를 제거하기 위한 방법으로 정상성을 만족하지 않는 자료는 차분을 통해 해결한다. 위의 특징을 지닌 자료를 기반으로 시계열 모형을 제안하는 방법은 자기회귀모형(Auto-regressive model; AR 모형)과 이동평균모형(Moving average model; MA모형) 2가지로부터 시작된다. AR 모형은 현 시점의 자료를 이전 시점의 자료들로서 나타낼 수 있도록 만든 모형이며, MA 모형은 과거 자료를 정해진 개수만큼의 평균을 기반으로 계산하여 현시점을 예측하고자 하는 모형이다. 이 둘의 특징을 합친 모형은 자기회귀이동평균모형(Auto-regressive moving average model; ARMA 모형)이며, 모형의 적합성을 판단하기 위해 자기상관계수(Auto-correlation; AC)와 편자기상관계수(Partial auto-correlation; PAC)를 확인한다. <표 2-2>는 시계열 모형에 대한 설명이며, <표 2-2>에서 나타난 p와 q는 차수이며 과거 시점에서 p, q만큼 반영한다(Box 외, 2015).

<표 2-2> 시계열 모형

모형	수식
AR 모형	$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \cdots + \alpha_p y_{t-p} + \epsilon_t$
MA 모형	$y_t = \epsilon_t - \beta_1 y_{t-1} - \beta_2 y_{t-2} - \cdots - \beta_p y_{t-p}$
ARMA 모형	$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \cdots + \alpha_p y_{t-p} - \beta_1 y_{t-1} - \beta_2 y_{t-2} - \cdots - \beta_q y_{t-q} + \epsilon_t$

Singpurwalla와 Wilson(1994)은 고장 사이의 시간에 초점이 맞춰진 모형과 고장 횟수에 초점이 맞춰진 모형으로 구분 지었다. 여기서 마코프 과정을 통해 모형을 구현하거나, NHPP를 통해 모형이 구현되는, 두 가지 과정으로 모형이 구현된다. 마코프 과정은 현재 시점 이후의 환경은 현재 상태에만 의존하고 과거 시점과는 독립적인 관계가 있음을 가정하며, 주로 고장 사이의 시간에 초점을 둔다. 마코프 과정은 상태와 시간으로 구성되며 상태는 발생한 사건을 의미하고, 시간은 이산 시간과 연속 시간 두 가지로 정의한다. 이산 시간은 일정한 간격에 따라 상태의 변화를 관찰하고, 연속 시간은 연속적인 시간 속에서 변화를 관찰한다. 마코프 과정을 활용하기 위해선 정상성과 독립성이 충족되어야 한다. 이는 다음과 같은 과정을 통해 설명된다.

$$P[X_{n+1}=x_{n+1}|X_n=x_n,\cdots,X_0=x_0]=P[X_{n+1}=x_{n+1}|X_n=x_n],\ n=0,1,2,\cdots$$

Gokhale 외(1996)는 마코프 과정을 기반으로 고장 사이의 시간이 지수분포를 따름을 가정하여 연구하였으며 이를 마코프 모형이라 정의하였다. 이는 소프트웨어의 잔여 고장 수 또는 해당 시점까지 제거된 고장 수에 따라 언제든지 다른 결과를 도출시킬 수 있다. 따라서 마코프 모형은 일반적으로 가변적인 조건에 매우 효율적인 모형으로 제안되었다. 이에 따라 마코프 과정을 가정한 소프트웨어 신뢰성 모형은 다음과 같이 두 가지 유형으로 분류된다(Sideratos 외, 2014).

1. 동질 마코프 모형: 소프트웨어 전체의 총 결함 수를 유한하고, 일정한 값으로 가정한다. 또한, $t>0$ 시간에 스프트웨어 잔여 결함 수는 동질 마코프 연쇄로 계산한다.
2. 비동질 마코프 모형: 소프트웨어 결함 수를 확률 변수로 가정하며, $t>0$ 시간 동안 소프트웨어에서 발견된 결함 수는 비동질 마코프 모형을 따른다.

대표적인 마코프 모형의 소프트웨어 신뢰성 모형은 불완전 디버깅을 갖는 마코프 모형이다. 해당 모형은 디버깅에 성공할 확률 p와 디버깅에 실패할 확률 $q=1-p$가 반영된 k번째 고장 구간에 대한 신뢰도함수를 제안하였다. N은 초기 고장수, ϕ는 단위 시간당 평균 고장 발생률을 의미하며 다음과 같은 수식을 갖는다.

$$R_k(t) = \sum_{j=0}^{k-1} {}_{k-1}C_j \, p^{k-j-1} q^j e^{-[N-(k-j-1)]\phi t}, \quad j = 0,1,2,\cdots N$$

고장 횟수에 초점을 맞춘 모형은 대부분 비동질 포아송 과정인 NHPP를 기반으로 소프트웨어 신뢰성 모형을 정의한다. 초기 소프트웨어 신뢰성에 대한 연구는 1979년에 Goel, Okumoto가 연구한 NHPP 기반 소프트웨어 신뢰성 모형을 개발하면서부터 진행되었다. 해당 연구는 소프트웨어 고장 발생은 고장끼리 영향을 주지 않는, 독립적으로 고장이 일어날 것을 가정하면서 연구가 진행되었다. 이를 시작으로 소프트웨어 누적 고장 수가 S자 형태로 증가하는 형태나 소프트웨어 고장이 오목(Concave)한 형태로 발생함을 가정한 소프트웨어 고장 형태에 초점을 두고 연구가 진행되었다. NHPP S자형 곡선 모형이 소프트웨어 고장 형태를 가정한 대표적 모형이다. 이후 단순 형태만을 가정하지 않고, 외부적 요인에 대한 가정도 추가하여 소프트웨어를 개발하는 데 있어 테스트 단계에 투입되는 환경적 요인, 인적 자원 등을 고려한 Testing-effort NHPP 모형에 관한 연구가 진행되었다. 이를 기반으로 소프트웨어 고장에만 초점을 맞춘 모형이 아닌 현실적인 요소가 가미된 모형이 제안되었다.

기존의 소프트웨어 신뢰성 모형은 결함 및 고장이 발생하면 완전한 수정을 통해 결함 및 고장을 제거한 후 다음 단계로 나아가는 것을 가정하였다. 하지만 모든 결함과 고장이 즉시 수정 보완이 될 수 없기 때문에 불완전 디버깅에 대한 가정을 기반으로 연구가 진행되었으며 관련된 NHPP 불완전 디버깅 모형이 개발되었다. 불완전 디버깅을 가정한 모형 역시도 실제 운용환경과 개발과정에서의 운용환경에 대한 차이를 가정하지는 않는다. 실제 운용환경에서는 테스트 환경보다 더 많은 알 수 없는 상황이 발생할 수 있기 때문에 이에 대한 가정을 추가한 운용환경 불확실성을 고려한 소프트웨어 신뢰성 모형이 제안되었다. 또한, 과거 소프트웨어가 복잡한 구조나 다양한 형태를 갖고 있지 않을 때는 소프트웨어 고장이 독립적으로 발생함을 가정하였으나 소프트웨어가 복잡해지고 구조가 어려워지면서 독립적 발생이 아닌 종속적 발생을 가정하는 것이 필연적이게 되었다. 따라서 소프트웨어 고장이 종속적으로 발생함을 가정한 종속 고장 소프트웨어 신뢰성 모형이 등장하였고, 최근 많은 연구가 진행되고 있다.

여러 가정에 의해 개발된 소프트웨어 신뢰성 모형은 연구가 발전하면서 현실적인 부분을 가정한 특수한 경우에 적합한 모형이 많이 개발되었으며, 이를 일반화하는데 어려움이 발견되었다. 이를 개선하기 위해 비모수적 소프트웨어 신뢰성 모형

중 심층신경망을 활용한 소프트웨어 신뢰성 모형이나 소프트웨어의 고장 자체가 순차적 데이터 특성임을 이용한 순환신경망 계열을 활용한 소프트웨어 신뢰성 모형 등 기계학습을 활용한 소프트웨어 신뢰성 모형에 관한 연구가 진행되고 있다.

베이지안 모형은 소프트웨어 신뢰성 모형의 또 다른 유형으로 분류된다. 베이지안 소프트웨어 신뢰성 모형은 각 모수마다 사전분포를 가정하여 새로운 하나의 사후분포를 도출하며, 이는 예측 분석에 가장 적합한 분석 중 하나이다. 베이지안 방법의 가장 큰 장점은 테스트 결과와 같은 이전 정보를 테스트 데이터의 최신 정보와 결합할 수 있다는 점이며, 이는 소프트웨어 개발자가 사용 가능한 모든 정보의 조합을 기반으로 신뢰성을 예측하는 데 도움이 되기 때문에 매우 중요하다. 오류 유입 모형, 고장률 모형, 곡선 적합 모형, 시계열 모형 등 여러 확률론적 모형 중 가장 많이 활용되는 모형은 NHPP를 가정한 소프트웨어 신뢰성 모형이고, 이는 다음 절에서 자세히 소개한다.

2. NHPP 소프트웨어 신뢰성 모형

포아송 과정은 신뢰성 분야에서 핵심적인 확률과정(Stochastic process)이며, 셈과정(Counting process) 중 독립증분(Independent increment)과 정상증분(Stationary increment)을 만족하는 과정이다. 독립증분은 서로 겹쳐지지 않는 시간 단위 구간에서 발생하는 사건 수가 독립임을 가정한다. 특정 시간 동안 발생하는 총 사건 수를 $N(t)$라고 할 때, $N(t)$가 독립증분이면 $t_1 < t_2 < \cdots < t_n$에서 $N(t_1), N(t_2), \cdots, N(t_n)$은 서로 독립이다. 정상증분은 시간 단위 구간 s에서 발생하는 사건은 동일한 분포를 가정한다. $N(t)$가 정상증분일 때, $t_1 < t_2$, $s > 0$이면 $N(t_2) - N(t_1)$과 $N(t_2 + s) - N(t_1 + s)$는 동일한 분포를 갖는다. 위 두 조건을 만족하는 경우와 함께 길이 s의 시간 단위 구간 동안의 고장 횟수는 평균 λs를 갖는 포아송 분포를 가진다면 포아송 과정은 다음과 같이 표현되며, 초깃값은 $N(0) = 0$이다.

$$P[N(t+s) - N(t) = n] = \frac{e^{-\lambda s}(\lambda s)^n}{n!}, \quad n = 0, 1, 2 \cdots$$

여기서 단위 시간당 평균 고장 수를 나타내는 λ가 상수이며, λ는 시간 t에 의존하지 않는 동질성 포아송 과정을 따른다. 시간이 변하더라도 소프트웨어에서 발생하는 평균 고장 수는 일정함을 가정한다. 하지만 실제 신뢰성 분야 또는 소프트웨어에

서 발생하는 고장은 시간의 흐름에 독립적이지 않고, 종속적인 관계를 띈다. NHPP는 시간의 흐름에 따라 모수 λ가 영향을 받는 과정으로, 시간이 지남에 따라 모수 λ가 변하는 형태인 함수를 가진다. NHPP는 세 가지 조건을 만족하면 성립된다.

1. 초깃값 $N(0)$는 0이다.
2. 독립증분을 만족한다.
3. 구간 $[t_1, t_2]$에서의 고장 수의 평균은 $\int_{t_1}^{t_2} \lambda(t)dt$로 표현된다. $(t_2 > t_1)$

여기서 $\lambda(t)$는 강도함수로 시점 t에서의 순간 고장 수를 의미한다. 강도함수 $\lambda(t)$가 감소하면 순간 고장 수가 줄어들어 품질이 향상되는 것을 의미하며, $\lambda(t)$가 증가하면 순간 고장 수가 증가하여 품질이 떨어지는 것을 의미한다. 또한, 동질 포아송 과정에서 가정한 동일한 분포를 갖는 정상증분은 가정하지 않는다. NHPP를 가정한 소프트웨어 신뢰성 모형은 다음과 같은 식을 따른다.

$$P[N(t)=n] = \frac{(m(t))^n e^{-m(t)}}{n!}, \quad n = 0,1,2\cdots$$

여기서 $N(t)$는 시간 t까지 발생한 소프트웨어 고장 수이고, $m(t)$는 강도함수 $\lambda(t)$를 0 시점부터 t 시점까지를 적분한 평균값함수를 의미한다. $m(t)$는 다음과 같이 표현된다.

$$m(t) = \int_0^t \lambda(s)ds$$

$m(t)$가 t에 대해 단조증가함수(Monotonic increasing function)인 미분 가능한 함수이면 강도함수 $\lambda(t)$는 $\lambda(t) = m'(t)$을 만족한다. 따라서 $\lambda(t)$가 상수이면 동질 포아송 과정, $\lambda(t)$가 함수형태로 시간의 흐름에 따라 변하면 NHPP로 정의한다. 또한, 강도함수 $\lambda(t)$는 다음과 같은 수식으로 정리할 수 있으며, $m(t)$와 $\lambda(t)$의 관계에 대한 함수에 대입하면 신뢰도함수 $R(t)$는 다음과 같이 표현된다.

$$\lambda(t) = \frac{\frac{d}{dt}R(t)}{R(t)} = \frac{R'(t)}{R(t)},$$

$$R(t) = e^{-m(t)} = e^{-\int_0^t \lambda(s)ds}$$

궁극적으로 NHPP를 기반으로 한 소프트웨어 신뢰성 모형은 t시점까지의 누적 고장 수를 예측하는 것을 목적으로 하며 여러 형태의 수학적·통계적 가정이 포함된 평균값함수 $m(t)$를 활용하여 소프트웨어 신뢰성을 개선한다(Malaiya와 Srimani, 1991; Lee, 2004).

이를 기반으로 제안되었던 소프트웨어 신뢰성 모형은 발생한 고장의 오류에 대해 완전한 수정을 요구하는 완전 디버깅 환경을 가정한다. 여기서 디버깅은 컴퓨터 프로그램 개발 단계 중에 발생하는 시스템의 오류나 비정상적 연산을 찾아내고 수정하는 작업 과정이다. 따라서 소프트웨어 고장이 발생한다면 이를 찾아내 보완 및 개선 과정을 거쳐 완전한 수정 단계를 거친 후 다음 단계로 넘어가는 과정을 모두 포함한다. 이때 발생하는 소프트웨어 누적 고장 수의 증가 형태에 따라 다양한 소프트웨어 신뢰성 모형이 제안되었다.

가. NHPP exponential 모형

NHPP 소프트웨어 신뢰성 모형에서 가장 기본이 되는 형태는 NHPP exponential 모형을 기반으로 한다. NHPP exponential 모형은 시간이 지날수록 폭발적으로 증가하는 지수함수 형태를 띠고 있기 때문에 소프트웨어에서 발생하는 고장 수와 유사하다. 따라서 지수함수 형태를 가정한 소프트웨어 신뢰성 모형을 제안하였으며, 대부분의 NHPP 소프트웨어 신뢰성 모형은 이를 기반으로 확장되었다. 소프트웨어 신뢰성 모형은 다음과 같은 미분방정식을 기반으로 유도한다.

$$\frac{dm(t)}{dt} = b(t)[a(t) - m(t)] \quad (1)$$

여기서 $a(t)$는 소프트웨어 테스트 동안 t 시간에서의 고장 수 함수, $b(t)$는 결함당 고장검출률, $m(t)$는 평균값함수를 의미한다.

(1) Goel-Okumoto 모형

Goel-Okumoto 모형(GO 모형)은 지수함수 형태를 가정하는 소프트웨어 신뢰성 모형을 대표하는 소프트웨어 신뢰성 모형이며 3가지 가정을 만족한다(Goel과 Okumoto, 1979).

1. 소프트웨어의 고장은 독립적으로 발생한다.
2. 검출된 고장 수는 현재 고장 수에 비례한다($b(t)=b$).
3. 검출된 오류 및 고장은 다음 테스트 단계 이전에 제거하며, 새로운 오류 및 고장은 발생하지 않는다(완전 디버깅).

수식 (1)을 따르는 미분방정식의 $a(t)$와 $b(t)$를 일정한 값인 a와 b로 하여 평균값함수 $m(t)$와 강도함수 $\lambda(t)$를 유도한다. 여기서 a는 소프트웨어에 존재하는 테스트 단계 이전의 총 기대 결함 수, b는 고장검출률을 의미한다.

$$m(t)=a\left(1-e^{-bt}\right),$$
$$\lambda(t)=abe^{-bt}$$

(2) Hossian Dahiya GO 모형

Hossian Dahiya GO 모형(HDGO 모형)은 GO모형에 제약모수 c를 추가한 수정된 모형이다. 이는 다음과 같은 가정을 만족한다(Hossain과 Dahiya, 1993).

1. 초기 고장 수는 0개이다.
2. 검출된 오류 및 고장은 다음 테스트 단계 이전에 제거하며, 새로운 오류 및 고장은 발생하지 않는다(완전 디버깅).
3. 독립증분을 가정한다.
4. 잔여 고장에 대한 고장검출률은 일정하다.

HDGO 모형은 GO 모형의 최대우도함수의 해가 없다는 문제를 해결한다. 제약모수 c를 통해 GO모형의 최대우도함수에 대한 해의 조건을 완화하고, 무한대에서의 확률 질량을 줄일 수 있다. 모형 유도는 수식 (1)을 통해 유도하며 평균값함수 $m(t)$는 다음과 같다.

$$m(t)=\log\left[\frac{e^{a}-c}{e^{ae^{-bt}}-c}\right],$$

$$\lambda(t)=abe^{-bt}$$

GO모형에서 제약조건만 추가한 모형으로 강도함수 $\lambda(t)$는 GO모형과 동일하다. 만약 $c=0$일 때, HDGO 모형은 GO 모형과 동일하다. $c=1$일 때는 $m(\infty)=\infty$로 발산하여 소프트웨어의 고장 수를 결정하는데 다른 문제가 발생한다. 따라서 c의 범위는 0에서 1사이의 값으로 제한한다.

나. NHPP S-shaped 모형

NHPP S-shaped 모형은 완전 디버깅 모형의 형태 중 S자 곡선 형태를 가정한 모형이다. 시간이 지날수록 발생하는 고장 수가 증가하다 감소하는 형태를 띠며 NHPP exponential 모형처럼 일정한 고장검출률 b를 가지는 것이 아닌 특정 시점을 기준으로 시간이 지나면서 고장검출률이 점점 작아지는 함수 형태인 $b(t)$를 가진다. NHPP S-shaped 모형은 다음과 같은 미분방정식 (2)를 기반으로 평균값함수 $m(t)$를 유도한다.

$$\frac{dm(t)}{dt}=b(t)[a-m(t)] \tag{2}$$

여기서 a는 소프트웨어에 존재하는 테스트 단계 이전의 총 기대 결함 수, $b(t)$는 t 시점에서의 고장검출률을 의미한다.

(1) Delayed S-shaped 모형

Yamada 외(1983)가 제안한 Delayed S-shaped 모형(DS 모형)은 소프트웨어 고장 수가 S자 곡선 형태로 증가함을 가정한 모형이다. S자 곡선으로 설명되는 소프트웨어 고장 탐지 과정은 테스트 기술이 점차 향상되는 과정으로 특징지을 수 있다. DS 모형은 다음 가정을 기반으로 한다.

1. 소프트웨어의 초기 고장 및 결함은 일정한 값이 아닌 확률변수이다.
2. 소프트웨어는 소프트웨어에 존재하는 고장 및 결함으로 인해 또 다른 고장 및 결함이 발생할 수 있다.
3. $(i-1)$번째 고장과 i번째 고장 사이의 시간은 $(i-1)$번째 고장까지의 시간에 영향을 받는다.

DS 모형은 수식 (2)의 미분방정식을 통해 유도하며 여기서 고장검출률 $b(t)$는 $\frac{b^2t}{bt+1}$로 정의한다. 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음 수식으로 표현된다.

$$m(t)=a\left(1-(1+bt)e^{-bt}\right),$$
$$\lambda(t)=ab^2te^{-bt}$$

여기서 a는 소프트웨어에 존재하는 테스트 단계 이전의 총 기대 결함 수, $b(t)$는 고장검출률을 의미한다.

(2) Inflection S-shaped 모형

Inflection S-shaped 모형(IS 모형)은 결함의 종속성을 기반으로 하며 다음을 가정한다(Ohba, 1984).

1. 다른 고장 및 결함을 제거하기 전에는 일부 고장 및 결함이 검출되지 않는다.
2. 결함 탐지 확률은 소프트웨어에서 현재 탐지 가능한 결함 수에 비례한다.
3. 각각의 탐지 가능한 결함의 고장률은 고정되며 동일하다.
4. 결함의 원인이나 위치를 판별하여 분리시키는 고장 분리는 완전히 제거될 수 있다.

이를 기반으로 하는 IS 모형은 수식 (2)와 동일한 과정을 통해 유도되며, 여기서 고장검출률 $b(t)$는 $\frac{b}{1+\beta e^{-bt}}$로 정의한다. 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음 수식으로 표현된다.

$$m(t)=\frac{a(1-e^{-bt})}{1+\beta e^{-bt}},$$

$$\lambda(t)=\frac{ab(1+\beta)e^{-bt}}{(1+\beta e^{-bt})^2}$$

여기서 a는 소프트웨어에 존재하는 테스트 단계 이전의 총 기대 결함 수, $b(t)$는 고장검출률, β는 변곡 요인을 의미한다.

다. Testing-effort NHPP 모형

기존에 연구된 소프트웨어 신뢰성 모형은 테스트 단계에서 검출된 고장데이터를 기반으로 소프트웨어 신뢰성을 개선하기 위한 방향으로 연구가 진행되었다. 하지만 이는 소프트웨어를 개발함에 있어서 투입되는 다른 외부적 환경의 요소를 반영하지 않고, 고장데이터에 초점이 맞춰져 있다.

Testing-effort NHPP 모형(Yamada 외, 1986)은 테스트 단계에 투입되는 여러 자원, 즉, 테스트 횟수, 투입되는 인력, 테스트 단계에 소비된 CPU 시간 등 고장데이터 이외에 투입된 여러 자원이 반영된 모형이다. 이를 테스트 노력이라 하며, 테스트 노력을 통해 더욱 현실적인 소프트웨어 신뢰성 성장 모형을 개발하는 데 활용되었다. Testing-effort NHPP 모형은 다음의 가정을 만족한다.

1. 소프트웨어 시스템은 시스템에 남아있는 고장 및 결함으로 인해 또 다른 고장 및 결함이 발생할 수 있다.
2. 고장 및 결함은 검출되면 제거되고 새로운 고장 및 결함이 발생하지 않는다.
3. 테스트 노력에 투입되는 비용은 지수 곡선 또는 레일리 곡선을 가정한다.
4. 테스트 노력에 투입되는 비용에 대한 시간 구간$(t, t+\Delta t]$에서 검출된 s개의 기대 고장 및 결함 수는 s개의 기대 잔여 고장 및 결함 수에 비례한다.

5. 소프트웨어 테스트 단계의 고장 및 출은 NHPP를 기반으로 모형화한다.

Testing-effort NHPP 모형을 유도하기 위한 미분방정식은 다음과 같다.

$$\frac{dm(t)}{dt} = b(t)\gamma[a - m(t)] \tag{3}$$

기존의 미분방정식에 γ가 추가된 형태로 γ는 테스트 노력당 고장검출률 $(0 < \gamma < 1)$, a는 소프트웨어에 존재하는 테스트 단계 이전의 총 기대 결함 수, $b(t)$는 t 시점에서의 고장검출률을 의미한다.

(1) Yamada exponential 모형

Yamada exponential 모형(YE 모형)은 Testing-effort NHPP 모형에서 테스트 노력에 투입되는 비용이 지수 곡선을 가정함을 만족하는 모형이다. 미분방정식 (3)을 통해 유도되며, 이때의 $b(t)$는 $b(t) = a\beta e^{-\beta t}$로 정의한다. 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음 수식으로 표현된다.

$$m(t) = a\left(1 - e^{-\gamma\alpha\left(1 - e^{-\beta t}\right)}\right),$$
$$\lambda(t) = a\gamma\alpha\beta e^{-\gamma\alpha\left(1 - e^{-\beta t}\right) - \beta t}$$

여기서 α, β는 테스트 노력 함수의 모수$(\alpha > 0,\ \beta > 0)$이고, γ는 테스트 노력당 고장검출률$(0 < \gamma < 1)$이다.

(2) Yamada rayleigh 모형

Yamada rayleigh 모형(YR 모형)은 Testing-effort NHPP 모형에서 테스트 노력에 투입되는 비용이 레일리 곡선을 가정함을 만족하는 모형이다. 미분방정식 (3)을 통해 유도되며, 이때의 $b(t)$는 $b(t)=a\beta e^{-\frac{\beta}{2}t^2}$로 정의한다. 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음 수식으로 표현된다.

$$m(t)=a\left(1-e^{-\gamma\alpha\left(1-e^{-\frac{\beta t^2}{2}}\right)}\right),$$

$$\lambda(t)=a\gamma\alpha\beta t e^{-\gamma\alpha\left(1-e^{-\frac{\beta}{2}t^2}\right)-\frac{\beta}{2}t^2}$$

여기서 α,β는 테스트 노력 함수의 모수($\alpha>0,\ \beta>0$)이고, γ는 테스트 노력당 고장검출률($0<\gamma<1$)이다.

라. NHPP 불완전 디버깅 모형

기존의 소프트웨어 신뢰성 모형은 완전 디버깅 모형으로 완전한 수정 과정을 함께 가정한다. 그러나 실제 환경에서 이는 적합한 가정이 아니다. 테스트 단계에서 발생한 고장 및 결함은 수정 및 보완 과정을 거쳐 소프트웨어를 개선해나가지만, 모든 고장 및 결함에 대한 수정은 이루어지기 어렵기 때문에 발생하는 여러 비용을 고려하여 소프트웨어 테스트를 중단하고 출시할 최적의 시점을 결정할 필요가 있다. 이를 위해 NHPP 불완전 디버깅(Imperfect debugging) 모형은 완벽한 수정 및 보완 과정을 거친 완전 디버깅을 가정하는 것이 아닌 불완전 디버깅을 가정한다. 불완전 디버깅은 검출된 오류를 정정하거나 제거할 때, 추가적인 오류가 생길 수 있다는 가정을 만족한다. NHPP 불완전 디버깅 모형은 식 (1)인 미분방정식을 통해 평균값함수 $m(t)$를 유도한다. 여기서 $a(t)$는 초기 발생한 고장 및 결함과 추가로 발생한 고장 및 결함을 포함한 소프트웨어의 총 결함 함수를 의미하고, $b(t)$는 결함당 고장검출률을 의미한다.

(1) Yamada imperfect debugging 모형

Yamada imperfect debugging 모형(Yamada 외 1992)은 불완전 디버깅을 가정한 소프트웨어 신뢰성 모형으로 소프트웨어 고장 수 함수인 $a(t)$에 따라 두 가지 모형을 제안하였다. 또한, NHPP 불완전 디버깅 모형의 미분방정식에서 고장검출률 함수 $b(t)$가 일정한 값을 갖는 $b(t)=b$를 따른다. 첫 번째, Yamada imperfect debugging 모형1(YID1 모형)에서 소프트웨어 고장 수 함수 $a(t)$는 $a(t)=ae^{\alpha t}$로 정의한다. 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음 수식으로 표현된다.

$$m(t)=\frac{ab}{\alpha+b}\left(e^{\alpha t}-e^{-bt}\right),$$

$$\lambda(t)=\frac{ab(\alpha e^{\alpha t}+be^{-bt})}{\alpha+b}$$

두 번째, Yamada imperfect debugging 모형2(YID2 모형)에서 소프트웨어 고장 수 함수 $a(t)$는 $a(t)=a(1+\alpha t)$로 정의한다. 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음 수식으로 표현된다.

$$m(t)=a\left(1-e^{-bt}\right)\left(1-\frac{\alpha}{b}\right)+\alpha at,$$

$$\lambda(t)=abe^{-bt}\left(1-\frac{\alpha}{b}\right)+\alpha a$$

YID1 모형, YID2 모형에 대한 모수는 다음과 같다. a는 소프트웨어에 존재하는 총 기대 결함 수, b는 결함탐지율, α는 초기 결함에 도입된 결함 수의 증가율을 의미한다.

(2) Pham-Zhang 모형

Pham-Zhang 모형(PZ 모형)은 NHPP 불완전 디버깅 모형을 일반화한 모형으로 고장에 따라 고장검출률의 차이가 있고, 고장의 원인이 되는 오류는 제거되며 새로운 오류나 고장이 검출될 수 있음을 가정한다(Pham과 Zhang, 1997). 또한, PZ 모형은 다음과 같은 가정을 추가한다.

1. 오류 도입률은 지수함수 형태를 따른다.
2. 고장검출률 함수는 Inflection S-shaped 모형에서 비감소함수이다.

여기서 오류 도입률은 소프트웨어 테스트 단계에서 소프트웨어의 오류를 찾아내기 위해 새로운 코드를 도입할 때 발생하는 오류의 비율을 의미한다. 즉, 테스트 단계가 끝날 때보다 시작될 때 오류 수가 빠르게 증가한다. 이는 초기 소프트웨어에 많은 오류가 발생하지만, 시간의 흐름에 따라 많은 수정 및 보완 과정을 거치므로 소프트웨어에 적은 오류가 발생한다는 사실을 반영한다. 수식 (1)을 통해 평균값함수 $m(t)$와 강도함수 $\lambda(t)$를 다음과 같이 유도하며, 미분방정식 (1)에서 $a(t)$는 $a(t)=c+a(1-e^{-\alpha t})$, $b(t)$는 $b(t)=\frac{b}{1+\beta e^{-bt}}$로 정의한다.

$$m(t)=\frac{\left((c+a)\left[1-e^{-bt}\right]-\left[\frac{ab}{b-\alpha}\right]\left(e^{at}-e^{-bt}\right)\right)}{1+\beta e^{-bt}},$$

$$\lambda(t)=\frac{b(c+a)e^{-bt}-\frac{ab}{b-\alpha}\left(be^{-bt}-\alpha e^{-\alpha t}\right)}{1+\beta e^{-bt}}+\frac{b\beta e^{-bt}\left[(c+a)\left(1-e^{-bt}\right)-\frac{ab}{b-\alpha}\left(e^{-\alpha t}-e^{-bt}\right)\right]}{\left(1+\beta e^{-bt}\right)^2}$$

(3) Pham-Nordmann-Zhang 모형

Pham-Nordmann-Zhang 모형(PNZ 모형)은 PZ 모형과 동일하게 NHPP 불완전 디버깅 모형을 일반화한 모형으로 다음과 같은 가정을 만족한다(Pham 외, 1997).

1. 도입률은 시간에 따라 변하는 선형함수이다.
2. 고장검출률 함수는 Inflection S-shaped 모형에서 비감소함수이다.
3. 디버깅 단계에서 일정한 비율로 결함이 발생할 수 있다.

도입률은 개발과정에서 코드 결함이 나타나는 비율을 의미한다. 미분방정식 (1)을 통해 평균값함수 $m(t)$와 강도함수 $\lambda(t)$를 다음과 같이 유도한다. 여기서, $a(t)$는 $a(t)=a(1+\alpha t)$, $b(t)$는 $b(t)=\frac{b}{1+\beta e^{-bt}}$로 정의한다.

$$m(t)=\frac{a(1-e^{-bt})\left(1-\frac{\alpha}{b}\right)+\alpha at}{1+\beta e^{-bt}},$$

$$\lambda(t)=\frac{ab\left(1-\frac{\alpha}{b}\right)e^{-bt}+\alpha a}{1+\beta e^{-bt}}+\frac{b\beta e^{-bt}\left[a(1-e^{-bt})\left(1-\frac{\alpha}{b}\right)+\alpha at\right]}{(1+\beta e^{-bt})^2}$$

PZ 모형과 PNZ 모형에서 활용한 모수는 다음과 같다. a는 소프트웨어에 존재하는 총 기대 결함 수, b는 고장검출률, α는 초기 결함에 도입된 결함 수의 증가율, β는 변곡 요인을 의미한다.

(4) Fault removal efficiency NHPP 소프트웨어 신뢰성 모형

Fault removal efficiency NHPP 소프트웨어 신뢰성 모형(ZFR 모형)은 기존 NHPP 소프트웨어 신뢰성 모형에 결함 제거 효율성을 통합한 모형을 제안하였다 (Zhang 외 2003). 이는 아래와 같은 가정을 만족한다.

1. 소프트웨어 고장은 비동질 포아송 과정(NHPP)을 따른다.
2. 소프트웨어 고장률은 결함탐지율과 해당 시점에 나타난 잔여 결함 수에 따라 달라진다.
3. 소프트웨어 고장이 발생하면 확률에 따라 디버깅 작업이 이뤄지며, 이는 독립적으로 시행되고, 이와 별개로 새로운 오류가 확률적으로 도입될 수 있다.

ZFR 모형을 유도하기 위한 미분방정식은 다음과 같으며 $a(t)$는 초기 소프트웨어 기대 결함 수와 시간별 도입 결함 수의 합, $b(t)$는 결함탐지율, p는 검사 및 테스트를 통해 제거된 결함의 비율이다.

$$\frac{dm(t)}{dt} = b(t)[a(t) - pm(t)]$$

여기서 $a(t) = a\left(1 + \int_o^t \beta(u)b(u)e^{\int_o^u (p-\beta(\tau))b(\tau)d\tau} du\right)$, $b(t) = \frac{c}{1+\alpha e^{-bt}}$, $\beta(t) = \beta$를 만족하며, 이를 통해 평균값함수 $m(t)$와 강도함수 $\lambda(t)$를 다음과 같이 유도한다.

$$m(t) = \frac{a}{p-\beta}\left[1 - \left(\frac{(1+\alpha)e^{-bt}}{1+\alpha e^{-bt}}\right)^{\frac{c}{b}(p-\beta)}\right],$$

$$\lambda(t) = \frac{ac}{1+\alpha e^{-bt}}\left[\left(\frac{(1+\alpha)e^{-bt}}{1+\alpha e^{-bt}}\right)^{\frac{c}{b}(p-\beta)}\right]$$

여기서 a는 테스트 단계 이전의 소프트웨어 시스템의 초기 결함 수, b는 고장검출률, β는 일정한 결함 발생 확률을 의미한다.

(5) Testing coverage and imperfect debugging 모형

테스팅 커버리지(Testing coverage)는 소프트웨어 테스트에서 사용되는 지표 중 하나로 소프트웨어에 대해 충분히 테스트가 되었는지를 나타낸다. 테스팅 커버리지를 기준으로 개발자는 소프트웨어 신뢰성을 향상시키는데 추가적인 노력이 얼마나 필요한지를 측정할 수 있고, 사용자는 이를 통해 소프트웨어에 대한 정량적인 신뢰 기준으로서 판단이 가능하다. 따라서 테스팅 커버리지는 개발자와 사용자에게 모두 중요한 지표로 활용될 수 있다. 테스팅 커버리지가 포함된 소프트웨어 신뢰성 모형은 다음과 같은 미분방정식을 통해 유도한다.

$$\frac{dm(t)}{dt}=\frac{c'(t)}{1-c(t)}[a(t)-m(t)] \tag{4}$$

여기서 $a(t)$는 총 결함 수에 대한 함수, $c(t)$는 시간 t에서 코드 커버리지의 백분율을 의미하는 테스트 커버리지 함수, $c'(t)$는 커버리지 비율을 의미한다. 결함검출률 함수는 $\frac{c'(t)}{1-c(t)}$로 표현된다. Testing coverage and imperfect debugging 모형(IFD 모형)은 테스팅 커버리지에 대한 가정을 추가한 불완전 디버깅 모형으로 다음과 같은 가정을 만족한다(Pham과 Zhang, 2003).

1. 디버깅 과정 중에 각 결함을 제거하려는 노력이 완벽하지 않을 수 있으므로 불완전 디버깅 강도 비율 $d(t)$로 소프트웨어에 새로운 결함이 도입될 수 있다.
2. 불완전 디버깅 비율은 테스트 단계가 진행되면서 감소하고 테스트 단계가 끝날 무렵에는 무시할 수 있는 것으로 가정한다.

위의 미분방정식을 통해 평균값함수 $m(t)$와 강도함수 $\lambda(t)$를 구하면 다음과 같다. 여기서 $c(t)$는 고장검출률 b를 포함하는 $c(t)=1-(1+bt)e^{-bt}$, $d(t)$는 모수 d를 포함하는 $d(t)=\frac{d}{1+dt}$로 정의한다.

$$m(t)=a\left(1-e^{-bt}\right)\left(1+(b+d)t+bdt^2\right),$$
$$\lambda(t)=e^{-bt}\left((2abdt+ad+ab)e^{bt}+ab^2dt^2+(ab^2-abd)t-ad\right)$$

마. 운용환경 불확실성을 고려한 소프트웨어 신뢰성 모형

기존에 개발된 소프트웨어 신뢰성 모형은 테스트 단계에서 발생한 고장 및 결함 데이터를 기반으로 소프트웨어 고장 수를 예측한다. 이는 소프트웨어 출시시점을 결정하는 데 매우 유용하게 활용된다. 하지만 위의 과정으로 개발된 소프트웨어 신뢰성 모형은 테스트 환경과 실제 환경이 동일하다는 가정하에 이뤄진다. 하지만 소프트웨어 신뢰성 테스트 단계에서 최적의 출시시점을 계산하였더라도 실제 운용환경에서의 무작위성이 고려되지 않기 때문에 많은 문제를 야기할 수 있다. 특히, 비용적인 측면에서 실제 운용환경을 고려하지 않게 되면 무작위성에 의해 발생할 여러 고장에 대한 비용이 추가로 발생하여 많은 손해를 불러일으킬 수 있다. 결과적으로 이 방법은 실제 운용환경에서 소프트웨어 신뢰성에 대해 좋은 결과를 얻지 못할 수 있다. 이를 개선하기 위해 테스트 단계에서의 개발 환경과 실제 운용환경의 차이에 대한 운용환경에 대한 무작위성을 포함시킨 운용환경 불확실성을 고려한 소프트웨어 신뢰성 모형에 관한 연구를 꾸준히 진행하고 있다. 운용환경 불확실성을 고려한 소프트웨어 신뢰성 모형은 다음과 같은 미분방정식을 통해 모형을 유도하며, 운용환경의 불확실성을 나타내는 확률변수는 η이다.

$$\frac{dm(t)}{dt}=\eta b(t)[a(t)-m(t)] \tag{5}$$

(1) Generalized random field environment 모형

일반화된 NHPP 소프트웨어 신뢰성 모형은 다음과 같은 가정을 따른다.

1. 소프트웨어 작동 중 결함으로 인해 고장이 발생할 수 있다.
2. 특정 시점의 소프트웨어 고장검출률은 해당 시점에 잔여 고장 수와 비례한다.
3. 소프트웨어 고장 및 결함이 발생하는 경우, 확률적으로 오류가 즉시 제거되며, 이와 별개로 새로운 오류가 확률적으로 도입될 수 있다.
4. 운용환경은 고장검출률 $b(t)$에 η를 곱하여 영향을 미치며, η는 시간에 독립적이고, 단위가 없는 확률변수이다.

Generalized random field environment 모형(TP 모형)은 일반화된 NHPP 소프트웨어 신뢰성 모형의 가정을 만족하며 운용환경의 불확실성 모수 η의 분포가 모수 α, β를 갖는 감마분포를 따르며 실제 운용환경에서의 소프트웨어 고장검출률이 테스트 단계의 개발환경의 고장검출률보다 크거나 작은 경우에 대한 소프트웨어 신뢰성 모형이다(Teng과 Pham, 2006). 미분방정식 (5)를 활용하여 모형을 유도하며, 이때의 $b(t)$는 $b(t)=\frac{b}{1+ce^{-bt}}$로 정의한다. TP 모형의 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음과 같이 얻을 수 있다.

$$m(t)=\frac{a}{p-q}\left[1-\left(\frac{\beta}{\beta+(p-q)\ln\left(\frac{c+e^{bt}}{c+1}\right)}\right)^{\alpha}\right],$$

$$\lambda(t)=\frac{\alpha ab e^{bt}\left(\frac{\beta}{\beta+(p-q)\ln\left(\frac{c+e^{bt}}{c+1}\right)}\right)^{\alpha}}{\left(c+e^{bt}\right)\left(\beta+(p-q)\ln\left(\frac{c+e^{bt}}{c+1}\right)\right)}$$

여기서 TP 모형에서 활용한 모수는 다음과 같다. a는 소프트웨어에 존재하는 총 기대 결함 수, b는 고장검출률, c는 학습 곡선의 모양을 정의하는 모수, α,β는 운용환경의 불확실성 모수 η에 대한 형상모수와 척도모수, p는 소프트웨어에서 결함이 성공적으로 제거될 확률, q는 테스트 단계에서의 오류도입률이다.

(2) Vtub-shaped fault detection rate 모형

Vtub-shaped fault detection rate 모형(Vtub 모형)은 일반화된 NHPP 소프트웨어 신뢰성 모형의 가정을 만족하며, 운용환경에 따라 단위 시간당 소프트웨어 시스템 고장검출률의 불확실성을 통합한 새로운 소프트웨어 신뢰성 모형을 제안하였다(Pham, 2014). Vtub 모형을 유도하기 위해 미분방정식 (5)를 활용하며, 여기서 η는 모수 α와 β를 가지는 감마분포를 따르는 운용환경의 불확실성에 대한 확률변수고, $a(t)$는 테스트 단계 이전에 존재하는 총 기대 고장 수인 $a(t)=N$이다. $b(t)$는 모수 a와 b를 포함한 V자 형태를 가지는 $b(t)=bln(a)t^{b-1}a^{t^b}$로 정의한다. Vtub 모형에 대한 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음과 같다.

$$m(t)=N\left(1-\left(\frac{\beta}{\beta+a^{t^b}-1}\right)^{\alpha}\right),$$

$$\lambda(t)=\frac{Na^{t^b}b\alpha\beta^{\alpha}t^{b-1}\ln a}{\left(\beta+a^{t^b}-1\right)^{\alpha+1}}$$

(3) Three parameter fault detection rate 모형

Three parameter fault detection rate 모형(3P 모형)은 일반화된 NHPP 소프트웨어 신뢰성 모형을 확장하여 3가지 모수를 포함하는 결함탐지율 함수를 고려한 운용환경 불확실성을 고려한 소프트웨어 신뢰성 모형을 제안하였다(Song 외, 2017a). 해당 모형은 일반화된 NHPP 소프트웨어 신뢰성 모형의 기본 가정을 만족한다. 3P 모형을 유도하기 위해 미분방정식 (5)를 활용한다. 여기서 η는 모수 β를 가지는 지수분포를 따르는 운용환경의 불확실성에 대한 확률변수이며, $a(t)$는 테스트 단계 이전에 존재하는 총 기대 고장 수인 $a(t)=N$이다. $b(t)$는 변곡 S자 형태를 가지며 시간에 영향을 받는 비감소함수인 $b(t)=\frac{a}{1+ce^{-bt}}$로 모수 a와 b, c를 포함한다. 3P 모형에 대한 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음과 같다.

$$m(t)=N\left[1-\left(\frac{\beta}{\beta-\frac{a}{b}\ln\left(\frac{(1+c)e^{-bt}}{1+ce^{-bt}}\right)}\right)\right],$$

$$\lambda(t)=\frac{Na\beta\left(1-\frac{ce^{-bt}}{1+ce^{-bt}}\right)}{\left(\beta-\frac{a}{b}\ln\left(\frac{(1+c)e^{-bt}}{1+ce^{-bt}}\right)\right)^2}$$

(4) Testing coverage 모형

Testing coverage 모형(TC 모형)은 운용환경 불확실성을 고려한 소프트웨어 신뢰성 모형에서 테스트 커버리지까지 포함한 모형을 제안하였다(Chang 외, 2014). TC 모형을 유도하기 위한 미분방정식은 수식 (4)와 수식 (5)를 함께 고려한 모형과 같으며, 일반화된 NHPP 소프트웨어 신뢰성 모형과 TC 모형에서 가정한 내용은 동일하다.

$$\frac{dm(t)}{dt}=\eta\left(\frac{c'(t)}{1-c(t)}\right)[a(t)-m(t)]$$

여기서 $c(t)$는 $c(t)=1-e^{-(at)^b}$로 정의하고, η는 모수 α와 β를 갖는 감마분포를 따르는 확률변수이며 $a(t)$는 테스트 단계 이전에 존재하는 총 기대 고장 수인 $a(t)=N$을 따른다. 이를 기반으로 한 TC 모형의 평균값함수 $m(t)$는 다음과 같이 얻을 수 있다.

$$m(t)=N\left[1-\left(\frac{\beta}{\beta+(at)^b}\right)^{\alpha}\right],$$

$$\lambda(t)=\frac{Na^b b\alpha\beta^{\alpha}t^{b-1}}{(\beta+(at)^b)^{\alpha+1}}$$

바. 종속 고장 소프트웨어 신뢰성 모형

기존에 개발된 소프트웨어 신뢰성 모형의 가장 큰 공통점은 소프트웨어 고장은 독립적으로 발생함을 가정한다. 이는 발생하는 고장이 이후 발생하는 고장에 영향을 미치지 않는다고 판단하는 것이다. 하지만 소프트웨어의 시스템이 복잡해지고 여러 가지의 소프트웨어가 얽혀있는 구조에서는 발생하는 고장이 다른 고장에 영향을 미치지 않는다고 단정할 수 없다. 따라서 소프트웨어 고장은 종속적으로도 발생할 수 있음을 가정한 종속 고장(Dependent failure) 소프트웨어 신뢰성 모형이 제안되었다. 종속 고장을 가정하기 위한 미분방정식은 식 (1)에서 평균값함수 $m(t)$를 한번 더 곱해준 식과 같다. 여기서 $a(t)$는 소프트웨어의 총 고장 수 함수, $b(t)$는 고장검출률 함수이다.

$$\frac{dm(t)}{dt} = b(t)[a(t) - m(t)]m(t) \tag{6}$$

(1) 종속 고장을 가정한 NHPP 소프트웨어 신뢰성 모형

종속 고장을 가정한 NHPP 소프트웨어 신뢰성 모형(DPF1 모형)은 미분방정식 (6)의 소프트웨어 총 고장 수 함수인 $a(t)$를 $a(t) = a(1+\alpha t)$에서 α가 0인 $a(t) = a$로 제한한 특수한 모형이다(Lee 외, 2020). 고장검출률 함수 $b(t)$는 b와 c를 모수로 갖는 비감소함수인 $b(t) = \frac{b}{b + ce^{-bt}}$로 정의한다. 또한, 발생하는 초기 고장 값 $m(0)$는 $m(0) = h$으로 한다. 이를 통해 미분방정식을 풀어 종속 고장을 가정한 NHPP 소프트웨어 신뢰성 모형에 대한 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음과 같다.

$$m(t) = \frac{a}{1 + \frac{a}{h}\left(\frac{b+c}{c+be^{bt}}\right)^{\frac{a}{b}}},$$

$$\lambda(t) = \frac{a^3 bhe^{bt}\left(\frac{b+c}{be^{bt}+c}\right)^{\frac{a}{b}}}{(be^{bt}+c)\left(a\left(\frac{b+c}{be^{bt}+c}\right)^{\frac{a}{b}} + h\right)^2}$$

(2) 운용환경의 불확실성을 고려한 NHPP 종속고장 소프트웨어 신뢰성 모형

운용환경의 불확실성을 고려한 NHPP 종속고장 소프트웨어 신뢰성 모형(UDPF 모형)은 종속 고장을 가정한 NHPP 소프트웨어 신뢰성 모형에 운용환경의 불확실성까지 포함한 모형을 제안하였다(Lee 외, 2022). 이를 유도하기 위한 미분방정식은 다음과 같으며, 미분방정식 (6)에 운용환경의 불확실성에 대한 확률변수인 η를 곱해주는 형태로 이뤄진다.

$$\frac{dm(t)}{dt} = \eta b(t)[N - m(t)]m(t)$$

여기서 소프트웨어 총 고장 수 함수인 $a(t)$는 $a(t) = N$이고, 고장검출률 함수 $b(t)$는 $\frac{b^2 t}{bt+1}$로 정의한다. 이를 통해 운용환경의 불확실성을 고려한 NHPP 종속고장 소프트웨어 신뢰성 모형에 대한 평균값함수 $m(t)$와 강도함수 $\lambda(t)$는 다음과 같이 유도할 수 있다.

$$m(t) = N\left[1 - \left(\frac{\beta}{\alpha + bt - \ln(bt+1)}\right)^{\alpha}\right],$$

$$\lambda(t) = \frac{N\alpha\left(b - \frac{b}{bt+1}\right)\left(\frac{\beta}{\alpha + bt - \ln(bt+1)}\right)^{\alpha}}{\alpha + bt - \ln(bt+1)}$$

<표 2-3>은 앞서 소개한 모든 모형에 대한 정리된 수식이며, 본 논문에서 제안하고자 하는 모형과 비교하고자 한다.

<표 2-3> 소프트웨어 신뢰성 모형

No.	Model	Mean Value Function	Note
1	GO	$m(t)=a\left(1-e^{-bt}\right)$	Concave
2	HDGO	$m(t)=\log\left[\frac{e^{a}-c}{e^{ae^{-bt}}-c}\right]$	Concave
3	DS	$m(t)=a\left(1-(1+bt)e^{-bt}\right)$	S-Shape
4	IS	$m(t)=\frac{a\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$	S-Shape
5	ZFR	$m(t)=\frac{a}{p-\beta}\left[1-\left(\frac{(1+\alpha)e^{-bt}}{1+\alpha e^{-bt}}\right)^{\frac{c}{b}(p-\beta)}\right]$	S-Shape
6	YE	$m(t)=a\left(1-e^{-\gamma\alpha\left(1-e^{-\beta t}\right)}\right)$	Concave
7	YR	$m(t)=a\left(1-e^{-\gamma\alpha\left(1-e^{-\frac{\beta t^{2}}{2}}\right)}\right)$	S-Shape
8	YID1	$m(t)=\frac{ab}{\alpha+b}\left(e^{\alpha t}-e^{-bt}\right)$	Concave
9	YID2	$m(t)=a\left(1-e^{-bt}\right)\left(1-\frac{\alpha}{b}\right)+\alpha at$	Concave
10	PZ	$m(t)=\frac{\left((c+a)\left[1-e^{-bt}\right]-\left[\frac{ab}{b-\alpha}\right]\left(e^{at}-e^{-bt}\right)\right)}{1+\beta e^{-bt}}$	S-Shape, Concave

<표 2-3 (계속)> 소프트웨어 신뢰성 모형

No.	Model	Mean Value Function	Note
11	PNZ	$m(t)=\frac{a(1-e^{-bt})\left(1-\frac{\alpha}{b}\right)+\alpha at}{1+\beta e^{-bt}}$	S-Shape, Concave
12	TP	$m(t)=\frac{a}{p-q}\left[1-\left(\frac{\beta}{\beta+(p-q)\ln\left(\frac{c+e^{bt}}{c+1}\right)}\right)^{\alpha}\right]$	S-Shape
13	IFD	$m(t)=a(1-e^{-bt})(1+(b+d)t+bdt^2)$	Concave
14	Vtub	$m(t)=N\left[1-\left(\frac{\beta}{\beta+a^{t^b}-1}\right)^{\alpha}\right]$	S-Shape
15	TC	$m(t)=N\left[1-\left(\frac{\beta}{\beta+(at)^b}\right)^{\alpha}\right]$	S-shape, Concave
16	3P	$m(t)=N\left[1-\left(\frac{\beta}{\beta-\frac{a}{b}\ln\left(\frac{(1+c)e^{-bt}}{1+ce^{-bt}}\right)}\right)\right]$	S-Shape
17	DPF1	$m(t)=\frac{a}{1+\frac{a}{h}\left(\frac{b+c}{c+be^{bt}}\right)^{\frac{a}{b}}}$	S-Shape, Dependent
18	UDPF	$m(t)=N\left[1-\left(\frac{\beta}{\alpha+bt-\ln(bt+1)}\right)^{\alpha}\right]$	Concave, Dependent

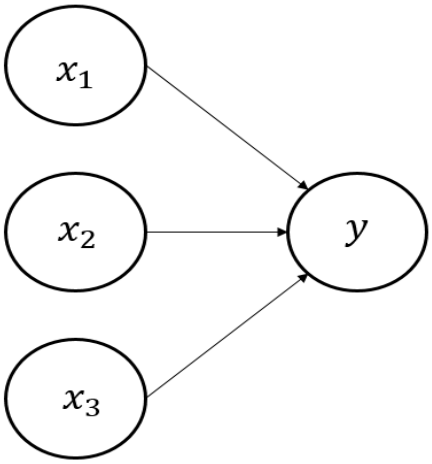
제3장 딥러닝

제1절 인공신경망 개요

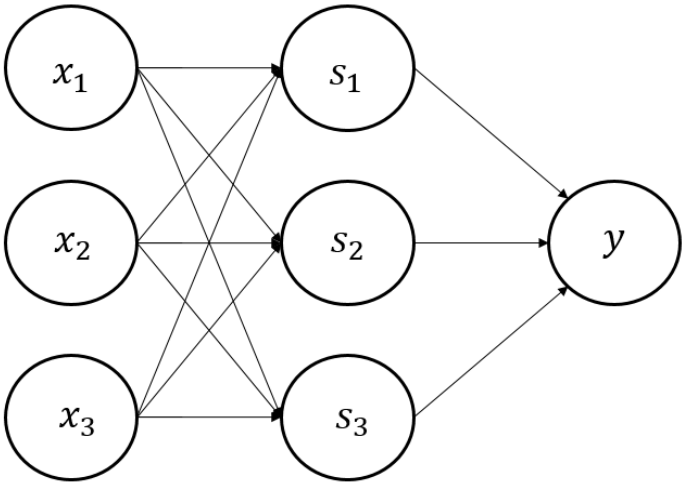
과거 인간은 인간의 뇌를 통해 사고하는 것을 컴퓨터에 구현시키고자 하는 노력을 꾸준히 진행하였다. 이에 대한 분야는 신경망(Neural network) 또는 인공신경망(Artificial neural network)이다. 인간은 시각, 미각, 후각, 청각, 촉각에 해당하는 5가지 감각으로부터 얻어진 정보를 통해 뇌로 전달하며 사고한다. 뇌까지 전달되는 과정은 신경세포 뉴런의 수상돌기에 신호 및 정보가 투입되면 축삭을 통해 다음 신경세포로 시냅스 과정을 거쳐 전달한다. 해당 과정 사이의 정보 전달은 어떠한 정보가 오고가는지, 더 나아가 최종적인 사고는 어떻게 이뤄지는지 알 수 없다. 하지만 신경세포를 통해 뇌까지 전달되는 과정을 컴퓨터 알고리즘을 통해 구현하면 인간이 사고하는 것을 모방할 수 있다. 해당 과정을 모형화한 것이 인공신경망이며 인간이 사고하는 과정을 모태로 한 네트워크를 의미한다.

초창기 인공신경망은 퍼셉트론으로부터 시작하였다(Rosenblatt, 1958). 퍼셉트론(Perceptron)은 입력된 값과 그에 맞는 가중치들과의 곱과 합을 통해 활성화함수를 거쳐 특정 임계값을 넘으면 1, 그렇지 않으면 0을 출력하는 구조를 가진 모형이다. 가장 최적의 값을 출력할 때까지 가중치를 조금씩 변화시키며 학습 과정을 거치고, 최적의 결과값을 도출한다. 하지만 퍼셉트론은 선형분류에만 특화된 구조로 AND와 OR 같은 분류는 쉽게 학습하고, 분류하였으나 XOR의 문제는 해결할 수 없었다. 이를 극복하기 위해 층을 여러 개로 설정한 다층 퍼셉트론(Multi layer perceptron)이 등장하였다(Rumelhart 외, 1986).

선형 분류만을 수행했던 퍼셉트론의 문제를 입력층과 출력층 사이에 은닉층을 두어 XOR과 같은 비선형 분류의 문제를 해결하였다. 이는 복잡한 데이터를 학습하는 데 매우 큰 도움을 주게 되었다. 또한, 은닉층의 수가 많아질수록 많은 연산 과정을 통해 실제 주어진 결과값과 매우 유사한 결과를 얻을 수 있었다. 층이 깊어질수록 학습하는데 어려움을 보였지만 역전파 알고리즘을 통해 이 문제도 극복하였다. 역전파 알고리즘은 입력값이 투입되면, 가중치와의 연산(곱과 합)을 통해 결과를 출력하고, 실제값과의 오차 함수를 연쇄 법칙을 활용하여 역으로 미분하면서 계산해내는 방법이다. 다층 퍼셉트론과 인공신경망은 같은 의미로서 많이 활용된다. 위와 같은 과정을 통해 많은 문제들이 해결되어 지금의 심층신경망, 합성곱신경망, 순환신경망 등 많은 딥러닝 모형으로 발전되었다. <그림 3-1>과 <그림 3-2>는 퍼셉트론과 다층 퍼셉트론에 대한 구조이다.



<그림 3-1> 퍼셉트론



<그림 3-2> 다층 퍼셉트론

제2절 심층신경망

인공신경망은 세 가지 성격을 지닌 뉴런 및 층이 존재한다. 첫 번째, 입력 뉴런 및 입력층은 외부로부터 발생한 정보 및 신호를 받아 인식하는 역할을 하고, 두 번째 출력 뉴런 및 출력층은 받은 정보를 여러 단계의 연산을 거쳐 결과값을 출력하는 역할을 담당한다. 세 번째, 은닉 뉴런 및 은닉층은 입력층에서 받은 정보를 출력층까지 비선형 연산을 거친 정보 및 신호를 전달하는 역할을 한다. 층에서 층으로 또는 뉴런에서 뉴런으로 정보가 전달될 때는 가장 기초가 되는 정보처리 단위인 뉴런이 시냅스 역할을 하는 비선형 결합이 가능한 활성화함수를 통해 다음 뉴런으로 전달한다. 마지막 출력층까지 전달되면 계산된 결과를 통해 사고나 판단을 한다. 심층신경망(Deep neural network; DNN)은 인공신경망에서 은닉층을 다수의 형태로 하여 비선형 연산을 여러 번 할 수 있도록 구성한 네트워크이다 (Karunanithi 외, 1992; Tamura와 Yamada, 2016).

입력층에서 데이터가 투입되면 다음 은닉층은 입력값 x와 가중치 W, 편향 b와 결합 후 활성화함수를 거쳐 다음 층으로 이동하며 출력층까지 전달된다. 층에서 층으로 넘어가는 수식은 다음과 같다. 이전 층에서 활성화함수를 통해 변환된 z(또는 입력값 x)와 가중치의 곱, 그리고 편향의 합으로 구성되어 있다.

$$u_{1i} = b_1 + \sum_{n=1}^{i} W_{1n} x \qquad u_{sj} = b_s + \sum_{m=1}^{j} W_{sm} z_{s-1m}$$
$$z_{1i} = f(u_{1i}) \qquad z_{sj} = f(u_{sj})$$

각 층마다 다음 층으로 넘어갈 때의 활성화함수는 주로 시그모이드 함수, 하이퍼볼릭 탄젠트 함수(Hyperbolic tangent; tanh), ReLU 함수인 비선형 함수를 사용한다. 마지막 출력층으로 전달된 결과를 출력할 때 사용되는 활성화함수는 연속형 변수일 때 항등함수, 범주형 변수일 때 시그모이드 함수 또는 소프트맥스(Softmax) 함수를 사용한다. <그림 3-3>는 ReLU, 시그모이드, tanh 활성화함수를 보여준다.

위의 과정을 통해 최종적인 예측값 y를 얻는다. 예측값 y와 실제값 t의 차이를 손실함수로 계산하고, 차이를 가장 작게 하는 각각의 가중치에 대해 손실함수의 최소값을 가지는 방향으로 업데이트한다. 가중치 업데이트는 역전파법 알고리즘을 기반으로 손실함수 수식 (7)을 특정 가중치 수식 (8)에 대한 편미분 값을 기준으로

학습률(α)만큼 업데이트되는 것을 의미한다. DNN은 일련의 과정들을 거쳐서 학습을 하고, 실제값에 가까운 예측값을 출력한다. <그림 3-4>는 위의 과정대로 전개된 DNN 모형의 구조를 보여준다.

$$Loss(W) = \sum_{i=1}^{n}(y_i - t_i)^2 \tag{7}$$

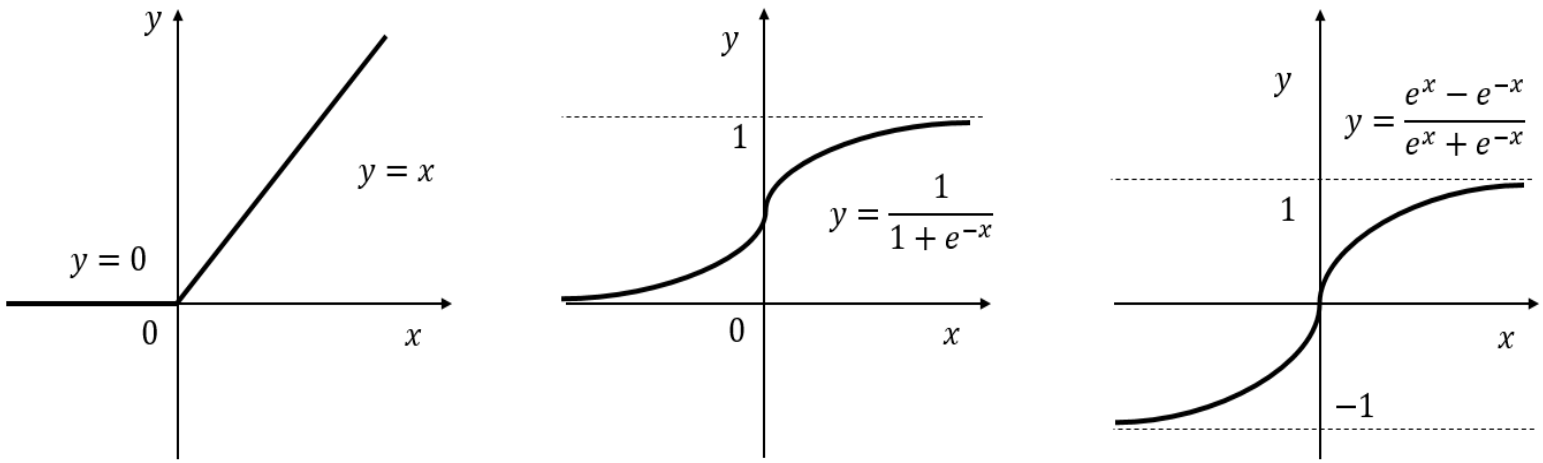
$$W = W - \alpha \frac{\partial}{\partial W} Loss(W) \tag{8}$$

제3절 순환신경망

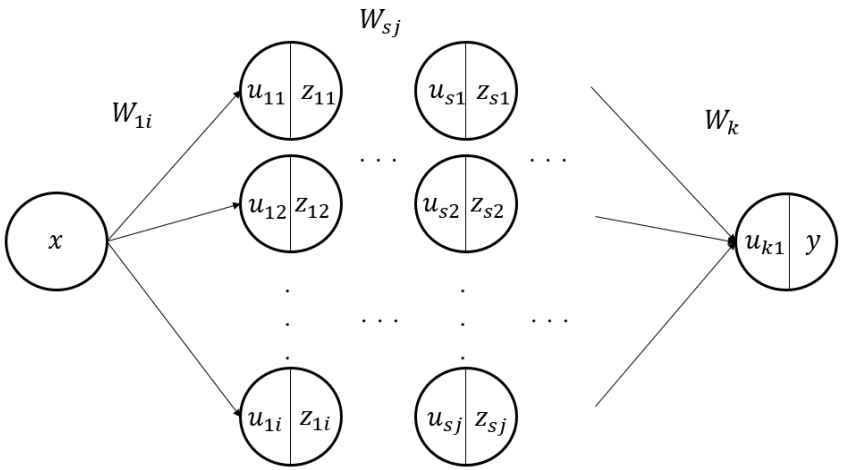
순환신경망(Recurrent neural network; RNN)은 가장 기본이 되는 DNN에서 은닉층 안에 이전 시점의 정보를 기억할 수 있는 셀이 있는 모형으로 과거 시점의 정보를 기억하여 이후에 일어날 일들에 영향을 미칠 수 있도록 설계된 구조이다. 순서에 의미가 있는 순차적 데이터 구조에 특화된 모형으로 텍스트 데이터나, 날씨나 주식 등과 같은 시계열 데이터에 매우 적합한 모형이다(Sutskever 외, 2014). 소프트웨어 고장데이터 역시 시간의 흐름에 정리된 데이터로 RNN 모형에 활용된다. 이를 모형의 구조로 표현하면 재귀적 모형으로 <그림 3-5>와 같은 구조를 이루고 있으며, 이를 전개하면 <그림 3-6>과 같이 표현된다.

$$h_t = \tanh(W_{h_t} h_{t-1} + W_{x_t} x_t + b)$$

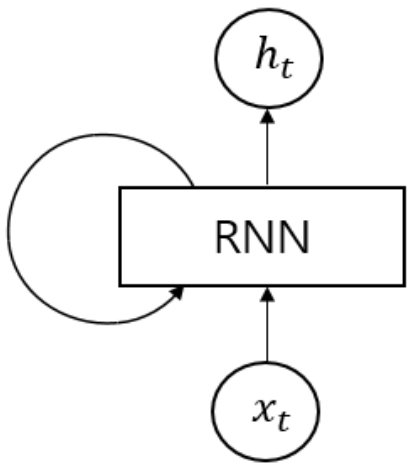
h_{t-1}은 $t-1$ 시점의 은닉상태를 의미하고 해당 정보가 이전 층의 정보를 포함한다. x_t는 현재 시점의 정보를 의미하며 각각의 가중치 W_{h_t}와 W_{x_t}의 곱과 편향 b의 결합을 통해 t 시점의 은닉상태와 $t+1$ 시점으로 정보를 전달한다.



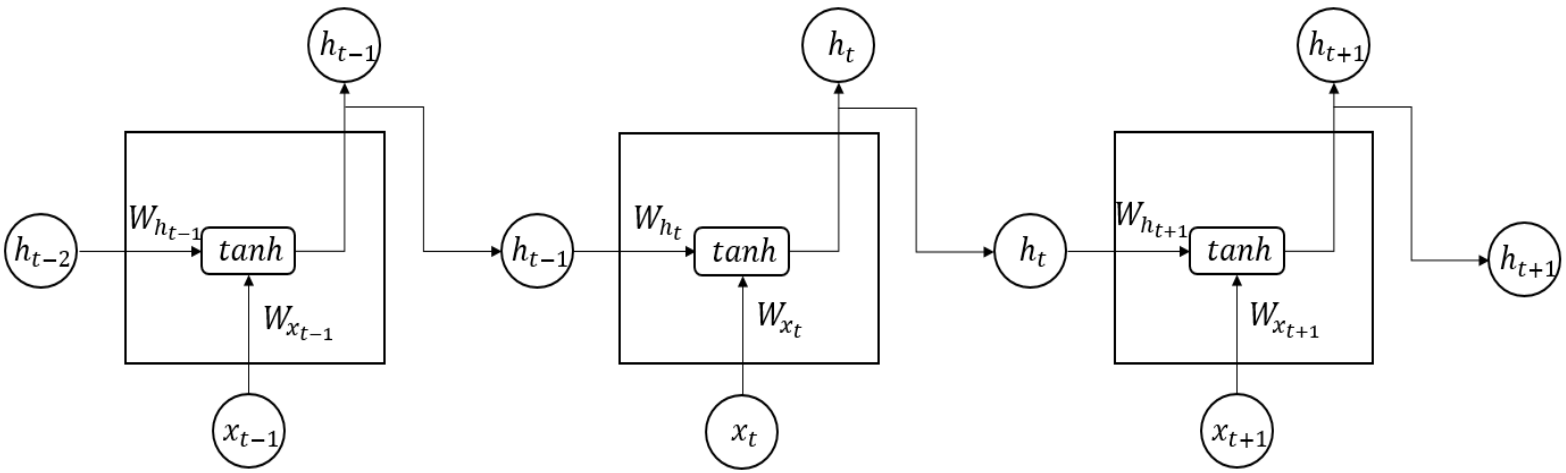
<그림 3-3> 활성화함수(ReLU, 시그모이드, tanh)



<그림 3-4> DNN 구조



<그림 3-5> RNN 구조


h_{t-1}
h_t
h_{t+1}
h_{t-2}
$W_{h_{t-1}}$
$tanh$
$W_{x_{t-1}}$
x_{t-1}
h_{t-1}
W_{h_t}
$tanh$
W_{x_t}
x_t
h_t
$W_{h_{t+1}}$
$tanh$
$W_{x_{t+1}}$
x_{t+1}
h_{t+1}

<그림 3-6> RNN 전개

제4절 장단기 메모리

RNN은 과거 시점을 반영하는 구조로 이루어져 있지만, 시점이 길어질수록 오래 전에 존재한 시점에 대한 정보가 잘 반영되지 않는 기울기 소실(Gradient vanishing) 문제를 가지고 있다. 이러한 기울기 소실 문제를 극복하기 위해 장기 의존성을 학습할 수 있는 구조로 만들어진 모형이 장단기 메모리(Long short term memory; LSTM)이다. RNN 구조와 동일하게 과거 시점의 정보를 반영하는 은닉층을 포함하고 있으나 내부의 구조는 다르다. LSTM에서는 메모리 셀 C가 존재한다. 메모리 셀 C는 망각(Forget) 게이트, 입력(Input) 게이트, 출력(Output) 게이트를 포함한다. 이때 활용되는 활성화함수는 시그모이드 함수(σ)와 tanh 함수이며, 새로운 정보가 얼마나 반영되는지를 의미하는 g와 함께 총 3가지 게이트와 기억 셀 C를 거치면서 과거 시점의 정보를 전달한다(Hochreiter와 Schmidhuber, 1997). 이를 통해 기울기가 소실되는 문제를 해결할 수 있다. 모형의 구조는 <그림 3-7>과 같다.

입력 게이트는 이전 층에 대한 가중치 $W_{i,h}$, $W_{g,h}$와 은닉상태 값, 현재 시점의 데이터에 대한 가중치 $W_{i,x}$, $W_{g,x}$와 현재 시점의 데이터가 곱하고 더한 후 시그모이드 함수와 tanh 함수를 통해 $0 \sim 1$, $-1 \sim 1$ 사이의 값으로 변환한 i_t와 g_t를 얻는다. i_t와 g_t를 통해 현재 새로운 정보를 얼마나 저장할지를 정하는 역할을 한다.

$$g_t = \tanh(W_{g,x}x_t + W_{g,h}h_{t-1} + b_g),$$
$$i_t = \sigma(W_{i,x}x_t + W_{i,h}h_{t-1} + b_i)$$

망각 게이트는 이전 층에 대한 가중치 $W_{f,h}$와 은닉상태 값, 현재 시점의 데이터에 대한 가중치 $W_{f,x}$와 현재 시점의 데이터가 곱하고 더한 후 시그모이드 함수를 통해 $0 \sim 1$ 사이의 값으로 변환한 f_t를 얻는다. 이를 통해 망각 게이트는 필요 없다고 판단하는 정보를 망각하는 역할을 한다. 여기서 0에 가까울수록 받은 정보를 망각하며, 1에 가까울수록 모든 정보를 기억하는 것을 의미한다.

$$f_t = \sigma(W_{f,x}x_t + W_{f,h}h_{t-1} + b_f)$$

입력 게이트와 망각 게이트를 통해 메모리 셀의 상태를 의미하는 셀 상태 C_t를 출력한

다. 이는 각 게이트를 통해 얻은 정보를 취합하여 정보를 망각할지 저장할지를 판단하는 역할을 한다. 수식은 다음과 같다. 여기서 $\odot$은 행렬 곱이 아닌 원소별 곱으로 이뤄진다.

$$C_t = f_t \odot C_{t-1} + g_t \odot i_t$$

출력 게이트는 이전 층에 대한 가중치 $W_{o,h}$와 데이터에 대한 가중치 $W_{o,x}$로 다른 게이트와 동일한 과정을 거쳐 셀 상태 C_t와 함께 최종 출력값인 h_t를 출력한다.

$$o_t = \sigma(W_{o,x}x_t + W_{o,h}h_{t-1} + b_o),$$
$$h_t = o_t \odot \tanh(C_t)$$

제5절 게이트 순환 유닛

LSTM은 RNN의 기울기 소실 문제를 해결할 수 있으나, 다음 단계로 넘어갈 때 1번의 연산만을 거쳤다면 LSTM에서는 4번의 연산 과정을 거친다. 빅데이터를 처리함에 있어서 이는 연산량이 4배가 증가하기 때문에 비효율적일 수도 있다. 이를 보완하여 게이트 순환 유닛(Gated recurrent unit; GRU)는 총 하나의 층에서 3개의 연산 과정을 거쳐 다음 층으로 값을 보내준다(Che 외, 2018). GRU는 LSTM에서 입력게이트와 망각게이트를 통합하여 업데이트 게이트로 새로 정의하고, 출력게이트는 리셋게이트로 대체한다. 모형의 구조는 <그림 3-8>과 같다.

리셋 게이트(γ)와 업데이트 게이트(z)는 이전 층에 대한 가중치 $W_{\gamma,h}$, $W_{z,h}$와 은닉상태 값, 현재 시점의 데이터에 대한 가중치 $W_{\gamma,x}$, $W_{z,x}$와 현재 시점의 데이터가 곱하고 더한 후 시그모이드 함수를 통해 $0 \sim 1$ 사이의 값으로 변환한 γ_t, z_t를 얻는다. 리셋 게이트에서는 이전 정보를 얼마나 잊을 것인지를 결정한다. 업데이트 게이트는 이전 정보를 얼마나 유지할지를 결정한다.

$$\gamma_t = \sigma(W_{\gamma,x}x_t + W_{\gamma,h}h_{t-1} + b_\gamma),$$
$$z_t = \sigma(W_{z,x}x_t + W_{z,h}h_{t-1} + b_z)$$

g_t는 은닉상태의 정보를 그대로 사용하지 않고, 리셋 게이트를 거친 결과를 활용한다. h_t는 현재의 정보와 이전 층의 정보를 리셋 게이트와 업데이트 게이트를 통

해 통합하여 다음 층으로 전달한다.

$$g_t = \tanh(W_{g,x}x_t + W_{g,h}(\gamma_t \odot h_{t-1}) + b_g)$$
$$h_t = (1 - z_t) \odot g_t + z_t \odot h_{t-1}$$

제6절 최적화 기법

앞 절에서 소개한 여러 형태의 딥러닝 모형은 단순 모형만 가지고 최적의 결과값을 도출할 수 없다. 실제값과 예측값의 차이를 가장 적게 학습하는 과정에서 최적의 방법을 통해 가중치를 조정해 나갈 수 있도록 설계해야 한다.

1. 경사 하강법

경사 하강법(Gradient descent)은 손실함수를 최소화시키는 가중치를 찾는 방법 중 하나로 가중치 W에 대한 손실함수 $Loss(W)$를 작아지는 방향으로 학습률 α만큼 이동시켜 손실함수 $Loss(W)$가 최소가 되도록 가중치 W를 변화시켜가며 학습한다(Ruder, 2016). 여기서, 학습률 α는 어느 정도의 기울기를 반영하여 가중치 W를 학습시킬 것인지를 결정하는 하이퍼파라미터이다. 학습률이 크면 최소값을 건너뛸 수 있고, 작으면 수렴하는데 시간이 오래 걸린다는 문제가 있기 때문에 적절한 수준의 학습률을 설정하는 것은 매우 중요하다. 경사 하강법의 종류는 반영하는 데이터의 크기에 따라 확률적 경사 하강법, 배치 경사 하강법, 미니배치 경사 하강법으로 나뉜다. 배치 경사 하강법은 전체 데이터 세트를 활용하고, 미니배치 경사 하강법은 전체 데이터 세트에서 일부를 뽑은(Mini batch) 데이터를 기반으로 가중치를 업데이트하며 확률적 경사 하강법은 데이터 하나씩 활용하여 가중치를 업데이트하는 방법이다. 학습하는 식은 수식 (8)을 기준으로 하며, 모멘텀, AdaGrad, Adam 최적화 기법은 이를 기반으로 구성한다.

2. 모멘텀

모멘텀(Momentum)은 물체가 움직이는 동안 움직이는 방향으로 쌓인 운동량을 의미한다. 모멘텀은 이전에 움직인 방향대로 움직일 수 있도록 도와준다. 따라서 이를 활용한 모멘텀 기법은 현재 학습 단계의 기울기와 이전 학습 단계의 기울기를 함께 고려하여 가중치를 학습시키는 방법이다(Qian, 1999). 모멘텀 방법은 다음과 같은 수식을 따른다.

$$v_t = \beta v_{t-1} - (1-\beta)\frac{\partial}{\partial W_t} Loss(W_t), \quad W_{t+1} = W_t - \alpha v_t$$

하이퍼파라미터 v는 속도를 의미하고, $\frac{\partial}{\partial W} Loss(W)$는 가속도를 의미하며 이 둘을 지수가중평균을 통해 정의한다. 가중치 W는 지수가중평균을 활용한 속도 v와 함께 가중치 W를 업데이트한다. 여기서 하이퍼파라미터 β는 모멘텀이며, 주로 0.9를 사용한다.

3. AdaGrad

AdaGrad(Adaptive gradient)는 가중치를 학습할 때 학습률을 다르게 조절하는 방법이다. 경사 하강법이나 모멘텀은 학습률을 일정하게 유지하여 학습하는데, 이는 손실함수가 최소가 되는 최적값을 찾는데 오류를 범할 수 있다. 이와 같은 문제를 해결하기 위한 AdaGrad는 학습 초기 단계에서는 학습률을 크게 하면서 점차 줄여나가는 방법이다(Ruder, 2016). 수식은 다음과 같다.

$$h_t = h_{t-1} + \frac{\partial}{\partial W_t} Loss(W_t) \odot \frac{\partial}{\partial W_t} Loss(W_t)$$

$$W_{t+1} = W_t - \alpha \frac{1}{\sqrt{h_t}} \frac{\partial}{\partial W_t} Loss(W_t)$$

가중치가 업데이트되는 과정에서 h는 학습률을 제곱하면서 학습률을 조정한다.

이를 무한히 반복하면 학습률이 0에 수렴하기 때문에 이 문제를 해결한 최적화 방법은 RMSprop(Root mean square propatation)이다. 이는 과거의 모든 기울기를 반영하지 않고, 먼 과거의 기울기는 줄여가면서 새로운 기울기 정보는 크게 반영하는 구조를 띈다.

$$u_t = \beta_1 u_{t-1} + (1-\beta_1)\left(\frac{\partial Loss(W_t)}{\partial W_t}\right),\ \ s_t = \beta_2 s_{t-1} + (1-\beta_2)\left(\frac{\partial Loss(W_t)}{\partial W_t}\right)^2,$$

$$W_{t+1} = W_t - \alpha\frac{u_t}{\sqrt{s_t+\epsilon}}$$

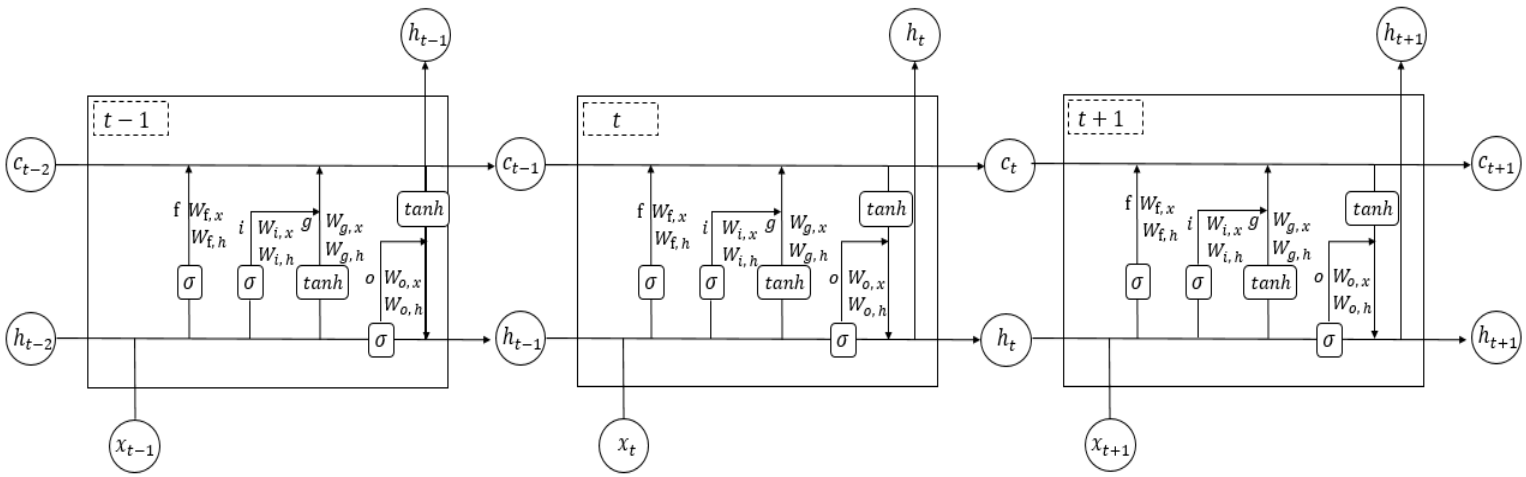
4. Adam

Adam(Adaptive moment estimation)은 모멘텀 방법과 AdaGrad 방법을 합친 방법으로, 이전 시점의 학습의 가속도를 이용하는 모멘텀 기법과 학습률 α가 일정하지 않고 학습 때마다 변하는 구조를 가지는 RMSprop 기법을 결합한 최적화 방법이다(Kingma와 Ba, 2014). 수식은 다음과 같다.

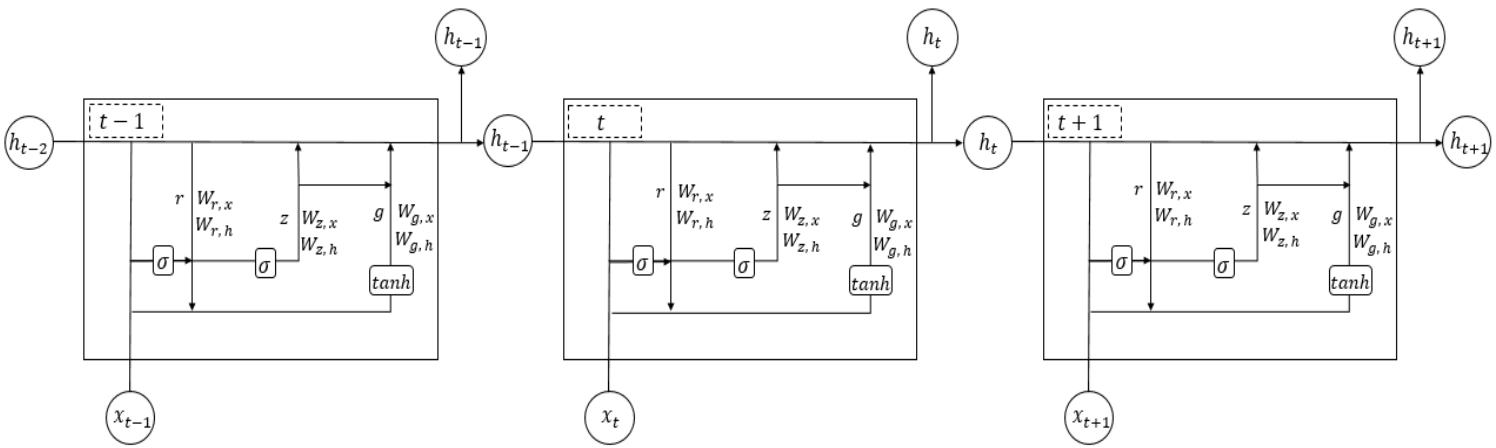
$$u_t = \beta_1 u_{t-1} + (1-\beta_1)\left(\frac{\partial Loss(W)}{\partial W}\right),\ \ s_t = \beta_2 s_{t-1} + (1-\beta_2)\left(\frac{\partial Loss(W)}{\partial W}\right)^2,$$

$$W_{t+1} = W_t - \alpha\frac{u_t}{\sqrt{s_t+\epsilon}}$$

β_1과 β_2는 모멘텀 기법과 RMSprop 기법의 지수 이동평균을 가정하며, 이를 기반으로 u_t와 s_t를 구한 후 학습률 α와 함께 가중치를 업데이트한다. 일반적으로 β_1과 β_2는 0.9와 0.99를 활용한다.



<그림 3-7> LSTM 구조 및 전개



<그림 3-8> GRU 구조 및 전개

제4장 새로운 소프트웨어 신뢰성 모형

제1절 종속고장을 고려한 새로운 NHPP 소프트웨어 신뢰성 모형

과거 많은 연구자들은 소프트웨어가 독립임을 가정한 소프트웨어 신뢰성 모형을 연구하였다. 하지만 소프트웨어가 복잡하고, 다양한 구조로 발전하면서 소프트웨어 고장은 발생하는 고장끼리 영향을 주는 종속적인 형태로 발생하는 경우가 많아졌다. 여기서 종속 고장이란 하나의 장애가 다른 장애에 영향을 미치거나 다른 장비의 고장확률을 높이는 것을 의미한다. 종속 고장에는 크게 두 가지 유형이 있다. 첫 번째, 공통원인 고장은 공통원인으로 인해 여러 소프트웨어가 동시에 고장이 발생하는 경우이며, 두 번째, 계단식 고장은 시스템의 일부가 고장을 일으켜 다른 소프트웨어에도 영향을 미치는 경우이다. 현재 여러 가지 소프트웨어가 결합된 형태인 소프트웨어 시스템에서의 고장은 독립적으로 발생하는 것보다 고장 발생이 서로 영향을 미쳐 고장이 가중되어 더 많은 고장이 발생할 것으로 보인다. 특히, 소프트웨어의 구조가 복잡하거나 여러 소프트웨어를 포함하고 있는 시스템일 경우, 발생하는 고장이 다른 소프트웨어의 고장에 영향을 미칠 수 있다. 따라서 제안하고자 하는 종속 고장을 고려한 NHPP 소프트웨어 신뢰성 모형(DPF2 모형)에서는 고장 발생이 서로 종속적인 관계로서 고장이 발생한 후 다음 발생하는 고장에 영향을 준다는 것을 가정한다. 종속 고장을 고려한 NHPP 소프트웨어 신뢰성 모형은 다음과 같은 가정을 만족한다.

1. 소프트웨어 고장 및 결함은 비동질성 포아송 과정(NHPP)를 따른다.
2. 소프트웨어에서 발생하는 고장은 다른 고장에 영향을 미친다.
3. 평균값함수 $m(t)$의 초깃값은 $m(0)=h$이다.

앞서 2장에서 소개한 소프트웨어 신뢰성 모형을 유도하기 위한 가장 기본이 되는 미분방정식은 소프트웨어 고장 수 함수인 $a(t)$와 고장검출률 함수 $b(t)$를 포함하며 다음과 같은 수식을 따른다.

$$\frac{dm(t)}{dt}=b(t)[a(t)-m(t)]$$

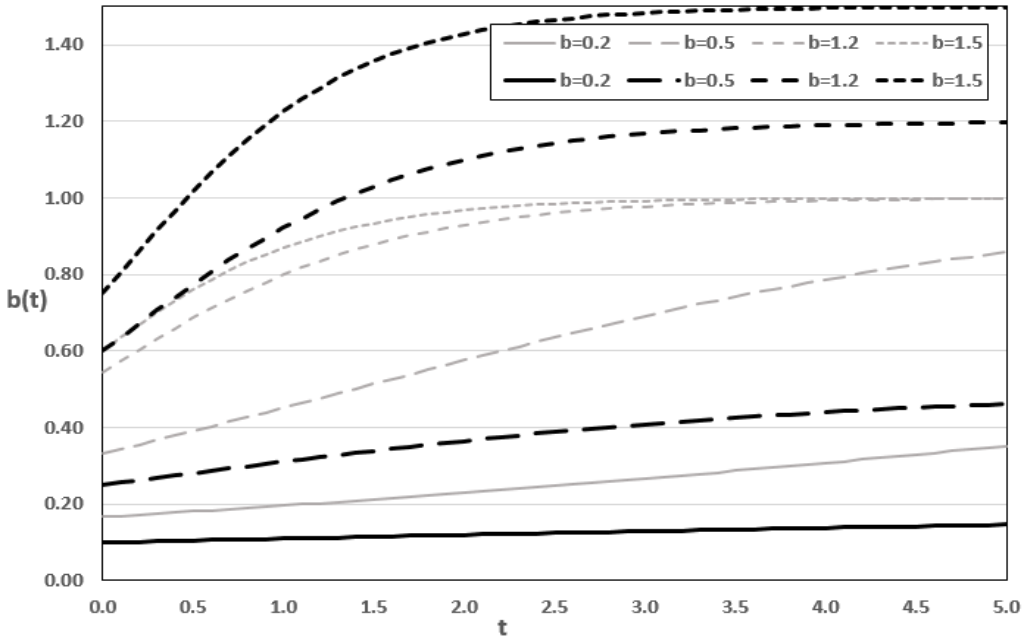
여기서 종속 고장을 포함하는 미분방정식은 앞선 미분방정식에 평균값함수인 $m(t)$를 한번 더 곱해준 형태로 이전에 발생한 고장이 다음 고장에 영향을 주는 형

태를 띄며 소프트웨어 고장 수 함수인 $a(t)$와 고장검출률을 의미하는 $b(t)$가 함께 미분방정식을 구성한다.

$$\frac{dm(t)}{dt}=b(t)[a(t)-m(t)]m(t) \tag{9}$$

식 (9)로부터 본 논문에서 제안하고자 하는 DPF2 모형에서의 $a(t)$는 $a(1+\alpha t)$, $b(t)$는 $\frac{b}{1+ce^{-bt}}$이며 a는 기대 고장 수, α는 결함 수의 증가율, b는 형상모수, c는 척도모수를 의미한다.

2장에서 제안된 DPF1 모형(Lee 외, 2020)의 $b(t)$는 $\frac{b}{b+ce^{-bt}}$이며, 제안하고자 하는 DPF2는 분모의 모수 b를 고장검출률이 높은 경우를 고려하여 $b=1$인 경우로 제한한다. 이는 고장이 발생하는 비율이 높은 데이터에 적합한 모형이다. <그림 4-1>은 고장검출률 $b(t)=\frac{b}{1+ce^{-bt}}$와 $b(t)=\frac{b}{b+ce^{-bt}}$의 모수 b의 따른 변화를 나타낸 그래프이다. 굵은 진한 선 계열은 제안하고자 하는 DPF2 모형의 $b(t)$이다. 모수 b가 1보다 작은 0.2, 0.5인 경우 $b(t)$가 $\frac{b}{b+ce^{-bt}}$인 DPF1 모형의 경우보다 제안하는 DPF2 모형의 $b(t)$가 작은 값을 가지나, 모수 b가 1보다 큰 1.2, 1.5인 경우는 제안하는 DPF2 모형의 $b(t)$가 더 크게 나타났다. 고장데이터 중 발생하는 고장의 검출률이 높은 데이터인 경우, 즉, b가 1보다 큰 경우에는 제안하는 DPF2 모형의 $b(t)$가 기존에 제안된 DPF1 모형의 $b(t)=\frac{b}{b+ce^{-bt}}$인 경우보다 잘 적합한 것으로 나타났다.



<그림 4-1> 시간 t의 변화에 따른 $b(t)$

미분방정식 (9)에서 본 논문에서 제안된 $a(t)$와 $b(t)$를 통해 평균값함수 $m(t)$를 계산하면 다음과 같은 식을 따른다.

$$m(t)=\frac{(c+e^{bt})^{a}\left(\frac{c+e^{bt}}{c}\right)^{a\alpha t}e^{\frac{a\alpha Li_2\left(\frac{c+e^{bt}}{c}\right)}{b}}}{\int\frac{(c+e^{bx})^{a}\left(\frac{c+e^{bx}}{c}\right)^{a\alpha x}e^{\frac{a\alpha Li_2\left(\frac{c+e^{bx}}{c}\right)}{b}}be^{bx}}{c+e^{bx}}dx+(1+c)^{a}e^{\frac{a\alpha Li_2\left(\frac{c+1}{c}\right)}{b}}}$$

여기서 이중로그함수 Li_2는 다중로그함수 $Li_s(z)=z+z^2/2^s+z^3/3^s+\cdots$인 급수로 정의되며, $s=2$인 경우로 제한한 형태인 다항식 $Li_2(x)=\sum_{n=1}^{\infty}\frac{x^n}{n^2}$을 따른다. 이때 결함 수의 증가율 α를 반영하지 않는 $a(t)$를 적용시키기 위해 $\alpha=0$으로 설정하고, 초기 고장 수를 $m(0)=h$으로 설정하여 $m(t)$를 유도하면, 평균값함수 $m(t)$는 다음

과 같이 얻어진다.

$$m(t)=\frac{h(c+e^{bt})^a}{h\left[\int\frac{(c+e^{bx})^a be^{bx}}{c+e^{bx}}dx\right]+(1+c)^a} \tag{10}$$

식 (10)을 간단히 정리하면 다음과 같다.

$$\begin{aligned} m(t)&=\frac{h(c+e^{bt})^a}{h\frac{(c+e^{bt})^a}{a}+(1+c)^a} \\ &=\frac{h(c+e^{bt})^a\cdot\frac{a}{h(c+e^{bt})^a}}{\left(h\frac{(c+e^{bt})^a}{a}+(1+c)^a\right)\cdot\frac{a}{h(c+e^{bt})^a}} \\ &=\frac{a}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{bt}}\right)^a} \end{aligned} \tag{11}$$

제안하는 DPF2 모형의 평균값함수 $m(t)$는 식 (11)과 같이 표현되며, 앞서 소개한 기존의 DPF1 모형과 매우 유사한 형태를 띄지만 고장검출률 함수인 $b(t)$ 구조를 변화시켜 시점마다 발생하는 고장검출률이 높을수록 더 적합한 특징을 나타낸다. 새로 제안된 모형을 통해 기존의 NHPP 소프트웨어 신뢰성 모형과의 적합도 비교를 통해 우수함을 입증하고자 한다.

다음으로, 본 논문에서는 새롭게 제안된 DPF2 모형을 통해 최적의 출시시점을 찾고자 한다. 소프트웨어를 출시하기 위해서는 테스트 단계에서 발생하는 비용이나 출시 후 유지 및 보수를 위해 발생할 수 있는 비용 등 모든 비용적인 측면을 통해 최적의 출시시점을 고려한다. 완성된 형태의 높은 신뢰성을 갖는 소프트웨어를 출시하는 것은 중요한 일이지만, 이는 현실적으로 매우 어렵다. 일반적으로 테스트에 많은 시간을 할애할수록 오류를 미리서 제거할 수 있기 때문에 소프트웨어의 신뢰성은 높아지지만, 그만큼 투입되는 테스트 비용도 증가하게 된다. 반대로 테스트 시

간을 적게 한다면 테스트 단계에 투입되는 비용은 적어지더라도 높은 신뢰성을 보장 받을 수 없다(Zhang과 Pham, 1998). 그리고 운용 단계에서 발생하는 소프트웨어 고장은 테스트 단계에서 투입되는 비용보다 더 많은 비용을 필요로 하기 때문에 완벽한 소프트웨어를 개발하여 출시하는 것은 매우 어려워, 어느 시점부터 테스트를 중단하고, 소프트웨어를 출시할 것인지를 결정하는 문제도 매우 중요한 문제이다. 이를 결정하기 위하여 본 논문에서는 NHPP 소프트웨어 신뢰성 모형을 기반으로 하는 비용모형을 활용하여 총 비용이 적게 발생할 때의 최적의 시점을 고려하고자 한다. 비용모형은 다음과 같은 가정을 만족한다(Pham, 2006).

1. 테스트 비용은 테스트 시간에 비례한다.
2. 테스트 단계에서 오류를 제거하는 데 드는 비용은 테스트 단계가 끝날 때까지 감지된 모든 오류를 제거하는 데 소요되는 총 시간에 비례한다.
3. 테스트 중 각 오류를 제거하는 데 걸리는 시간은 절단된 지수 분포를 따른다.
4. 각 출시시점의 신뢰성과 관련된 위험 비용은 존재한다.

이를 통해 구성되는 비용모형은 다음과 같다.

$$EC(T) = C_0 + C_1 T + C_2 m(T) + C_3(1 - R(x|T))$$

본 논문에서 적용하고자 하는 비용모형은 DPF2를 활용하여 소프트웨어 설치 비용부터 소프트웨어 테스트 비용, 운용 비용, 소프트웨어 제거 비용 및 소프트웨어가 발생할 때 위험 비용까지를 반영한 비용모형을 기준으로 최적의 시점을 제안한다. 여기서 C_0는 시스템 테스트를 위한 설치 비용, C_1은 단위 시간당 시스템 테스트 비용, C_2는 테스트 단계 (보증 기간) 동안 단위 시간당 오류 제거 비용, C_3는 시스템 장애로 인한 패널티 비용을 의미하며, x는 소프트웨어가 사용되는 시간을 의미한다. 또한, T는 소프트웨어 출시시간, $m(T)$는 발견될 기대 오류 수, $R(x|T)$는 소프트웨어 신뢰도 함수이며, $R(x|T)$는 아래와 같은 식을 따른다(Yamada와 Osaki, 1985; Singpurwalla, 1991).

$$R(x|T) = e^{-[m(T+x)-m(T)]}$$

이를 통해 DPF2를 활용한 비용모형은 다음과 같다.

$$m(T) = \frac{a}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{bT}}\right)^{a}}$$

$$R(x|T) = e^{-a\left[\frac{1}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{b(x+T)}}\right)^{a}} - \frac{1}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{bT}}\right)^{a}}\right]}$$

$$EC(T) = C_0 + C_1 T + C_2 \frac{a}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{bT}}\right)^{a}} + C_3\left(1-e^{-a\left[\frac{1}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{b(x+T)}}\right)^{a}} - \frac{1}{1+\frac{a}{h}\left(\frac{1+c}{c+e^{bT}}\right)^{a}}\right]}\right)$$

본 논문에서는 새롭게 제안된 종속 고장을 가정한 NHPP 소프트웨어 신뢰성 모형인 DPF2를 기반으로 데이터 세트1을 이용하여 비용모형을 제안하고, C_0부터 C_3까지의 비용계수의 변화를 통해 출시 시간과 최소 비용 간의 최적의 시점을 찾고자 한다.

제2절 딥러닝 소프트웨어 신뢰성 모형

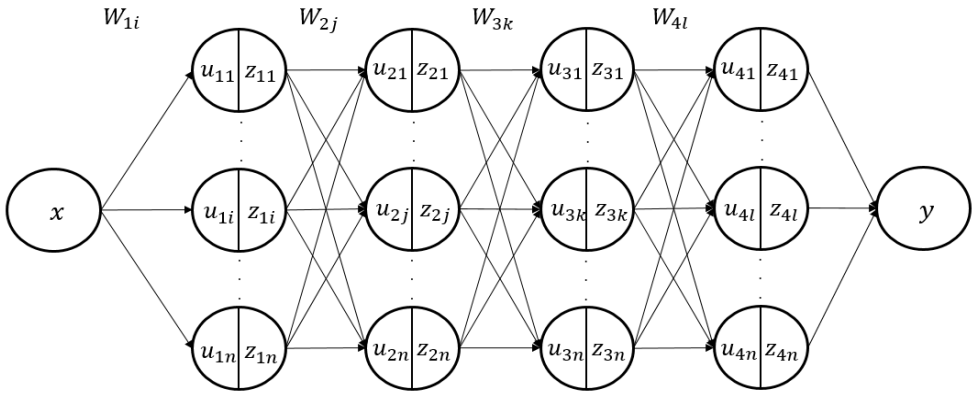
1. 딥러닝을 활용한 소프트웨어 신뢰성 모형

소프트웨어 신뢰성을 평가하기 위한 NHPP 소프트웨어 신뢰성 모형은 때로는 불합리한 경우가 존재한다. NHPP 소프트웨어 신뢰성 모형은 독립, 종속, 운용환경의 불확실성 등의 여러 가정을 통해 제안되지만, 많은 소프트웨어는 각기 다른 다양한 형태를 가지고 있기 때문에 여러 가정으로 정의한 소프트웨어 신뢰성 모형이 실제 환경과 맞지 않을 수 있다. 소프트웨어의 구조가 복잡해지고 소프트웨어의 쓰임새 역시 다양해지면서 하나의 가정만 포함하지 않고, 여러 가정이 함께 포함된 구조를 갖는 경우가 많아졌다. 단 하나의 가정으로 소프트웨어 신뢰성을 판단하기란 매우 어려우며, 소프트웨어의 특성에 맞는 모든 가정을 다 포함하는 것 역시도 어렵다. 또한, 여러 가정을 갖는 소프트웨어 신뢰성 모형은 가정이 적합한 경우에만 최적화되는 결과를 얻기 때문에 특수한 경우에만 잘 적합하는 어려움을 갖고 있다. 소프트웨어의 고장 발생에 대한 가정이 적절하다면 소프트웨어 신뢰성을 올바른 수학적·통계적 근거를 토대로 결과를 해석하고, 소프트웨어 신뢰성을 판단할 수 있을 것이다. 하지만 소프트웨어 고장에 대한 가정이 적절하지 않은 모형을 통해 소프트웨어 신뢰성을 계산하고 이를 통해 판단하면 소프트웨어에 의존하는 현재 우리 사회는 매우 큰 문제를 야기 시킬 수 있다. 따라서 수학적·통계적 가정을 기반으로 소프트웨어 신뢰성을 판단하는 모형이 아닌 얻어진 데이터를 기반으로 신뢰성을 판단할 수 있는 비모수적 방법 중 기계학습 방법을 활용한 소프트웨어 신뢰성 모형을 활용하여 소프트웨어 신뢰성 모형을 제안하고자 한다. 기계학습 방법 중에서도 딥러닝을 활용한 소프트웨어 신뢰성 모형을 제안하고자 한다.

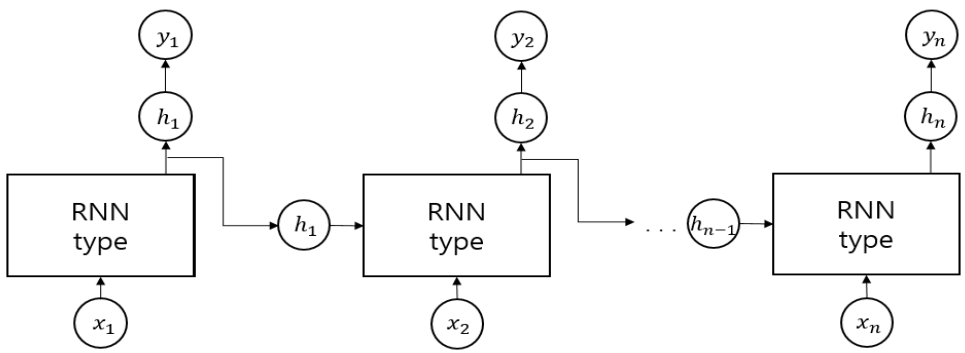
신경망은 인간이 사고하는 과정을 컴퓨터 알고리즘을 통해 구현시킨 방법으로 인간이 얻을 수 있는 정보들을 뇌까지 전달시켜 판단할 수 있도록 유도한다. 정보를 받는 역할인 입력층, 판단하는 출력층, 정보를 전달하는 은닉층 총 3가지 층으로 구성되며 정보가 입력층에서 은닉층으로, 은닉층에서 출력층으로 각 층에 있는 가중치와 편향들의 합과 곱의 결합을 통해 전달한다. 데이터가 얻어지면 이를 신경망 모형에 투입하여 입력층부터 출력층까지 전달하여 결과를 도출한다. 이는 데이터를 기반으로 학습하는 모형으로 데이터에 의존하여 최적의 결과를 이끌어낸다. 여기서 딥러닝은 깊게 학습할 수 있도록 정보를 전달하는 은닉층을 하나의 층으로 구성되는 것이 아닌 여러 층을 두어 정보를 여러 연산 과정을 거치며 전달하도록

한다. 다수의 은닉층으로 구성되는 모형은 DNN이다. DNN은 딥러닝에서 가장 기본이 되는 신경망으로 비선형 연산을 통해 학습하는 모형이다. RNN은 신경망 중에서 순차적 데이터에 적합하는 모형으로 시계열적 특성을 지닌 데이터를 학습하는데 많이 활용된다. LSTM은 RNN이 시점이 커질수록 과거 시점의 정보를 소실하는 문제를 해결하기 위해 만들어진 모형으로 기억셀을 필두로 과거 시점의 정보를 계속해서 포함하여 연산한다. 하지만 이 역시도 복잡한 구조에 의해 연산속도가 느리다는 단점이 존재하여 이를 개선한 GRU도 연구되었다. 이는 기울기 소실 문제도 해결하면서 구조도 LSTM 보다 단순한 형태로 구성되어 있다. 따라서 본 논구에서는 딥러닝을 활용한 소프트웨어 신뢰성 모형을 제안하고자 하며, 활용하는 딥러닝 기법은 가장 기본 형태인 DNN, 순차적 데이터에 적합한 RNN, LSTM, GRU이다. 따라서, 본 논문에서는 기존의 소프트웨어 신뢰성 모형들과 새롭게 제안하는 딥러닝을 활용한 소프트웨어 신뢰성 모형들의 비교 연구를 하고자 한다.

본 논문에서 활용하고자 하는 딥러닝을 활용한 소프트웨어 신뢰성 모형의 구조는 다음과 같다. <그림 4-2>는 DNN을 활용한 소프트웨어 신뢰성 모형으로 은닉층의 수를 4개로 하며, 그 안에 포함된 노드의 수는 학습에 활용되는 학습데이터 크기만큼으로 한다. <그림 4-3>은 순환신경망 계열인 RNN, LSTM, GRU의 구조이며, 학습에 활용되는 학습데이터 크기만큼 시점을 두어 이전 시점의 값이 다음 시점에 영향을 미치는 연산과정에 활용한다. 또한, 얻어진 데이터의 차분은 1로 두어 이전 차수의 값을 뺀 것을 기준으로 학습한다. 활용하고자 하는 모든 딥러닝을 활용한 소프트웨어 신뢰성 모형은 300번의 에포크(Epoch) 과정을 거쳐 학습하며, 모형을 학습하기 위해 활용하는 최적화 기법은 모멘텀과 Adagrad 방법이 혼용된 Adam 방법을 활용하여 모형의 가중치를 학습시켜 최적의 결과를 얻고자 한다.



<그림 4-2> 활용하는 심층신경망(DNN) 구조



<그림 4-3> 활용하는 순환신경망(RNN) 계열 구조

2. 딥러닝 소프트웨어 신뢰성 모형

앞절에서 소개한 딥러닝을 활용한 모형은 데이터에 의존하는 모형으로서 수학적·통계적 가정 없이 데이터에 최적화된 모형을 학습한다. 하지만 이는 데이터에 의존하는 만큼 새롭게 발생한 상황에 있어서 오류가 생길 수 있는 과적합 문제가 발생할 수 있다. 또한, 데이터에만 의존하여 학습시킨 모형이기 때문에 얻은 결과를 논리적 근거 없이 믿고 따를 수밖에 없다. 따라서 이를 개선하기 위해 데이터에 의존하는 딥러닝 모형에 수학적·통계적 가정을 추가한 모형을 제안하고자 한다.

일반적인 고장 발생은 고장이 점점 증가하면서 발생하다가 특정 시점을 지나면서 줄어드는 형태인 S자 형태(S-shaped)나 초기와 마지막 시점에 많이 발생하는 오목한 형태를 따른다. 지금까지 개발된 소프트웨어 신뢰성 모형은 소프트웨어 고장이 S자 형태나 오목한 형태로 발생할 것을 가정한 모형들이다. 이 중 S자 형태는 딥러닝의 활성화함수 중 시그모이드 함수와 매우 유사하다. 시그모이드 함수는 t가 $-\infty$일 때 0에 수렴, t가 ∞일 때 1에 수렴, t가 0일 때 0.5를 갖는 S자형 곡선을 갖는 함수이다.

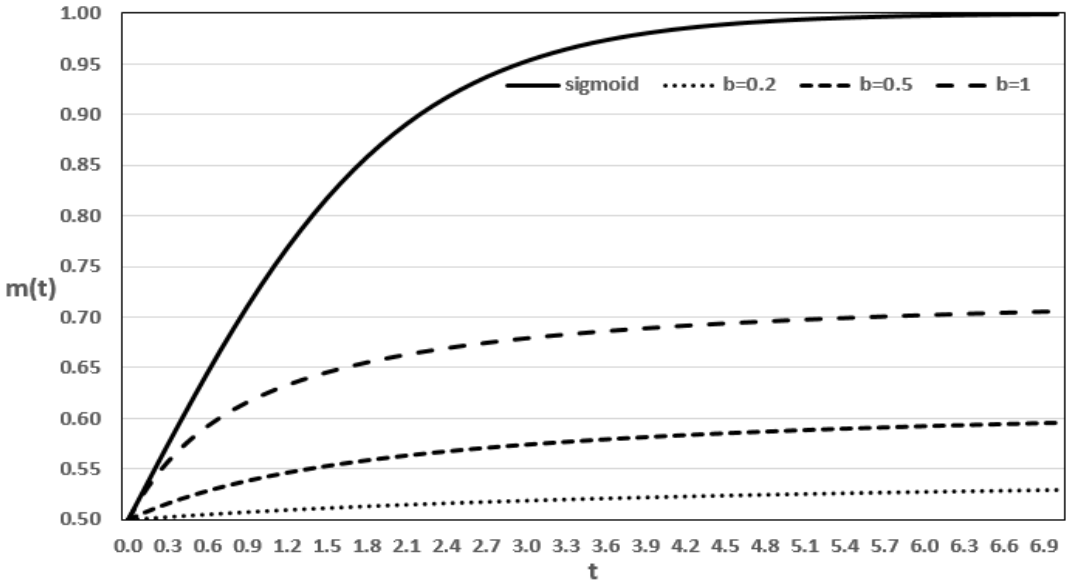
$$f(t)=\frac{1}{1+\exp(-t)}$$

본 논문에서는 활성화함수 중 시그모이드 함수가 소프트웨어 신뢰성 모형의 형태와 가장 비슷하기 때문에 이를 활용하며, 입력값으로 투입되는 값이 단순 고장 시점이 아닌 소프트웨어 신뢰성 모형의 고장검출률을 의미하는 $b(t)$ 함수가 포함된 일반화 소프트웨어 신뢰성 모형을 제안하고자 한다. 여기서 고장검출률 함수 $b(t)$는 $\frac{b^2t}{bt+1}$를 활용하며, 시점이 증가할 때마다 고장이 발생할 확률이 변하는 구조를 띤다. 따라서 최종 모형은 수식 (12)를 따른다.

$$m(t)=\frac{1}{1+\exp\left(-\frac{b^2t}{bt+1}\right)} \tag{12}$$

<그림 4-4>는 기존 시그모이드 함수와 변형 시그모이드 함수에서 $b(t)$의 모수 b 값이 변할 때, 시간 t에 따른 각 함수의 변화를 보여준다. 기존 시그모이드 함수는

1에 수렴하는 형태를 가지며 이는 연산과정에서 값이 크면 1만 반영되는 형태이기 때문에 변화가 크지 않거나 계속 같은 값만을 출력한다. 하지만 변형된 형태인 활성화 함수를 활용하면 계속 같은 값만을 출력하지 않고, 연속적인 데이터에 대한 학습 과정에서 다양한 값을 얻을 수 있다. 또한, 데이터에 의존하면서 수학적·통계적 가정을 포함하고 있는 모형이기 때문에 고장검출률 함수 $b(t)$의 형상모수인 b의 변화에 따라 변형 시그모이드 함수가 얻을 수 있는 값이 다름을 확인할 수 있다. 고장검출률이 큰 경우, 즉, b가 큰 값인 경우 이에 따른 변형 시그모이드 함수의 값이 더 크다. 고장검출률을 작게 하는, b가 작은 경우는 변형 시그모이드 함수가 0.5보다 약간 큰 값에서 큰 변화가 없는 형태로 나타난다. 따라서 변형 시그모이드 함수를 통해 고장검출률이 큰 데이터 특성을 지니는 경우 변형 시그모이드 함수를 통해 다양한 값을 얻어 학습하는데 활용한다.



<그림 4-4> 시간 t의 변화에 따른 시그모이드 함수와 변형 시그모이드 함수 결과

앞서 언급한 것처럼 해당 모형은 시그모이드의 변형된 형태이다. 본 논문에서는 제안된 모형을 딥러닝의 잘 알려진 시그모이드나 ReLU 활성화함수를 사용하는 대신 변형된 형태의 제안하는 소프트웨어 신뢰성 모형으로 변환하여 기존의 소프트웨어 신뢰성 모형과의 비교 연구에 활용하고자 한다. 이를 통해 데이터에 의존하는 모형만이 아닌 수학적·통계적 가정을 함께 적용한 딥러닝 소프트웨어 신뢰성 모형(Deep learning software reliability growth model; DL-SRGM)을 제안하고자 한다. 제안하는 DL-SRGM은 앞절에서 소개한 딥러닝을 활용한 소프트웨어 신뢰성 모형 중 DNN을 활용한 모형의 구조를 기반으로 하며, <그림 4-2>의 형태를 따른다. DL-SRGM의 구조는 은닉층의 수를 4개로 하며, 그 안에 포함된 노드의 수는 학습에 활용되는 학습데이터 크기만큼으로 한다. 층에서 층으로 전개되는 단계에서 활용되는 활성화 함수는 시그모이드 함수가 아닌 변형 시그모이드 함수를 활용하며, 다음과 같은 식을 통해 전개된다.

$$u_{1i} = b_1 + \sum_{i=1}^{n} W_{1i} x,$$

$$z_{1i} = \frac{1}{1 + \exp\left(-\frac{b^2 u_{1i}}{b u_{1i} + 1}\right)}$$

또한, 300번의 에포크 과정을 거쳐 학습하며, 모형을 학습하기 위해 활용하는 최적화 기법은 모멘텀과 Adagrad 방법이 혼용된 Adam 방법을 활용하여 모형의 가중치를 학습시켜 최적의 결과를 얻고자 한다.

제5장 수치적 예제

제1절 데이터 소개

본 논문에서 제안하고자 하는 모형의 비교를 위한 데이터 세트는 총 세 가지로, 이를 통해 제안하는 모형의 우수함을 입증하고자 한다. 첫 번째 데이터 세트는 ABC 소프트웨어 회사의 온라인 커뮤니케이션 시스템(On-line communication system)에서 12주간 발생한 고장데이터이다. 해당 프로젝트는 2000년에 완료되었으며(Pham, 2006), 유닛 관리자 1명, 사용자 인터페이스 소프트웨어 엔지니어 1명, 소프트웨어 엔지니어/테스터 10명으로 구성되어 있다. 해당 데이터는 12주 동안 수집되었으며, 오류 탐지는 개발 및 테스트 팀에서 문제의 심각한 정도에 따라 심각한 문제(Severe 1), 주요 문제(Severe 2), 경미한 문제(Severe 3) 세 가지 범주로 세분화하여 관리하였다. 본 논문에서는 이를 통합한 고장데이터를 활용하며 제안하고자 하는 종속 고장을 고려한 NHPP 소프트웨어 신뢰성 모형의 우수함을 입증하는데 활용하고자 한다.

두 번째 데이터 세트는 Apache open source software(OSS)를 사용하는 Hive에서 발생한 고장데이터를 포함한다(Singh와 Sharma, 2014). Apache Hive는 대규모 분석을 가능하게 하는 분산형 내결함성 데이터 웨어하우스 시스템이다. Hive는 Apache Hadoop을 기반으로 구축되며 hdf를 통해 스토리지를 지원한다. Hive를 사용하면 SQL을 사용하여 페타바이트 단위의 데이터를 읽고, 쓰고, 관리할 수 있다. 해당 데이터 세트는 2008년부터 2014년까지 발생한 결함 및 고장에 대한 정보를 포함하며 본 논문에서 딥러닝을 활용한 소프트웨어 신뢰성의 우수함을 입증하는데 활용하고자 한다.

세 번째 데이터 세트는 IoTDB에서 2019년 1월부터 2022년 1월까지 발생한 월별 고장데이터이다. IoTDB는 산업용 애플리케이션에서 IoT 센서의 타임스탬프 데이터와 같은 대량의 시계열 데이터를 관리하기 위한 데이터 저장소이다(https://iotdb.apache.org). Apache IoTDB(사물인터넷용 데이터베이스)는 데이터 관리 및 분석을 위한 고성능을 갖춘 IoT 네이티브 데이터베이스로, 엣지 및 클라우드에 배포할 수 있다. 경량 아키텍처, 고성능 및 풍부한 기능 세트와 함께 Apache Hadoop, Spark 및 Flink와의 긴밀한 통합으로 인해 Apache IoTDB는 IoT 산업 분야의 대용량 데이터 저장, 고속 데이터 수집 및 복잡한 데이터 분석 요구사항을 충족할 수 있다. Apache IoTDB를 운용하면서 Bug, New feature, Improvement,

Task 등으로 발생하는 결함 및 고장을 기록한 데이터로 딥러닝 소프트웨어 신뢰성 모형에 대한 추정 및 예측에 활용하고자 한다. <표 5-1>, <표 5-2>, <표 5-3>은 데이터 세트1, 2, 3을 설명한다.

총 세 가지 데이터 세트 중 첫 번째 데이터 세트는 종속 고장을 가정한 소프트웨어 신뢰성 모형에 적합 시키고자 하며, 두 번째와 세 번째 데이터 세트는 딥러닝을 활용한 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성에 추정 및 예측에 활용하고자 한다. 두 번째와 세 번째 데이터 세트는 전체 데이터 세트 중 90%는 학습 및 추정에 활용하며, 나머지 10%는 예측에 활용한다.

<표 5-1> 데이터 세트1 (OCS)

Week	Severe 1	Severe 2	Severe 3	Total
1	4	7	10	21
2	1	5	2	29
3	0	0	4	33
4	1	4	6	44
5	1	4	6	55
6	10	15	8	88
7	4	6	4	102
8	1	5	3	111
9	1	1	1	114
10	3	7	6	130
11	0	0	1	131
12	0	1	4	136

<표 5-2> 데이터 세트2 (Hive OSS)

Month	Failure	Cum.fail	Month	Failure	Cum.fail	Month	Failure	Cum.fail
1	49	49	22	36	859	43	35	1,634
2	38	87	23	40	899	44	70	1,704
3	38	125	24	35	934	45	60	1,764
4	64	189	25	27	961	46	53	1,817
5	50	239	26	19	980	47	53	1,870
6	42	281	27	22	1,002	48	78	1,948
7	35	316	28	42	1,044	49	119	2,067
8	28	344	29	47	1,091	50	31	2,098
9	45	389	30	36	1,127	51	47	2,145
10	53	442	31	46	1,173	52	89	2,234
11	41	483	32	48	1,221	53	83	2,317
12	32	515	33	43	1,264	54	118	2,435
13	28	543	34	24	1,288	55	143	2,578
14	23	566	35	27	1,315	56	91	2,669
15	35	601	36	37	1,352	57	90	2,759
16	29	630	37	37	1,389	58	89	2,848
17	42	672	38	33	1,422	59	57	2,905
18	30	702	39	45	1,467	60	142	3,047
19	43	745	40	40	1,507	61	76	3,123
20	44	789	41	34	1,541			
21	34	823	42	58	1,599			

<표 5-3> 데이터 세트3 (Apache)

TIME	Bug	New feature	Improvement	Total
1	0	0	2	2
2	0	3	2	5
3	0	5	3	8
4	0	5	5	10
5	0	5	5	10
6	0	6	5	11
7	0	6	5	11
8	0	6	7	13
9	0	8	9	17
10	0	9	11	20
11	0	10	12	22
12	0	11	17	28
13	0	12	19	31
14	0	12	25	37
15	0	12	28	40
16	0	12	29	41
17	0	21	40	61
18	0	26	41	67
19	0	27	43	70
20	0	30	46	76
21	1	41	58	100
22	1	43	60	104
23	1	47	66	114
24	4	48	68	120
25	4	48	71	123
26	6	51	75	132
27	10	53	84	147
28	18	58	95	171
29	24	63	102	189
30	28	64	110	202
31	34	69	118	221
32	41	76	123	240
33	57	96	144	297
34	66	105	157	328
35	89	113	176	378
36	104	121	200	425

제2절 적합도

종속 고장을 고려한 새로운 소프트웨어 신뢰성 모형과 딥러닝을 활용한 소프트웨어 신뢰성 모형, 딥러닝 소프트웨어 신뢰성 모형의 우수성을 입증하기 위해 앞 절에서 소개한 데이터를 기반으로 모형의 추정값 및 예측값을 계산한다. 또한, 기존에 개발된 NHPP 소프트웨어 신뢰성 모형의 추정 및 예측값을 계산하여 제안하고자 하는 모형의 우수성을 보이고자 하며 여기서 사용되는 척도는 총 10가지이다 (Pham, 2006; Askari 외, 2021, Jeske와 Zhang, 2005; Pham, 2020).

각각의 척도는 실제값 y와 추정값 $\widehat{m}(t)$ 간의 차이를 기반으로 한다. 평균제곱오차(Mean squared error; MSE)는 각 시점마다 추정된 고장 수 $\widehat{m}(t_i)$와 실제 고장 수 y_i간의 차이를 제곱 합으로 계산한 후 데이터 수 n과 모수의 수 m의 차$(n-m)$로 나눠주는 것으로 정의한다. 평균절대오차(Mean absolute error; MAE)는 관측 수를 고려한 예측된 고장 수 $\widehat{m}(t_i)$와 관찰 값 y_i 사이의 차이를 절댓값의 합으로 측정한다. 예측비 위험(Predictive ratio risk; PRR)은 모형 추정과 관련하여 예측 고장 수 $\widehat{m}(t_i)$에서 실제 고장 수 y_i까지의 거리를 예측값 $\widehat{m}(t_i)$으로 나눠주는 것으로 측정한다. 예측력(Predictive power; PP)은 실제 고장 수 y_i에서 예측 고장 수 $\widehat{m}(t_i)$까지의 거리를 실제값 y_i으로 나눠주는 것으로 측정한다. R^2은 데이터의 변화를 설명하는 데 성공적으로 적합이 됐는지를 측정한 척도이다. 회귀분석에서는 설명력으로도 쓰인다. 평균 제곱 오차의 근(Root mean square prediction error; RMSPE)은 관측 수 $\widehat{m}(t)$를 예측하는 모형의 근접성(Closeness)을 추정한다. 평균 예측 오류(Mean error of prediction; MEOP)는 실제 관측 수 y_i와 추정 곡선 $\widehat{m}(t_i)$ 간의 편차의 절댓값의 합으로 측정한다. Theil 통계량(Theil statistic; TS)은 실제 관측값 y_i에 대한 모든 기간의 평균 편차 비율이다. PC(Pham's criterion)는 모형의 불확실성과 모형의 파라미터 수 사이의 상충관계를 고려한다. 예측 잔차 제곱 합(Predictive SSE; preSSE)은 예측 결과값 $\widehat{m}(t_i)$과 실제값 y_i의 차이의 합을 나타낸다(Li와 Pham, 2017). 이는 추정값 간의 비교가 아닌 추정된 모형을 통해 예측값을 계산하여 비교하는 척도이다. <표 5-4>는 활용된 적합도 척도에 대한 수식이다.

종속 고장을 고려한 새로운 소프트웨어 신뢰성 모형은 NHPP 소프트웨어 신뢰성 모형으로 모수의 수가 반영된 9가지의 적합도를 기준으로 비교하며, 딥러닝을 활용한 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형과의 비교를 위한 적

합도는 추정하고자 하는 모수의 수를 제외하고 비교한다. 모형 비교를 위한 적합도 척도는 R^2이 1에 가까울수록, 다른 적합도 척도는 0에 가까울수록 적합도가 우수함을 나타낸다. 또한, preSSE는 예측에 관한 적합도 척도이기 때문에 딥러닝을 활용한 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형에서 10%에 해당하는 데이터를 통해 preSSE를 계산하여 비교하며, 0에 가까울수록 좋은 결과를 나타낸다.

R과 Matlab을 활용하여 NHPP 소프트웨어 신뢰성 모형은 LSE 방법을 통해 각 모형의 모수를 추정하고, 딥러닝을 활용한 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형은 R에서 keras 패키지의 함수의 딥러닝 방법을 통해 에포크를 300번으로 제한하여 데이터를 학습한다. 추정한 모형을 통해 적합도를 계산하여 우수성을 비교하고자 한다.

<표 5-4> 적합도 척도

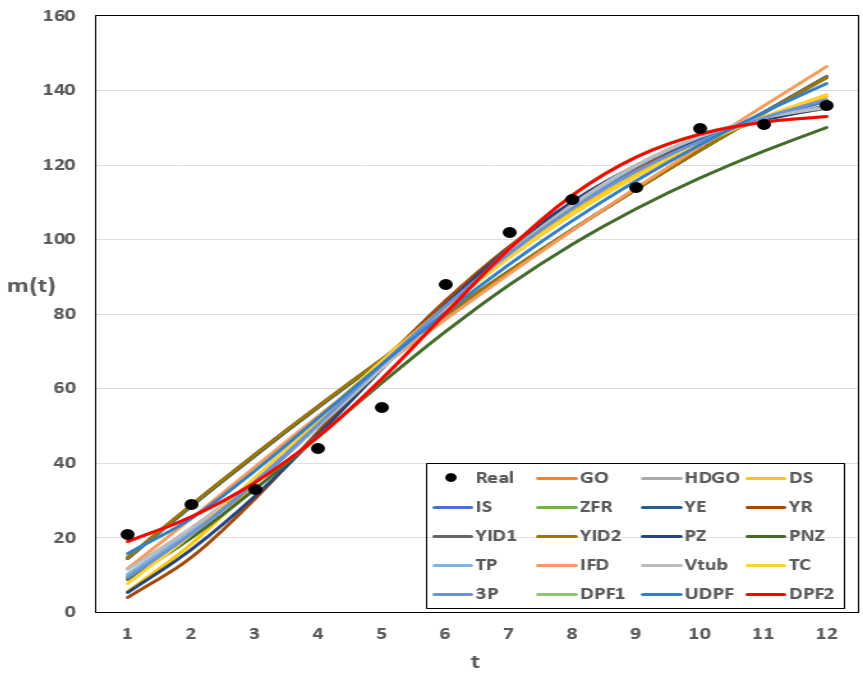
Criteria	Formula
MSE	$\frac{\sum_{i=1}^{n}(\hat{m}(t_i)-y_i)^2}{n-m}$
MAE	$\frac{\sum_{i=1}^{n}\left\|\hat{m}(t_i)-y_i\right\|}{n-m}$
PRR	$\sum_{i=1}^{n}\left(\frac{\hat{m}(t_i)-y_i}{\hat{m}(t_i)}\right)^2$
PP	$\sum_{i=1}^{n}\left(\frac{\hat{m}(t_i)-y_i}{y_i}\right)^2$
R^2	$1-\frac{\sum_{i=1}^{n}\left(y_i-\hat{m}(t_i)\right)^2}{\sum_{i=1}^{n}\left(y_i-\bar{y}\right)^2}$
RMSPE	$\sqrt{Variation^2+Bias^2}$, $Variation=\sqrt{\frac{\sum_{i=1}^{n}\left(\left(\hat{m}(t_i)-y_i\right)-Bias\right)^2}{n-1}}$, $Bias=\sum_{i=1}^{n}\left(\frac{\hat{m}(t_i)-y_i}{n}\right)$
MEOP	$\frac{\sum_{i=1}^{n}\left\|\hat{m}(t_i)-y_i\right\|}{n-m+1}$
TS	$100^*\sqrt{\frac{\sum_{i=1}^{n}\left(\hat{m}(t_i)-y_i\right)^2}{\sum_{i=1}^{n}y_i^2}}$
PC	$\left(\frac{n-m}{2}\right)\log\left(\frac{\sum_{i=1}^{n}\left(\hat{m}(t_i)-y_i\right)^2}{n}\right)+m\left(\frac{n-1}{n-m}\right)$
preSSE	$\sum_{i=1}^{n}(\hat{m}(t_i)-y_i)^2$

제3절 모형 비교

1. 종속 고장을 고려한 NHPP 소프트웨어 신뢰성 모형 비교

가. 모형 추정 결과 및 모형 비교

<표 5-5>는 데이터 세트1을 통해 얻은 각 모형의 모수에 대한 추정값을 나타낸다. 제안하는 종속 모형의 각 모수는 $\hat{a}=134.0947$, $\hat{b}=0.09953$, $\hat{c}=41.8672$, $\hat{h}=15.9670$으로 나타났다. <그림 5-1>은 데이터 세트1의 각 시점의 누적 고장 수와 각 모형 식을 기반으로 각 시점의 $m(t)$의 추정값을 계산한 결과이다. 검은색 점선은 실제 데이터를 나타내고, 진한 빨간 실선은 제안하는 모형의 각 시점의 고장 추정값을 나타낸다. 다른 모형에 비해 실제값에 가장 가까운 추정값을 나타냈다. <표 5-6>은 데이터 세트1을 통해 얻어진 모수를 통한 각 모형의 적합도 척도를 계산한 결과이다. MSE, $RMSPE$, TS, PC는 30.919, 4.742, 4.892, 17.604로 가장 작은 값을, R^2은 0.9882로 가장 큰 값을 보였다. 또한, PRR, PP, MAE, $MEOP$는 0.070, 0.067, 5.573, 4.954로 DPF1 다음으로 작은 적합도를 보였다. 이를 통해 데이터 세트1에 대해서 제안하고자 하는 모형이 우수함을 보였다.



<그림 5-1> $\hat{m}(t)$ 추정(데이터 세트1)

<표 5-5> 소프트웨어 신뢰성 모형에 대한 모수 추정값(데이터 세트1)

No.	Model	Estimated value			
1	GO	$\hat{a}=403.8259$	$\hat{b}=0.03668$		
2	HDGO	$\hat{a}=403.8259$	$\hat{b}=0.03668$	$\hat{c}=1.35426$	
3	DS	$\hat{a}=161.4689$	$\hat{b}=0.28594$		
4	IS	$\hat{a}=147.6179$	$\hat{b}=0.37282$	$\hat{\beta}=5.71882$	
5	ZFR	$\hat{a}=22.6281$	$\hat{b}=0.25548$	$\hat{\alpha}=30.9109$	$\hat{\beta}=0.0000427$
		$\hat{c}=12.9109$	$\hat{p}=0.16459$		
6	YE	$\hat{a}=404.0825$	$\hat{\alpha}=2.33982$	$\hat{\beta}=0.0000249$	$\hat{\gamma}=629.6753$
7	YR	$\hat{a}=196.9994$	$\hat{\alpha}=4.06772$	$\hat{\beta}=0.02992$	$\hat{\gamma}=0.32537$
8	YID1	$\hat{a}=356.6652$	$\hat{b}=0.04192$	$\hat{\alpha}=0.002783$	
9	YID2	$\hat{a}=378.2602$	$\hat{b}=0.03934$	$\hat{\alpha}=0.00114$	
10	PZ	$\hat{a}=113.5751$	$\hat{b}=0.42841$	$\hat{\alpha}=2.11324$	$\hat{\beta}=7.43123$
		$\hat{c}=28.8211$			
11	PNZ	$\hat{a}=92.6184$	$\hat{b}=0.40235$	$\hat{\alpha}=0.04756$	$\hat{\beta}=3.79017$
12	TP	$\hat{a}=2442.731$	$\hat{b}=0.25023$	$\hat{\alpha}=7.85582$	$\hat{\beta}=0.42208$
		$\hat{c}=1223.388$	$\hat{p}=108.710$	$\hat{q}=91.0487$	
13	IFD	$\hat{a}=25.9849$	$\hat{b}=0.39029$	$\hat{d}=0.00000048$	
14	Vtub	$\hat{a}=2.32007$	$\hat{b}=0.64841$	$\hat{\alpha}=51.0865$	$\hat{\beta}=778.8182$
		$\hat{N}=137.3928$			
15	TC	$\hat{a}=0.06883$	$\hat{b}=1.48911$	$\hat{\alpha}=25.7474$	$\hat{\beta}=9.74100$
		$\hat{N}=163.053$			
16	3P	$\hat{a}=1.05300$	$\hat{b}=0.38698$	$\hat{\beta}=1.02670$	$\hat{c}=19.7890$
		$\hat{N}=166.698$			
17	DPF1	$\hat{a}=133.8837$	$\hat{b}=0.09883$	$\hat{c}=42.7876$	$\hat{h}=15.8403$
18	UDPF	$\hat{b}=0.33101$	$\hat{\alpha}=1.49714$	$\hat{\beta}=1.30679$	$\hat{N}=263.059$
19	DPF2	$\hat{a}=134.0947$	$\hat{b}=0.09953$	$\hat{c}=41.8672$	$\hat{h}=15.9670$

<표 5-6> 소프트웨어 신뢰성 모형 적합도 척도 비교(데이터 세트1)

Model	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC
GO	76.257	0.356	0.318	0.9637	8.326	8.417	7.651	8.590	22.959
HDGO	84.730	0.356	0.318	0.9637	8.326	9.352	8.417	8.590	22.350
DS	68.367	8.488	0.777	0.9675	7.872	7.383	6.712	8.133	22.413
IS	53.162	1.789	0.462	0.9772	6.589	7.057	6.351	6.804	20.252
ZFR	74.696	1.378	0.409	0.9787	6.379	10.223	8.763	6.585	21.861
YE	95.322	0.356	0.318	0.9637	8.326	10.521	9.352	8.590	22.107
YR	93.001	21.083	0.984	0.9646	8.199	8.736	7.765	8.485	22.009
YID1	84.959	0.349	0.323	0.9636	8.337	9.300	8.370	8.601	22.362
YID2	84.839	0.353	0.320	0.9637	8.331	9.333	8.400	8.595	22.356
PZ	87.951	9.472	0.793	0.9707	7.459	9.276	8.116	7.718	21.639

<표 5-6 (계속)> 소프트웨어 신뢰성 모형 적합도 척도 비교(데이터 세트1)

Model	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC
PNZ	138.706	2.159	0.519	0.9472	9.822	12.893	11.460	10.362	23.608
TP	89.855	1.321	0.401	0.9786	6.386	12.379	10.316	6.593	24.457
IFD	86.191	0.785	0.363	0.9631	8.395	9.822	8.839	8.663	22.427
Vtub	58.695	0.852	0.331	0.9805	6.109	8.544	7.476	6.305	20.224
TC	87.258	3.077	0.579	0.9709	7.447	10.618	9.291	7.688	21.612
3P	69.918	1.742	0.461	0.9767	6.666	9.300	8.138	6.881	20.836
DPF1	30.958	**0.068**	**0.066**	**0.9882**	4.745	**5.569**	**4.950**	4.895	17.609
UDPF	62.944	0.229	0.202	0.976	6.766	8.900	7.911	6.980	20.447
DPF2	**30.919**	0.070	0.067	**0.9882**	**4.742**	5.573	4.954	**4.892**	**17.604**

나. 비용모형

제 4장 제 2절에서 소개한 DPF2를 활용한 비용모형을 통해 C_0부터 C_3까지의 비용계수의 변화를 통해 출시 시간과 최소 비용 간의 최적의 시점을 찾고자 한다. 비용계수 C_0는 시스템 테스트를 위한 설치 비용으로 설치하는 데만 비용이 소비되기 때문에 총 비용에는 영향을 미치나 출시시점에는 영향을 미치지 않는 패턴을 보이고, 비용계수 C_1은 테스트 단계에서 발생하는 비용으로 단위 시간당 비용이 증가하면 총 비용이 증가하지만 출시시점은 당겨지는 패턴을 보인다. 비용계수 C_2는 테스트 단계에서 발생한 오류를 제거하기 위한 비용으로 단위 시간당 비용이 증가하면 출시시점이 당겨지는 패턴을 보이며, 비용계수 C_3은 시스템 장애로 인한 패널티 비용으로 비용이 증가할수록 총 비용이 증가하며, 출시시점도 늘어나는 패턴을 보인다(Williams, 2007; Song 외, 2017b). 이러한 패턴을 통해 DPF2 모형을 활용한 비용모형에서 비용계수 값의 변화에 따라 나타나는 총 비용과 출시시점의 변화 패턴을 비교해보고자 한다.

해당 비용모형의 모수는 수치적 예제의 데이터 세트1을 통해 계산된 $\hat{a}=134.0947$, $\hat{b}=0.09953$, $\hat{c}=41.8672$, $\hat{h}=15.9670$을 사용하고, 비용모형의 비용계수는 여러 값의 변화를 통해 가장 최적의 값을 찾아 비용이 가장 적게 드는 최적의 출시시점 T^*을 찾는 것을 목표로 한다. 데이터 세트1의 단위가 주 단위이므로 시점 또한 동일한 단위를 갖는다. 비용계수의 기준값은 $C_0=500$, $C_1=20$, $C_2=30$, $C_3=5000$을 기본으로 하고, x는 5를 고정한다. 이때의 총비용 $EC(T)=4873.120$이고, 그때의 출시시점 T^*는 16.9주이다.

<표 5-7>은 비용계수 C_0과 C_1에 대한 비용과 출시시점에 대한 결과이다. 비용계수 C_0은 설치 비용으로서 값이 증가하는 만큼 비용이 정비례하는 관계이므로 설치 비용이 적을수록 비용값이 최소값을 띈다. 비용계수 C_0는 100, 300, 500, 700, 1000일 때의 변화를 비교한다. C_0가 100일 때 총 비용은 4473.120, 300일 때 총 비용은 4673.120, 500일 때 총 비용은 4873.120, 700일 때 총 비용은 5073.120, 1000일 때 총 비용은 5373.120으로 나타났으며 이때의 출시시점은 16.9주로 동일하게 나타났다. C_0 값이 클수록 총 비용은 증가하지만, 최적의 시점은 변화하지 않기 때문에 적정 실치비용 C_0를 설정하여야 한다.

비용계수 C_1은 5, 10, 20, 30, 40일 때의 변화를 비교한다. C_1이 5일 때 총 비용은 4614.805, 출시시점은 17.8주, 10일 때 총 비용은 4702.461, 출시시점은

17.3주로 나타났고, 20일 때 총 비용은 4873.120, 출시시점은 16.9주, 30일 때 총 비용은 5040.078, 출시시점은 16.6주, 40일 때 총 비용은 5204.616, 출시시점은 16.3주로 나타났다. 단위 시간당 시스템 테스트 비용을 증가시키면 총비용은 증가하지만, 출시시점이 조금씩 앞당겨지는 결과를 보였으며, 기존의 패턴과 동일한 패턴을 보였다.

<표 5-7> 비용계수 C_0와 C_1 변화에 대한 최적의 출시시점과 총 비용

$C_1=20,\ C_2=30,\ C_3=5000$			$C_0=500,\ C_2=30,\ C_3=5000$		
C_0	T^*	$EC(T^*)$	C_1	T^*	$EC(T^*)$
100	16.9	4473.120	5	17.8	4614.805
300	16.9	4673.120	10	17.3	4702.461
500	16.9	4873.120	20	16.9	4873.120
700	16.9	5073.120	30	16.6	5040.078
1000	16.9	5373.120	40	16.3	5204.616

<표 5-8>은 비용계수 C_2과 C_3에 대한 비용과 출시시점에 대한 결과이다. 비용계수 C_2는 20, 25, 30, 35, 40일 때의 변화를 비교한다. C_2가 20일 때 총 비용은 3532.198, 25일 때 총 비용은 4202.659, 30일 때 총 비용은 4873.120, 35일 때 총 비용은 5543.581, 40일 때 총 비용은 6084.725로 나타났으며 C_2의 출시시점은 16.9로 동일하게 나타났다. C_2은 C_0와 같이 값이 클수록 총 비용은 증가하지만, 최적의 출시시점은 16.9주로 변화하지 않는 것으로 나타났다. 이는 기존의 패턴과는 다른 패턴을 나타냈다.

비용계수 C_3는 5000, 7000, 10000, 12000, 15000일 때의 변화를 비교한다. C_3가 5000일 때 총 비용은 4873.120, 출시시점은 16.9주, 7000일 때 총 비용은 4877.520, 출시시점은 17.1주, 10000일 때 총 비용은 4882.121, 출시시점은 17.3주, 12000일 때 총 비용은 4884.450, 출시시점은 17.5주, 15000일 때 총 비용은 4887.204, 출시시점은 17.6주로 나타났으며, 기존의 패턴과 동일한 패턴을 보였다.

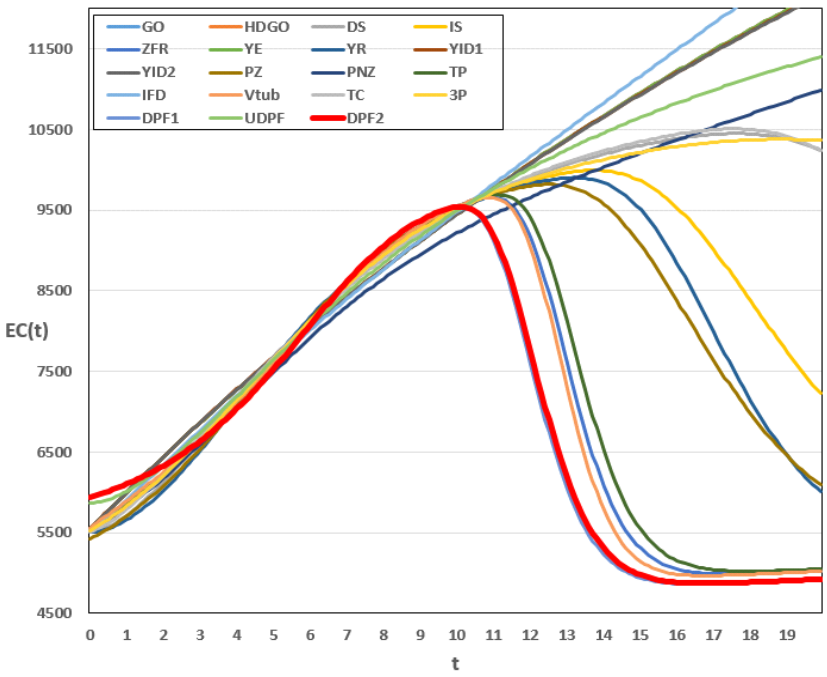
<표 5-8> 비용계수 C_2와 C_3 변화에 대한 최적의 출시시점과 총 비용

$C_0 = 500,\ C_1 = 20,\ C_3 = 5000$			$C_0 = 500,\ C_1 = 20,\ C_2 = 30$		
C_2	T^*	$EC(T^*)$	C_3	T^*	$EC(T^*)$
20	16.9	3532.198	5000	16.9	4873.120
25	16.9	4202.659	7000	17.1	4877.520
30	16.9	4873.120	10000	17.3	4882.121
35	16.9	5543.581	12000	17.5	4884.450
40	16.9	6214.043	15000	17.6	4887.204

<표 5-9>는 비용계수를 기준값으로 고정시킨 후, 모형 비교에 활용된 모형에 적용시킨 결과이며, <그림 5-2>는 각 모형에서 시점마다 발생하는 총 비용에 관한 결과이다. <표 5-9>에서 제시하지 않은 모형은 테스트 시작 단계에서 발생하는 총 비용보다 시간이 흐르면서 발생하는 총 비용이 크기 때문에 총 비용이 최소가 되는 최적의 출시시점을 제시하지 못하여 제외시켰다. ZFR 모형은 17.5주에서 총 비용이 4988.111로 가장 작은 비용값을 보였고, TP 모형은 18.0 시점에서 총 비용이 5023.403으로 가장 작은 비용값을 보였며, Vtub 모형은 16.8 시점에서 총 비용이 4968.494로 가장 작은 비용값을 보였다. DPF1 모형은 16.6 시점에서 1862.163으로 가장 작은 비용값을 보였으며 DPF2 모형은 16.9시점에서 4873.120으로 가장 작은 비용값을 보였다. 전체적으로 DPF1과 DPF2 모형이 출시시점도 빠르고, 총 비용도 적게 나타나는 모습을 보였다. 이는 모형의 형태가 유사하기 때문에 비슷한 결과가 나타났다.

<표 5-9> 소프트웨어 신뢰성 모형의 비용모형에 대한 출시시점과 총 비용

Model	T^*	$EC(T^*)$
ZFR	17.5	4988.111
TP	18.0	5023.403
Vtub	16.8	4968.494
DPF1	16.6	4862.163
DPF2	16.9	4873.120



<그림 5-2> 소프트웨어 신뢰성 모형의 비용모형에 대한 총 비용 비교

2. 딥러닝을 활용한 소프트웨어 신뢰성 모형 비교

가. 모형 추정 결과 및 모형 비교

해당 절에서는 제4장 제2절의 딥러닝 소프트웨어 신뢰성 모형 중 딥러닝을 활용한 소프트웨어 신뢰성 모형을 활용하여 데이터 세트2를 통해 다른 NHPP 소프트웨어 신뢰성 모형보다 우수함을 입증하고자 한다. <표 5-10>은 데이터 세트2를 사용하여 NHPP를 가정한 소프트웨어 신뢰성 모형의 모수 추정 결과와 딥러닝을 활용한 소프트웨어 신뢰성 모형의 구조를 나타낸다. 전체 데이터 중 90%만 추출하여 NHPP를 가정한 소프트웨어 신뢰성 모형의 모수를 추정하고, 딥러닝을 활용한 소프트웨어 신뢰성 모형을 학습하였다. 나머지 10%에 해당하는 데이터는 추정된 모형을 기반으로 한 예측값을 비교하였다. 딥러닝을 활용한 소프트웨어 신뢰성은 DNN, RNN, LSTM, GRU 총 4가지 모형을 모두 구성하여 기존 NHPP 소프트웨어 신뢰성 모형과 비교하였다. DNN은 층을 총 4개로 하였고, 각각의 노드 수는 학습하는 데이터 수와 동일하게 구성하였으며 학습률 α는 0.000000001로 설정하였다. 순환신경망 계열의 RNN, LSTM, GRU는 모두 차분을 1로 하여 반영되는 시점을 이전 한 시점으로 구성하였다. 각각의 모형의 학습률 α는 RNN에서 0.0000001, LSTM에서 0.000000001, GRU에서 0.000001로 설정하였다. 모든 딥러닝을 활용한 소프트웨어 신뢰성 모형은 Adam 최적화 기법을 활용하여 모형을 학습하였다.

<표 5-11>은 데이터 세트2를 통해 얻은 추정된 각 모형의 적합도 척도를 계산한 결과이다. 딥러닝을 활용한 소프트웨어 신뢰성 모형이 기존 NHPP 소프트웨어 신뢰성 모형을 통해 얻은 적합도 척도보다 우수한 결과를 보였다. 딥러닝을 활용한 소프트웨어 신뢰성 모형 중에서도 순환신경망 계열의 모형이 DNN보다 더 우수한 결과를 보였다. 이는 데이터 특성상 시계열 데이터이므로 시간의 흐름이 반영된 모형에 잘 적합 하는 모습을 보였다. 순환신경망 계열의 모형 중 모수의 수가 가장 많은 LSTM을 활용한 모형의 결과가 10개의 모든 적합도 척도에 대해 우수한 결과값을 보였다. 과적합의 경우를 판단하기 위해 preSSE 값을 비교하여도 LSTM에서 가장 좋은 결과를 보였다. 또한, NHPP 소프트웨어 신뢰성 모형 중 가장 우수한 모형은 YID1 모형으로 나타났다. 앞절에서 소개한 DPF2 모형은 Kim 외(2022)에서 소개한 데이터 세트 3가지를 기반으로 추정한 결과는 매우 좋은 결과를 보였으나 데이터 세트2를 활용하였을 때는 좋은 결과를 보이지 않아, 해당 데이터 세트2가 종속 가정에는 적합하지 않은 데이터인 것으로 보인다. 그러나 본 절에서 연구의 목적은 기존의 다양한

소프트웨어 신뢰성 모형들과 딥러닝을 활용한 소프트웨어 신뢰성 모형의 결과를 비교하고자 했던 목적에 부합한 결과로서 딥러닝을 활용한 소프트웨어 신뢰성 모형들의 적합도가 우수함을 확인할 수 있으며, 특히 LSTM이 가장 우수한 결과를 보였다.

<표 5-10> 소프트웨어 신뢰성 모형에 대한 모수 추정값 및 구조(데이터 세트2)

No.	Model	Estimated value and Structure			
1	GO	$\hat{a}=183078.71$	$\hat{b}=0.000223$		
2	HDGO	$\hat{a}=709.4483$	$\hat{b}=0.15422$	$\hat{c}=0.41844$	
3	DS	$\hat{a}=3592.167$	$\hat{b}=0.039104$		
4	IS	$\hat{a}=87134.617$	$\hat{b}=0.011107$	$\hat{\beta}=28.8987$	
5	ZFR	$\hat{a}=1337.445$ $\hat{c}=3.20401$	$\hat{b}=0.005033$ $\hat{p}=0.018173$	$\hat{\alpha}=130.084$	$\hat{\beta}=0.20062$
6	YE	$\hat{a}=384498.63$	$\hat{\alpha}=2.46282$	$\hat{\beta}=0.000198$	$\hat{\gamma}=0.21793$
7	YR	$\hat{a}=3746.444$	$\hat{\alpha}=1.25145$	$\hat{\beta}=0.000658$	$\hat{\gamma}=1.16620$
8	YID1	$\hat{a}=563.559$	$\hat{b}=0.10526$	$\hat{\alpha}=0.031778$	
9	YID2	$\hat{a}=1918.758$	$\hat{b}=0.016432$	$\hat{\alpha}=0.033505$	
10	PZ	$\hat{a}=397461.29$ $\hat{c}=82.1088$	$\hat{b}=0.011104$	$\hat{\alpha}=93.1859$	$\hat{\beta}=137.562$
11	PNZ	$\hat{a}=26384.460$	$\hat{b}=0.010056$	$\hat{\alpha}=0.002922$	$\hat{\beta}=7.16190$
12	TP	$\hat{a}=216.557$ $\hat{c}=0.040215$	$\hat{b}=0.009098$ $\hat{p}=0.10587$	$\hat{\alpha}=0.000152$ $\hat{q}=0.10588$	$\hat{\beta}=0.000009$
13	IFD	$\hat{a}=81.3328$	$\hat{b}=0.33064$	$\hat{d}=0.009539$	
14	Vtub	$\hat{a}=1.000019$ $\hat{N}=15648.313$	$\hat{b}=0.17332$	$\hat{\alpha}=742.279$	$\hat{\beta}=9.99242$
15	TC	$\hat{a}=0.004013$ $\hat{N}=238625.45$	$\hat{b}=1.13601$	$\hat{\alpha}=10.9063$	$\hat{\beta}=200.241$
16	3P	$\hat{a}=0.025557$ $\hat{N}=18499.916$	$\hat{b}=0.020425$	$\hat{\beta}=4.52564$	$\hat{c}=2.04304$
17	DPF1	$\hat{a}=4697.396$	$\hat{b}=0.00000022$	$\hat{c}=0.019983$	$\hat{h}=293.837$
18	UDPF	$\hat{b}=0.002917$	$\hat{\alpha}=0.67414$	$\hat{\beta}=0.67379$	$\hat{N}=36246.346$
19	DPF2	$\hat{a}=4694.518$	$\hat{b}=0.0000239$	$\hat{c}=1.16421$	$\hat{h}=293.886$
20	DNN	hidden layer = 4, optimizer = Adam, α=0.000000001			
21	RNN	difference = 1, optimizer = Adam, α=0.0000001			
22	LSTM	difference = 1, optimizer = Adam, α=0.000000001			
23	GRU	difference = 1, optimizer = Adam, α=0.000001			

<표 5-11> 소프트웨어 신뢰성 모형 적합도 척도 비교(데이터 세트2)

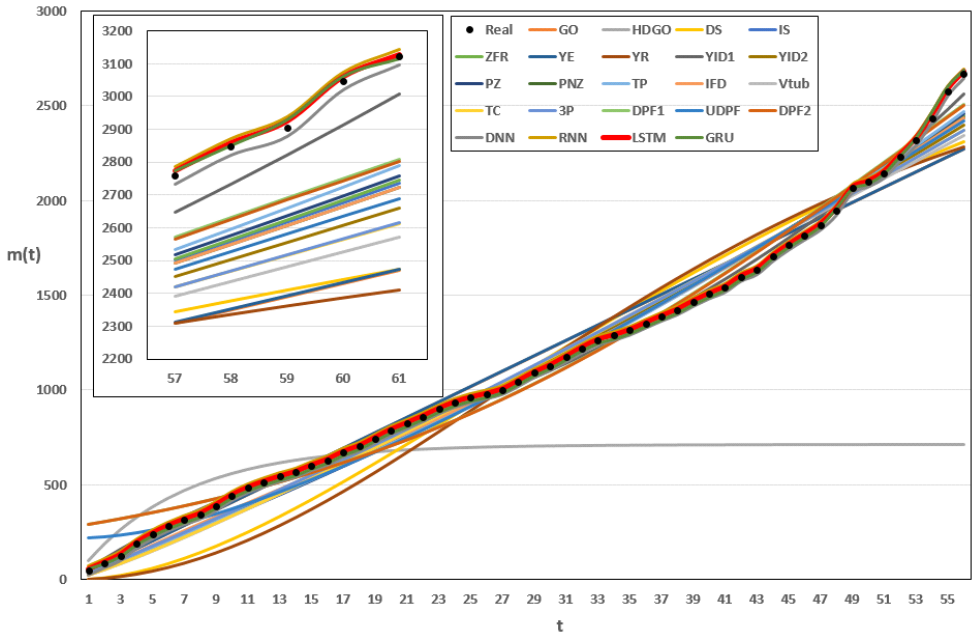
Model	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC	preSSE
GO	11132.2	0.355	0.316	0.976	106.455	74.947	73.632	7.900	261.875	1515391
HDGO	621131.8	70.494	15.315	0.001	792.230	572.876	562.825	59.010	374.483	43199548
DS	21447.6	412.638	6.527	0.954	147.646	126.470	124.251	10.966	280.237	1429503
IS	5871.9	1.694	0.968	0.987	77.283	63.327	62.216	5.738	243.964	521659
ZFR	5694.3	1.602	0.928	0.988	76.110	62.429	61.334	5.650	243.104	499445.4
YE	11214.2	0.352	0.317	0.976	106.843	75.739	74.410	7.929	262.080	1508404
YR	29035.4	943.487	8.106	0.938	171.750	146.945	144.367	12.759	288.718	1707054
YID1	1078.2	0.163	0.233	0.998	33.133	25.728	25.277	2.459	196.508	64118.01
YID2	6931.5	1.808	1.010	0.985	83.981	67.967	66.774	6.234	248.609	742754.2
PZ	5900.5	1.927	1.066	0.987	77.462	63.865	62.744	5.752	244.100	457488.3
PNZ	5963.4	1.634	0.942	0.987	77.891	63.832	62.712	5.782	244.397	553336
TP	5309.6	1.622	0.938	0.989	73.487	60.495	59.434	5.456	241.146	389830.9

<표 5-11 (계속)> 소프트웨어 신뢰성 모형 적합도 척도 비교(데이터 세트2)

Model	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC	preSSE
IFD	4855.5	1.082	0.641	0.990	70.298	55.487	54.513	5.217	238.642	555370.7
Vtub	9071.6	4.143	1.573	0.981	96.081	77.654	76.292	7.132	256.143	1059259
TC	8162.1	3.273	1.393	0.983	91.143	74.156	72.855	6.765	253.185	908459.5
3P	7214.1	1.076	0.690	0.985	85.702	66.798	65.626	6.360	249.728	895940.7
DPF1	7178.4	2.165	34.143	0.985	85.484	69.353	68.136	6.344	249.589	315085.9
UDPF	6757.8	1.625	16.166	0.986	82.949	65.212	64.068	6.155	247.899	649215.4
DPF2	7166.5	2.168	34.142	0.985	85.418	68.839	67.631	6.339	249.543	331986.6
DNN	651.4	1.526	0.488	0.999	25.522	25.522	25.074	1.911	182.396	3255.271
RNN	498.7	0.215	0.363	0.999	22.335	22.133	21.745	1.672	174.918	3435.082
LSTM	**52.83**	**0.031**	**0.038**	**0.999**	**7.282**	**6.482**	**6.368**	**0.544**	**112.062**	**851.6471**
GRU	279.4	0.055	0.052	0.999	16.791	15.722	15.446	1.252	158.697	1223.945

나. 예측 및 신뢰구간

<그림 5-3>은 90% 데이터를 기반으로 추정한 모형의 추정값과 10%에 대한 예측값을 보여준다. HDGO는 데이터의 추세선에 매우 많이 벗어나 예측값에 해당하는 그래프에는 포함시키지 않았으며, 딥러닝을 활용한 소프트웨어 신뢰성 모형은 데이터 추세선에 매우 잘 적합한 형태로 추정 및 예측하였다.



<그림 5-3> $\hat{m}(t)$ 추정 및 예측(데이터 세트2)

<표 5-12>는 추정된 소프트웨어 신뢰성 모형의 95% 예측 신뢰구간을 계산하여 나머지 10%에 해당하는 데이터가 95% 예측 신뢰구간 안에 포함되는지의 여부를 판단한 결과이다. 소프트웨어 고장은 포아송 분포를 따르기 때문에 평균과 분산은 각각 $\hat{m}(t)$로 간주하며, z_α는 표준 정규 분포의 $100(1-\alpha)$ 백분위 수로 정의된다. 따라서 $100(1-\alpha)$% 신뢰구간은 다음과 같은 형태를 갖는다(Pham, 2006).

$$\hat{m}(t) \pm z_{\alpha/2}\sqrt{\hat{m}(t)}$$

모든 NHPP 소프트웨어 신뢰성 모형은 추정된 결과를 기반으로 한 95% 예측 신뢰구간 안에 10% 데이터가 포함되는 모형이 없었고, 모든 딥러닝을 활용한 소프트웨어 신뢰성 모형에서는 10% 데이터가 95% 예측 신뢰구간 안에 포함되었다. 이는 데이터에 적합한 모형이기 때문에 증가하는 추세를 잘 예측한 것으로 판단된다.

<표 5-12> $\widehat{m}(t)$에 대한 95% 예측 신뢰구간(데이터 세트2)

index	Real	GO			DS			IS		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2310.61	2216.40	2404.83	2343.61	2248.72	2438.49	2500.70	2402.69	2598.72
58	2848	2350.89	2255.86	2445.92	2376.95	2281.39	2472.50	2558.50	2459.36	2657.64
59	2905	2391.16	2295.31	2487.00	2409.56	2313.35	2505.78	2616.86	2516.60	2717.13
60	3047	2431.41	2334.77	2528.06	2441.47	2344.62	2538.32	2675.79	2574.41	2777.18
61	3123	2471.66	2374.22	2569.11	2472.67	2375.20	2570.13	2735.30	2632.79	2837.81
index	Real	ZFR			YE			YR		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2505.75	2407.64	2603.86	2312.06	2217.81	2406.30	2309.28	2215.09	2403.47
58	2848	2564.36	2465.11	2663.61	2352.26	2257.20	2447.32	2335.77	2241.05	2430.50
59	2905	2623.62	2523.22	2724.01	2392.46	2296.59	2488.33	2361.25	2266.00	2456.49
60	3047	2683.53	2581.99	2785.06	2432.64	2335.97	2529.31	2385.72	2289.99	2481.46
61	3123	2744.10	2641.42	2846.77	2472.81	2375.34	2570.27	2409.23	2313.02	2505.43
index	Real	YID1			YID2			PZ		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2647.50	2546.65	2748.35	2452.17	2355.11	2549.23	2517.39	2419.05	2615.73
58	2848	2733.13	2630.66	2835.60	2503.73	2405.65	2601.80	2576.95	2477.45	2676.45
59	2905	2821.50	2717.39	2925.61	2555.49	2456.40	2654.57	2637.15	2536.50	2737.81
60	3047	2912.72	2806.94	3018.50	2607.45	2507.37	2707.54	2698.01	2596.20	2799.82
61	3123	3006.87	2899.39	3114.34	2659.62	2558.54	2760.70	2759.53	2656.57	2862.49

<표 5-12 (계속)> $\widehat{m}(t)$에 대한 95% 예측 신뢰구간(데이터 세트2)

index	Real	PNZ			TP			IFD		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2493.47	2395.60	2591.35	2533.07	2434.42	2631.72	2491.82	2393.98	2589.66
58	2848	2550.19	2451.21	2649.17	2595.80	2495.94	2695.66	2548.99	2450.03	2647.94
59	2905	2607.40	2507.32	2707.48	2659.53	2558.46	2760.61	2606.67	2506.60	2706.74
60	3047	2665.11	2563.93	2766.30	2724.30	2622.00	2826.60	2664.86	2563.68	2766.04
61	3123	2723.33	2621.05	2825.61	2790.14	2686.61	2893.67	2723.57	2621.28	2825.86
index	Real	Vtub			TC			3P		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2390.82	2294.98	2486.66	2418.97	2322.57	2515.37	2421.46	2325.02	2517.91
58	2848	2436.10	2339.36	2532.84	2466.96	2369.61	2564.31	2469.69	2372.29	2567.10
59	2905	2481.37	2383.74	2579.00	2515.06	2416.76	2613.35	2518.01	2419.66	2616.37
60	3047	2526.62	2428.10	2625.14	2563.25	2464.02	2662.49	2566.41	2467.12	2665.71
61	3123	2571.84	2472.44	2671.24	2611.55	2511.39	2711.71	2614.89	2514.67	2715.12
index	Real	DPF1			UDPF			DPF2		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2570.55	2471.17	2669.92	2473.91	2376.42	2571.40	2563.96	2464.71	2663.20
58	2848	2630.85	2530.32	2731.38	2527.15	2428.62	2625.68	2624.17	2523.76	2724.57
59	2905	2690.78	2589.11	2792.45	2580.57	2481.00	2680.14	2684.01	2582.47	2785.56
60	3047	2750.26	2647.47	2853.04	2634.16	2533.56	2734.75	2743.42	2640.76	2846.08
61	3123	2809.20	2705.32	2913.09	2687.91	2586.30	2789.53	2802.30	2698.55	2906.06

<표 5-12 (계속)> $\widehat{m}(t)$에 대한 95% 예측 신뢰구간(데이터 세트2)

index	Real	DNN			RNN			LSTM		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
57	2759	2733.48	**2631.01**	**2835.96**	2785.59	**2682.15**	**2889.04**	2772.64	**2669.43**	**2875.84**
58	2848	2822.48	**2718.35**	**2926.61**	2871.17	**2766.14**	**2976.19**	2857.70	**2752.92**	**2962.47**
59	2905	2879.48	**2774.31**	**2984.66**	2937.66	**2831.42**	**3043.89**	2924.51	**2818.51**	**3030.50**
60	3047	3021.48	**2913.75**	**3129.22**	3073.70	**2965.04**	**3182.37**	3060.13	**2951.70**	**3168.55**
61	3123	3097.48	**2988.40**	**3206.57**	3143.29	**3033.40**	**3253.18**	3127.34	**3017.73**	**3236.94**

index	Real	GRU		
		prediction	lower	upper
57	2759	2775.33	**2672.08**	**2878.59**
58	2848	2852.17	**2747.49**	**2956.84**
59	2905	2930.58	**2824.48**	**3036.69**
60	3047	3062.42	**2953.95**	**3170.88**
61	3123	3116.10	**3006.69**	**3225.51**

3. 딥러닝 소프트웨어 신뢰성 모형 비교

가. 모형 추정 결과 및 모형 비교

해당 절에서는 제4장 제2절에서 제안한 딥러닝 소프트웨어 신뢰성 모형으로 앞 절에서 제시한 NHPP 소프트웨어 신뢰성 모형들과 딥러닝을 활용한 소프트웨어 신뢰성 모형과의 비교를 통해 우수함을 입증하고자 한다. <표 5-13>은 데이터 세트 3을 사용하여 NHPP를 가정한 소프트웨어 신뢰성 모형의 모수 추정 결과와 딥러닝을 활용한 소프트웨어 신뢰성 모형, 딥러닝 소프트웨어 신뢰성 모형의 구조를 나타낸다. 전체 데이터 중 90%만 추출하여 NHPP를 가정한 소프트웨어 신뢰성 모형의 모수를 추정하고, 딥러닝을 활용한 소프트웨어 신뢰성 모형, 딥러닝 소프트웨어 신뢰성 모형을 학습하였다. 나머지 10%에 해당하는 데이터는 추정된 모형을 기반으로 한 예측값을 비교하였다. 활성화함수를 변환시킨 모형인 딥러닝 소프트웨어 신뢰성 모형은 은닉층 4개, 각각의 노드 수는 학습 데이터 수만큼 구성하였고, 학습률 α는 0.000001, 최적화 기법은 Adam을 활용하였다. 이를 통해 90%의 데이터를 활용하여 추정한 모수 b는 0.81로 나타났다.

<표 5-14>는 데이터 세트3을 통해 얻어진 추정된 각 모형의 적합도 척도를 계산한 결과이다. 딥러닝 소프트웨어 신뢰성 모형에 대한 MSE, R^2, $RMSPE$, MAE, $MEOP$, TS, PC는 16.043, 0.997, 4.005, 4.005, 3.888, 3.509, 46.762로 가장 적합한 결과를 보였다. 또한, 예측에 대한 척도인 $preSSE$는 48.134로 나타나, 딥러닝 소프트웨어 신뢰성 모형이 기존에 제안된 NHPP 소프트웨어 신뢰성 모형과 딥러닝을 활용한 소프트웨어 신뢰성 모형을 통해 얻은 적합도 척도보다 우수한 결과를 보였다. 앞절에서 소개한 DPF2 모형에 대한 결과와 마찬가지로 해당 데이터 세트3은 잘 적합하지 않은 모습을 보였으며, 데이터 세트3이 종속 가정에는 적합하지 않은 데이터 세트인 것으로 보인다. 그러나 본 절에서는 기존의 다양한 소프트웨어 신뢰성 모형들과 딥러닝 소프트웨어 신뢰성 모형의 결과를 비교하고자 했던 목적에 부합한 결과로서, 제안된 딥러닝을 활용한 소프트웨어 신뢰성 모형의 적합도가 우수한 결과를 나타냄을 확인할 수 있다.

<표 5-13> 소프트웨어 신뢰성 모형에 대한 모수 추정값 및 구조(데이터 세트3)

No.	Model	Estimated value and Structure			
1	GO	$\hat{a}=29665.362$	$\hat{b}=0.000187$		
2	HDGO	$\hat{a}=709.7827$	$\hat{b}=0.008520$	$\hat{c}=26.40058$	
3	DS	$\hat{a}=309921.28$	$\hat{b}=0.001215$		
4	IS	$\hat{a}=29625.705$	$\hat{b}=0.089533$	$\hat{\beta}=1933.363$	
5	ZFR	$\hat{a}=22.1649$ $\hat{c}=103.459$	$\hat{b}=0.069744$ $\hat{p}=0.17437$	$\hat{\alpha}=1442.274$	$\hat{\beta}=0.23978$
6	YE	$\hat{a}=6861.292$	$\hat{\alpha}=0.38719$	$\hat{\beta}=0.000027$	$\hat{\gamma}=77.8249$
7	YR	$\hat{a}=6634.205$	$\hat{\alpha}=0.27060$	$\hat{\beta}=0.000006$	$\hat{\gamma}=39.0121$
8	YID1	$\hat{a}=88.2434$	$\hat{b}=0.016888$	$\hat{\alpha}=0.091960$	
9	YID2	$\hat{a}=48.9940$	$\hat{b}=0.000699$	$\hat{\alpha}=13.11726$	
10	PZ	$\hat{a}=19640.559$ $\hat{c}=29.6669$	$\hat{b}=0.089741$	$\hat{\alpha}=275.047$	$\hat{\beta}=1287.636$
11	PNZ	$\hat{a}=1524.711$	$\hat{b}=0.049857$	$\hat{\alpha}=0.14599$	$\hat{\beta}=89.7911$
12	TP	$\hat{a}=148.509$ $\hat{c}=1.60499$	$\hat{b}=0.22531$ $\hat{p}=0.006426$	$\hat{\alpha}=0.54748$ $\hat{q}=0.69818$	$\hat{\beta}=5.69785$
13	IFD	$\hat{a}=0.37952$	$\hat{b}=0.020843$	$\hat{d}=23.6836$	
14	Vtub	$\hat{a}=1.01382$ $\hat{N}=13620.314$	$\hat{b}=1.40671$	$\hat{\alpha}=5.06585$	$\hat{\beta}=1371.070$
15	TC	$\hat{a}=0.001952$ $\hat{N}=20427.414$	$\hat{b}=2.43452$	$\hat{\alpha}=13.6666$	$\hat{\beta}=1.32591$
16	3P	$\hat{a}=0.69815$ $\hat{N}=31452.044$	$\hat{b}=0.089789$	$\hat{\beta}=4.25246$	$\hat{c}=3785.117$
17	DPF1	$\hat{a}=1161.079$	$\hat{b}=0.000114$	$\hat{c}=1.13177$	$\hat{h}=7.69249$
18	UDPF	$\hat{b}=0.19557$	$\hat{\alpha}=9.48098$	$\hat{\beta}=5.68418$	$\hat{N}=38790.461$
19	DPF2	$\hat{a}=1132.590$	$\hat{b}=0.000141$	$\hat{c}=0.36631$	$\hat{h}=7.60513$
20	DNN	hidden layer = 4, optimizer = Adam, α = 0.000000001			
21	RNN	difference = 1, optimizer = Adam, α = 0.00000001			
22	LSTM	difference = 1, optimizer = Adam, α = 0.000000001			
23	GRU	difference = 1, optimizer = Adam, α = 0.00000001			
24	DL-SRGM	$\hat{b}=0.81$, hidden layer = 4, optimizer = Adam, α = 0.000001			

<표 5-14> 소프트웨어 신뢰성 모형 적합도 척도 비교(데이터 세트3)

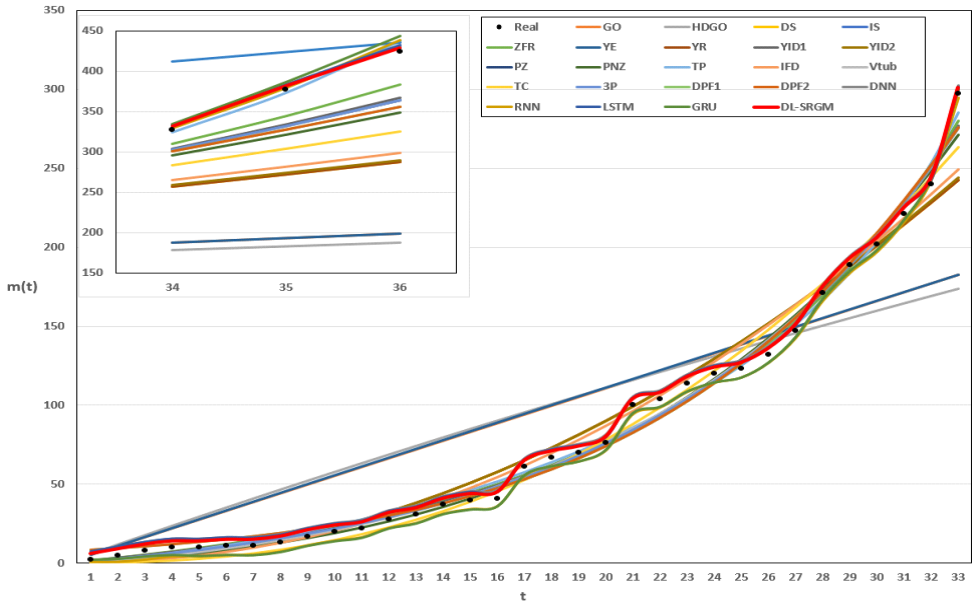
Model	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC	preSSE
GO	1316.794	7.115	45.647	0.785	36.797	29.757	28.882	31.791	119.489	104503.9
HDGO	1523.058	7.728	53.655	0.752	39.587	31.681	30.749	34.191	121.890	116771.6
DS	160.644	92.745	3.293	0.974	12.866	8.189	7.948	11.104	84.776	35032.12
IS	45.498	1.614	0.826	0.993	6.850	4.915	4.770	5.909	63.961	6329.658
ZFR	42.982	1.012	0.851	0.993	6.658	5.116	4.966	5.744	63.023	3060.905
YE	1332.715	7.175	46.582	0.783	37.018	29.982	29.100	31.983	119.687	105019
YR	162.450	93.798	3.312	0.974	12.938	8.257	8.014	11.166	84.961	35147.49
YID1	45.146	1.230	**0.786**	0.993	6.823	4.994	4.847	5.887	63.833	5724.79
YID2	156.961	74.245	3.226	0.974	12.716	8.126	7.887	10.976	84.394	33819.51
PZ	45.591	1.631	0.830	0.993	6.857	4.912	4.767	5.915	63.995	6474.618
PNZ	51.720	7.212	1.549	0.992	7.300	4.931	4.786	6.301	66.076	9954.81
TP	37.393	1.982	1.149	0.994	6.209	5.011	4.864	5.357	60.724	188.8706
IFD	120.169	131.593	3.313	0.980	11.130	6.678	6.481	9.604	79.987	28916.77

<표 5-14 (계속)> 소프트웨어 신뢰성 모형 적합도 척도 비교(데이터 세트3)

Model	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC	preSSE
Vtub	45.174	9.557	1.668	0.993	6.825	4.867	4.723	5.888	63.843	6508.331
TC	80.305	1728.406	5.001	0.987	9.093	6.783	6.584	7.851	73.336	17505.96
3P	45.603	1.649	0.827	0.993	6.857	4.884	4.741	5.916	63.999	6432.178
DPF1	58.322	1.580	12.992	0.991	7.754	6.189	6.007	6.691	68.058	7882.808
UDPF	57.819	**0.796**	6.138	0.991	7.722	4.977	4.831	6.662	67.915	9302.859
DPF2	58.285	1.548	12.595	0.991	7.751	6.169	5.987	6.689	68.048	8000.611
DNN	31.378	1.858	11.347	0.995	5.602	5.602	5.437	4.908	57.831	94.21431
RNN	26.042	29.466	3.151	0.996	5.112	4.992	4.845	4.471	54.755	198.6288
LSTM	23.435	1.759	10.822	0.996	4.842	4.808	4.667	4.241	53.015	101.3356
GRU	26.207	8.851	2.332	0.996	5.138	4.944	4.798	4.485	54.859	487.0868
DL-SRGM	**16.043**	1.280	5.803	**0.997**	**4.005**	**4.005**	**3.888**	**3.509**	**46.762**	**48.134**

나. 예측 및 신뢰구간

<그림 5-4>는 제안한 딥러닝 소프트웨어 신뢰성 모형을 포함한 24개의 모형에 대한 90% 데이터의 추정값과 10%에 대한 예측값을 보여준다. 데이터에 의존하는 딥러닝을 활용한 소프트웨어 신뢰성 모형, 딥러닝 소프트웨어 신뢰성 모형은 데이터 추세선에 매우 잘 적합한 형태로 추정 및 예측하였다. 이 중 제안하고자 하는 딥러닝 소프트웨어 신뢰성 모형이 가장 잘 적합하는 모습을 보였다.



<그림 5-4> $\hat{m}(t)$ 추정 및 예측(데이터 세트3)

<표 5-15>는 추정된 소프트웨어 신뢰성 모형의 95% 예측 신뢰구간을 계산하여 나머지 10%에 해당하는 데이터가 95% 예측 신뢰구간 안에 포함되는지의 여부를 판단한 결과이다. 앞 절에서 소개한 신뢰구간과 동일한 구조를 갖는다. 딥러닝을 활용한 소프트웨어 신뢰성 모형과 딥러닝 소프트웨어 신뢰성 모형은 10% 데이터가 95% 예측 신뢰구간 안에 포함되었다. NHPP 소프트웨어 신뢰성 모형 중 IS, YID1, PZ, PNZ, Vtub, 3P, DPF1, DPF2는 남은 10%의 데이터 중 첫 번째 시점의 데이터만 신뢰구간에 포함된 모형으로 나타났고, ZFR은 두 번째 시점까지 신뢰구간 안에 포함되었으며, TP는 세 번째 시점까지 신뢰구간 안에 모두 포함되었다.

<표 5-15> $\widehat{m}(t)$에 대한 95% 예측 신뢰구간(데이터 세트3)

index	Real	GO			HDGO			DS		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	188.17	161.29	215.06	178.52	152.33	204.70	257.12	225.69	288.55
35	378	193.69	166.41	220.97	183.02	156.51	209.54	272.25	239.91	304.59
36	425	199.21	171.54	226.87	187.49	160.65	214.33	287.80	254.55	321.05
index	Real	IS			ZFR			YE		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	303.05	**268.93**	**337.17**	310.67	**276.13**	**345.22**	187.79	160.93	214.65
35	378	332.52	296.78	368.26	345.10	**308.69**	**381.52**	193.24	165.99	220.48
36	425	364.68	327.25	402.11	384.03	345.62	422.44	198.67	171.05	226.30
index	Real	YR			YID1			YID2		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	257.11	225.68	288.53	304.41	**270.21**	**338.61**	258.79	227.26	290.33
35	378	272.10	239.77	304.43	334.60	298.75	370.45	274.14	241.69	306.60
36	425	287.50	254.26	320.73	367.69	330.10	405.27	289.93	256.56	323.30
index	Real	PZ			PNZ			TP		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	302.70	**268.60**	**336.80**	295.58	**261.89**	**329.28**	324.31	**289.01**	**359.61**
35	378	332.02	296.31	367.74	321.56	286.41	356.71	373.28	**335.42**	**411.15**
36	425	364.00	326.61	401.40	349.38	312.74	386.01	437.37	**396.38**	**478.36**

<표 5-15 (계속)> $\widehat{m}(t)$에 대한 95% 예측 신뢰구간(데이터 세트3)

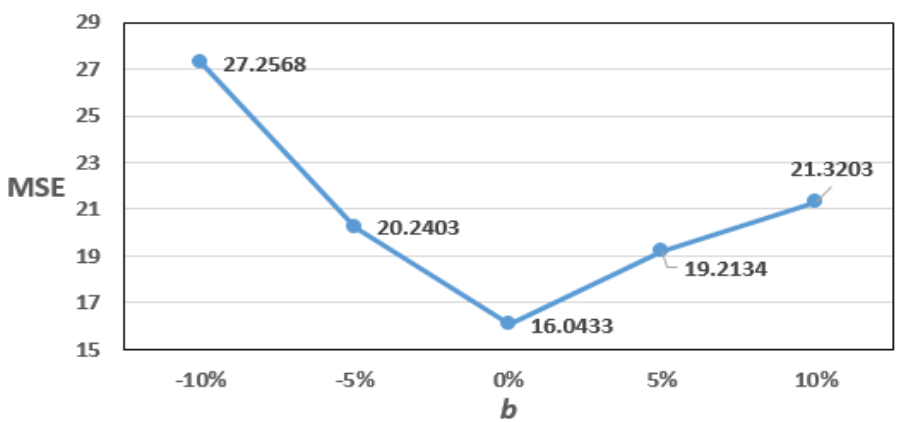
index	Real	IFD			Vtub			TC		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	265.43	233.50	297.37	302.34	**268.26**	**336.42**	283.18	250.20	316.17
35	378	282.10	249.18	315.02	331.79	296.09	367.50	303.72	269.56	337.88
36	425	299.28	265.38	333.19	364.05	326.65	401.45	325.10	289.76	360.44
index	Real	3P			DPF1			UDPF		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	302.69	**268.59**	**336.79**	301.40	**267.38**	**335.43**	412.05	372.26	451.83
35	378	332.14	296.42	367.86	328.23	292.72	363.74	424.02	383.66	464.38
36	425	364.27	326.86	401.68	356.46	319.45	393.46	436.00	**395.07**	**476.92**
index	Real	DPF2			DNN			RNN		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	301.24	**267.22**	**335.26**	333.60	**297.80**	**369.40**	329.15	**293.59**	**364.71**
35	378	327.90	292.41	363.39	383.60	**345.22**	**421.99**	379.45	**341.27**	**417.63**
36	425	355.90	318.93	392.88	430.60	**389.93**	**471.28**	438.97	**397.91**	**480.04**
index	Real	LSTM			GRU			DL-SRGM		
		prediction	lower	upper	prediction	lower	upper	prediction	lower	upper
34	328	332.32	**296.59**	**368.05**	335.03	**299.15**	**370.90**	332.01	**296.29**	**367.72**
35	378	383.29	**344.92**	**421.67**	386.95	**348.40**	**425.51**	382.01	**343.70**	**420.31**
36	425	432.39	**391.64**	**473.15**	443.91	**402.61**	**485.20**	429.01	**388.41**	**469.60**

다. 민감도 분석

소프트웨어 신뢰성 모형은 수학적·통계적 가정이 포함된 모수를 추정하여 고장수 $m(t)$를 추정하는 모형이다. 모수가 변화할 때마다 소프트웨어 신뢰성 모형의 $m(t)$ 값은 변하기 때문에 모수에 매우 민감하다. 또한, 데이터를 통해 추정한 $m(t)$의 모수는 얻어진 데이터를 기반으로 추정된다. 이 역시도 얻어진 데이터에 영향을 많이 받기 때문에 데이터의 특성을 파악해야 추정된 모수를 보다 잘 활용할 수 있을 것이다. 따라서 모수의 변화에 따른 민감도를 평가하기 위해 본 논문에서는 제안하는 모형의 모수 $\hat{b}=0.81$을 0.729(-10%), 0.7695(-5%), 0.8505(5%), 0.891(10%)만큼 변화시켜가면서 비교하고자 한다(Li 외, 2010). <그림 5-5>는 데이터 세트3에 대한 시점에 따라 변화된 모수에 대한 추정값을 보여준다. -10% ~ 10%까지는 모수 MSE가 변화하여도 $m(t)$의 변화폭은 크지 않다. 제안하는 딥러닝 소프트웨어 신뢰성 모형은 모수 b의 변화에도 좋은 결과를 얻는 것으로 나타났다. <표 5-16>은 b가 변함에도 잘 추정하는 모형들의 결과를 수치적으로 정확한 비교를 위해 데이터 세트3에 대한 추정값을 기반으로 한 MSE를 제안한 딥러닝 소프트웨어 신뢰성 모형의 MSE와 비교한 결과를 보여준다. 모수 b가 작아질수록 MSE가 기존에 추정한 값에 비해 1.699배(-10%), 1.262배(-5%)만큼 증가하는 모습을 보이며, 모수 b가 커질수록 MSE가 기존 추정한 값에 비해 1.198(5%), 1.329배(10%)만큼 증가하는 모습을 보였다. 이는 추정한 고장검출률 b이 작아질수록 MSE가 커지는 비율이 고장검출률 b가 커질수록 MSE가 커지는 비율보다 더 크게 증가하는 형태를 띠었다. 이는 데이터 세트3이 시점이 증가하더라도 고장 수가 줄지 않고, 고장이 발생하는 비율이 높은 데이터이기 때문인 것으로 나타났다.

<표 5-16> 모수 b 변화에 따른 DL-SRGM 적합도 결과

Change	MSE	PRR	PP	R^2	RMSPE	MAE	MEOP	TS	PC	preSSE
-10%	27.257	1.722	9.863	0.996	5.221	5.221	5.067	4.574	55.507	81.772
-5%	20.240	1.462	7.332	0.997	4.499	4.499	4.367	3.942	50.596	60.708
0%	16.043	1.280	5.803	0.997	4.005	4.005	3.888	3.509	46.762	48.134
5%	19.213	1.419	6.958	0.997	4.383	4.383	4.254	3.840	49.737	57.632
10%	21.320	1.504	7.714	0.997	4.617	4.617	4.482	4.045	51.454	63.963



<그림 5-5> 모수 b 변화에 따른 DL-SRGM MSE 변화

제6장 결론 및 제언

과거와 현재의 소프트웨어의 형태는 다르다. 과거 소프트웨어는 간단한 능력을 요구했거나 구조가 비교적 단순했다면, 현재의 소프트웨어는 복잡하고 다양한 형태로 존재하며 각 분야에서 많은 영향력을 끼친다. 이는 소프트웨어의 신뢰성을 바라보는 데도 개선되어야 할 부분이 많다는 것을 의미한다. 기존에 연구된 많은 소프트웨어 신뢰성 모형은 독립임을 가정하고 연구하였다. 하지만 복잡해진 현재 상황과는 맞지 않기 때문에 본 논문에서는 종속 고장을 가정한 소프트웨어 신뢰성 모형을 제안하였다. 또한, 4차 산업혁명으로 인공지능, 빅데이터 분야가 각광받고 있는 상황에서 관리되는 대부분 자료는 실시간으로 발생하는 자료이기 때문에 발생하는 데이터에 즉각적으로 반응하여 대응해야 한다. 그렇지 않으면 작은 연산부터 시작해서 잘못된 결과를 출력한다거나 오류가 날 경우 해당 산업의 막대한 피해를 입힐 수 있기 때문에 이는 매우 중요한 문제이다. 이러한 문제를 반영하여 데이터에 의존하는 딥러닝을 활용한 소프트웨어 신뢰성 모형을 제안하였으며, 더 나아가 단순 데이터에만 의존하지 않고, 수학적·통계적 가정을 포함한 활성화함수로 대체하는 딥러닝 소프트웨어 신뢰성 모형을 제안하였다.

첫 번째, 소프트웨어 고장은 종속적으로도 발생함을 가정한 종속 고장을 가정한 NHPP 소프트웨어 신뢰성 모형을 제안하였다. 12주간 발생한 데이터를 기반으로 모수 추정 및 총 9가지의 적합도 척도 비교 과정을 통해 우수함을 입증하였다. 총 9가지의 적합도 척도 중 5개의 척도가 가장 우수한 결과를 보였으며 나머지 4개의 척도에서는 두 번째로 좋게 나타났다. 예측에 대한 결과도 가장 좋은 것으로 나타났다. DPF1도 좋은 결과를 보였지만, 데이터 특성상 다시 발생하는 고장검출률의 비율이 높은 b를 가진 모형이 수학적 근거에 의해 적합한 것으로 판단된다. 얻어진 결과를 기반으로 비용모형에서 비용계수의 변화에 따른 최적의 출시시점을 제안하였고, 그에 맞는 총 비용을 분석하였다. 설치 비용 C_0와 오류 제거 비용 C_2는 비용이 증가한다고 출시시점이 당겨지거나 늘어나는 패턴을 보이지 않았고, 테스트 비용 C_1은 비용이 증가하면 출시시점이 당겨지는 모습 패턴을 보였으며, 시스템 장애로 인한 패널티 비용 C_3는 비용이 증가할수록 출시시점도 늘어나는 패턴을 보였다. 비용계수 C_2를 제외한 다른 비용계수는 기존과 동일한 패턴을 보였지만, 오류 제거 비용 C_2는 출시시점에 영향은 미치지 않는 것으로 나타나 기존의 비용모형의 패턴과는 다른 양상을 보였다.

두 번째, 데이터에 의존하는 딥러닝을 활용한 소프트웨어 신뢰성 모형을 제안하였다. 데이터에 의존하는 대표적인 방법 중 딥러닝 기법을 활용하였으며, 가장 기본이 되는 형태인 DNN, 시계열 데이터 분석에 특화된 순환신경망 계열의 RNN, LSTM, GRU를 모두 활용하였다. 61개월간 발생한 고장데이터의 90%만큼을 활용하여 모형을 추정하고, 나머지 10%에 대한 데이터를 통해 예측하였다. 9개의 추정에 대한 척도 비교와 1개의 예측에 대한 척도를 비교한 결과, 모든 척도에 대해 기존의 NHPP를 가정한 소프트웨어 신뢰성 모형보다 딥러닝을 활용한 소프트웨어 신뢰성 모형이 우수함을 보였으며 이중 LSTM이 가장 좋은 결과를 보였다. 또한, 10%에 대한 95% 예측 신뢰구간을 계산하여 실제값과의 차이를 비교하였으며 딥러닝을 활용한 소프트웨어 신뢰성 모형의 신뢰구간 안에는 다 포함됐으나 NHPP 소프트웨어 신뢰성 모형의 예측 신뢰구간에는 실제값이 포함되지 않았다. 데이터 기간이 길고, 발생한 고장 수가 많은 특징을 갖는 경우에는 본 논문에서 제안된 모형이 우수함을 보였다.

세 번째, 데이터에 의존하면서 수학적·통계적 가정이 추가된 딥러닝 소프트웨어 신뢰성 모형을 제안하였다. 딥러닝을 활용한 소프트웨어 신뢰성 모형은 수학적·통계적 근거 없이 데이터에만 의존해야 하기 때문에 논리적 근거가 부족할 수 있다. 이를 개선하기 위해 딥러닝의 활성화함수를 소프트웨어 신뢰성 모형으로 대체한 딥러닝 소프트웨어 신뢰성 모형을 제안하였다. 36개월간 발생한 고장데이터의 90% 만큼을 활용하여 소프트웨어 신뢰성 모형을 추정하고, 나머지 10% 데이터에 대한 예측값을 계산하여 실제값과 비교하였다. 9개의 추정에 대한 척도를 비교한 결과, 7개에 대한 적합도 척도가 가장 우수한 값을 보였으며, 예측에 대한 척도 역시 가장 좋은 결과를 보였다. 10%에 대한 95% 예측 신뢰구간을 계산하여 실제값과의 차이를 비교하였으며 딥러닝 소프트웨어 신뢰성 모형에 대해 좋은 결과가 나타났다. 또한, 딥러닝 소프트웨어 신뢰성 모형에 포함된 모수 b의 변화에 따라 적합도 척도의 변화를 비교하였으며, 데이터 특성상 시간의 흐름에 따라 소프트웨어 고장 발생이 큰 데이터이기 때문에 b가 0.81보다 커질수록 적합도 척도의 상승률이 적게 나타났고, 작을수록 적합도 척도의 상승률이 크게 나타났다.

본 논문에서는 현재 복잡해지는 소프트웨어에 맞는 가정과 방법 등을 기존 소프트웨어 신뢰성 모형을 개선해야 할 부분을 개선한 새로운 소프트웨어 신뢰성 모형을 제안하였고, 이는 소프트웨어 신뢰성 분야에서 매우 유용한 방법으로 활용될 것으로 판단된다. 또한, 과거 많은 딥러닝에 관한 연구는 알고리즘 개선에 기반한 연

구가 많이 진행되었다면, 본 논문을 통해 수학적·통계적 가정을 추가한 모형을 통해 단순 데이터에만 의존하는 모형이 아닌 계산한 결과가 왜, 어떻게 나타났는지에 대한 수학적·통계적 근거를 기반으로 해석 및 판단 근거를 제공할 것으로 보인다. 이를 확장하여, 복잡한 구조에 맞는 여러 가정이 추가된 더 포괄적인 딥러닝 소프트웨어 신뢰성 모형에 대한 연구가 필요할 것으로 보인다.

현재 소프트웨어 신뢰성 분야는 하루, 주, 월 단위 등 정리된 자료를 기반으로 고장을 예측하는 경우가 많다. 하지만 소프트웨어에 대한 의존도가 심해지는 만큼 즉각적으로 반응하고 대응해야 할 필요가 있다. 따라서 실시간 정보를 기반으로 하는 소프트웨어 고장에 관한 연구를 진행한다면 소프트웨어 신뢰성 분야에서 매우 활용가치가 높은 모형을 제안할 수 있을 것으로 사료된다.

참 고 문 헌

[1] Alven, W. (1964). Reliability engineering. Prentice-Hall.Iven.

[2] Askari, R., Sebt, M. V., & Amjadian, A. (2021). A multi-product EPQ model fo r defective production and inspection with single machine, and operational c onstraints: Stochastic programming approach. In Logistics and Supply Chain Management: 7th International Conference, LSCM 2020, Tehran, Iran, Decem ber 23-24, 2020, Revised Selected Papers 7 (pp. 161-193). Springer Internat ional Publishing.

[3] Barlow, R. E., & Proschan, F. (1965). Mathematical theory of reliability Wiley. New York.

[4] Batool, I., & Khan, T. A. (2022). Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature revie w. Computers and Electrical Engineering, 100, 107886.

[5] Box, G. E., Jenkins, G. M., & Reinsel, G. C. (2015). i GM Ljung, Time series analysis: forecasting and control.

[6] Cai, K. Y. (1998). On estimating the number of defects remaining in softwar e. Journal of Systems and Software, 40(2), 93-114.

[7] Chang, I. H., Pham, H., Lee, S. W., & Song, K. Y. (2014). A testing-coverage software reliability model with the uncertainty of operating environments. Inter national Journal of Systems Science: Operations & Logistics, 1(4), 220-227.

[8] Che, Z., Purushotham, S., Cho, K., Sontag, D., & Liu, Y. (2018). Recurrent n eural networks for multivariate time series with missing values. Scientific repo rts, 8(1), 6085.

[9] Das, S., Kundu, D., & Dewanji, A. (2022). Software reliability modeling based on NHPP for error occurrence in each fault with periodic debugging schedule. Communications in Statistics-Theory and Methods, 51(14), 4890-4902.

[10] Davis, D. J. F. (1952). An analysis of some failure data. Journal of the Amer ican Statistical Association, 47(258), 113-150.

[11] Epstein, B., & Sobel, M. (1954). Some theorems relevant to life testing from an exponential distribution. The Annals of Mathematical Statistics, 373-381.

[12] Goel, A. L., & Okumoto, K. (1979, June). A Markovian model for reliability an d other performance measures of software systems. In 1979 International Wo rkshop on Managing Requirements Knowledge (MARK) (pp. 769-774). IEEE.

[13] Gokhale, S. S., Philip, T., Marinos, P. N., & Trivedi, K. S. (1996, October). Unification of finite failure non-homogeneous Poisson process models throu

gh test coverage. In Proceedings of ISSRE'96: 7th International Symposium on Software Reliability Engineering (pp. 299-307). IEEE.

[14] Halstead, M. H. (1977). Elements of Software Science (Operating and progr amming systems series). Elsevier Science Inc..

[15] Hamada, M. S., Wilson, A. G., Reese, C. S., & Martz, H. F. (2008). Bayesia n Inference. Bayesian Reliability, 21-50.

[16] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[17] Hossain, S. A., & Dahiya, R. C. (1993). Estimating the parameters of a non -homogeneous Poisson-process model for software reliability. IEEE transact ions on Reliability, 42(4), 604-612.

[18] Inoue, S., & Yamada, S. (2017, December). On Bayesian inference of softw are reliability measurement. In 2017 International Conference on Infocom Te chnologies and Unmanned Systems (Trends and Future Directions)(ICTUS) (pp. 116-119). IEEE.

[19] Jang, W. G., & Lee, K. H. (2019). Big Data Governance Model for Effective Operation in Cyberspace. The Journal of Bigdata, 4(1), 39-51.

[20] Jelinski, Z., & Moranda, P. (1972). Software reliability research. In Statistical computer performance evaluation (pp. 465-484). Academic Press.

[21] Jeske, D. R., & Zhang, X. (2005). Some successful approaches to software rel iability modeling in industry. Journal of Systems and Software, 74(1), 85-99.

[22] Jung, H. J. (2006). On the Application Plan Study of International Standard Tendency in Software Quality Testing. Journal of Internet Computing and Se rvices, 7(4), 1-10.

[23] Kapur, P. K., Pham, H., Anand, S., & Yadav, K. (2011). A unified approach for developing software reliability growth models in the presence of imperfect deb ugging and error generation. IEEE Transactions on Reliability, 60(1), 331-340.

[24] Karunanithi, N., Whitley, D., & Malaiya, Y. K. (1992). Using neural networks in reliability prediction. IEEE Software, 9(4), 53-59.

[25] Kim, Y. S., Song, K. Y., Pham, H., & Chang, I. H. (2022). A software reliability model with dependent failure and optimal release time. Symmetry, 14(2), 343.

[26] Kim, Y. S., Song, K. Y., & Chang, I. H. (2023a). Prediction and Comparative Analysis of Software Reliability Model Based on NHPP and Deep Learning. A pplied Sciences, 13(11), 6730.

[27] Kim, Y. S., Pham, H., & Chang, I. H. (2023b). Deep-Learning Software Reliabili ty Model Using SRGM as Activation Function. Applied Sciences, 13(19), 10836.

[28] Kim, H. S., Park, D. H., & Yamada, S. (2009). Bayesian optimal release time based on inflection S-shaped software reliability growth model. IEICE transa ctions on fundamentals of electronics, communications and computer scien ces, 92(6), 1485-1493.

[29] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimizatio n. arXiv preprint arXiv:1412.6980.

[30] Kotz, S. (1970). Distributions in statistics: Continuous univariate distribution s. John Wiley & Sons.

[31] Lee, D. H., Chang, I. H., & Pham, H. (2020). Software reliability model with dependent failures and SPRT. Mathematics, 8(8), 1366.

[32] Lee, D. H., Chang, I. H., & Pham, H. (2022). Software reliability growth mod el with dependent failures and uncertain operating environments. Applied Sc iences, 12(23), 12383.

[33] Lee, S. S. (2004). Bayesian inference for software reliability models based o n NHPP.

[34] Li, Q., & Pham, H. (2017). NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. Applied Mathematical Modelling, 51, 68-85.

[35] Li, Q., & Pham, H. (2021). Modeling software fault-detection and fault-corr ection processes by considering the dependencies between fault amounts. Applied Sciences, 11(15), 6998.

[36] Li, X., Xie, M., & Ng, S. H. (2010). Sensitivity analysis of release time of sof tware reliability models incorporating testing effort with multiple change-poin ts. Applied Mathematical Modelling, 34(11), 3560-3570.

[37] Lieblein, J., & Zelen, M. (1956). Statistical investigation of the fatigue life of deep-groove ball bearings. Journal of research of the national bureau of sta ndards, 57(5), 273-316.

[38] Liu, Z. Y., & Rho, H. J. (2013). A Study on the Effects of the Competitivene ss of Intermediate Goods Competency on Customer Value and Enterprise Pe rformance in the China and Korea Electronic industry. Journal of The Korea Society of Computer and Information, 18(9), 201-208.

[39] Malaiya, Y. K., & Srimani, P. K. (1991). Software Reliability Models: Develop ments, Evaluation and Applications. IEEE Computer Society Press.

[40] Mann, N. R., Schafer, R. E., & Singpurwalla, N. D. (1974). Methods for stati stical analysis of reliability and life data(Book). Research supported by the U. S. Air Force and Rockwell International Corp. New York, John Wiley and

Sons, Inc., 1974. 573 p.

[41] McCabe, T. J. (1976). A complexity measure. IEEE Transactions on software Engineering, (4), 308-320.

[42] Mills, H. (1972). On the statistical validation of compute programs. 72-6015.

[43] Miyamoto, S., Tamura, Y., & Yamada, S. (2022). Reliability assessment tool based on deep learning and data preprocessing for OSS. American Journal of Operations Research, 12(3), 111-125.

[44] Moon, J. S. (2017). Exploratory study of enterprise software license complia nce disputes.

[45] Ogundoyin, S. O., & Kamil, I. A. (2020). A Fuzzy-AHP based prioritization of trust criteria in fog computing services. Applied Soft Computing, 97, 106789.

[46] Ohba, M. (1984, January). Inflection S-shaped software reliability growth m odel. In Stochastic Models in Reliability Theory: Proceedings of a Symposiu m Held in Nagoya, Japan, April 23-24, 1984 (pp. 144-162). Berlin, Heidelb erg: Springer Berlin Heidelberg.

[47] Oveisi, S., Moeini, A., Mirzaei, S., & Farsi, M. A. (2021). LSTM encoder-dec oder dropout model in software reliability prediction. Int. J. Reliab. Risk Saf. Theory Appl, 4, 1-12.

[48] Pham, H. (2006). System software reliability modeling. Springer.

[49] Pham, H. (2014). A new software reliability model with Vtub-shaped fault-d etection rate and the uncertainty of operating environments. Optimization, 6 3(10), 1481-1490.

[50] Pham, H. (2020). On estimating the number of deaths related to Covid-19. Mathematics, 8(5), 655.

[51] Pham, H., Nordmann, L., & Zhang, Z. (1999). A general imperfect-software -debugging model with S-shaped fault-detection rate. IEEE Transactions o n reliability, 48(2), 169-175.

[52] Pham, H., & Zhang, X. (1997). An NHPP software reliability model and its c omparison. International Journal of Reliability, Quality and Safety Engineerin g, 4(03), 269-282.

[53] Pham, H., & Zhang, X. (2003). NHPP software reliability and cost models with testing coverage. European Journal of Operational Research, 145(2), 443-454.

[54] Pham, L., & Pham, H. (2000). Software reliability models with time-depende nt hazard function based on Bayesian approach. IEEE Transactions on Syst ems, Man, and Cybernetics-Part A: Systems and Humans, 30(1), 25-35.

[55] Qian, N. (1999). On the momentum term in gradient descent learning algorit

hms. Neural networks, 12(1), 145-151.

[56] Raamesh, L., Jothi, S., & Radhika, S. (2022). Enhancing software reliability and fault detection using hybrid brainstorm optimization-based LSTM model. IETE Journal of Research, 1-15.

[57] Rafi, S., Akbar, M. A., Yu, W., Alsanad, A., Gumaei, A., & Sarwar, M. U. (2022). Exploration of DevOps testing process capabilities: An ISM and fuzzy TOPSIS analysis. Applied Soft Computing, 116, 108377.

[58] Raghuvanshi, K. K., Agarwal, A., Jain, K., & Singh, V. B. (2021). A time-variant fault detection software reliability model. SN Applied Sciences, 3, 1-10.

[59] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

[60] Roy, P., Mahapatra, G. S., & Dey, K. N. (2014). An NHPP software reliability growth model with imperfect debugging and error generation. International Journal of Reliability, Quality and Safety Engineering, 21(02), 1450008.

[61] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[62] Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986). A general framework for parallel distributed processing. Parallel distributed processing: Explorations in the microstructure of cognition, 1(45-76), 26.

[63] Sahu, K., Alzahrani, F. A., Srivastava, R. K., & Kumar, R. (2021). Evaluating the Impact of Prediction Techniques: Software Reliability Perspective. Computers, Materials & Continua, 67(2).

[64] Seo, J. H., & Kim, S. H. (2008). Development of the Reliability Evaluation Model and the Analysis Tool for Embedded Softwares. IE interfaces, 21(1), 109-119.

[65] Sideratos, I. G., Platis, A. N., Koutras, V. P., & Ampazis, N. (2014). Reliability analysis of a two-stage Goel-Okumoto and Yamada S-Shaped model. In Proceedings of the Ninth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX. June 30-July 4, 2014, Brunów, Poland (pp. 393-402). Springer International Publishing.

[66] Singh, V. B., & Sharma, M. (2014, November). Prediction of the complexity of code changes based on number of open bugs, new feature and feature improvement. In 2014 IEEE International Symposium on Software Reliability Engineering Workshops (pp. 478-483). IEEE.

[67] Singpurwalla, N. D. (1991). Determining an optimal time interval for testing and debugging software. IEEE Transactions on Software Engineering, 17(4), 313.

[68] Singpurwalla, N. D., & Wilson, S. P. (1994). Software reliability modeling. Int

ernational Statistical Review/Revue Internationale de Statistique, 289-317.
[69] Singpurwalla, N. D., & Wilson, S. P. (2012). Statistical methods in software engineering: reliability and risk. Springer Science & Business Media.
[70] Sommerville, I. (2001). Software Engineering 8th edition. sl.
[71] Song, K. Y., Chang, I. H., & Pham, H. (2017a). A three-parameter fault-detection software reliability model with the uncertainty of operating environments. Journal of Systems Science and Systems Engineering, 26, 121-132.
[72] Song, K. Y., Chang, I. H., & Pham, H. (2017b). An NHPP software reliability model with S-shaped growth curve subject to random operating environments and optimal release time. Applied Sciences, 7(12), 1304.
[73] Song, K. Y., Chang, I. H., & Pham, H. (2018). Optimal release time and sensitivity analysis using a new NHPP software reliability model with probability of fault removal subject to operating environments. Applied Sciences, 8(5), 714.
[74] Song, K. Y., Chang, I. H., & Pham, H. (2019a). A testing coverage model based on NHPP software reliability considering the software operating environment and the sensitivity analysis. Mathematics, 7(5), 450.
[75] Song, K. Y., Chang, I. H., & Pham, H. (2019b). NHPP software reliability model with inflection factor of the fault detection rate considering the uncertainty of software operating environments and predictive analysis. Symmetry, 11(4), 521.
[76] Song, K. Y., Kim, Y. S., & Chang, I. H. (2023). Software reliability model for open-source software that considers the number of finite faults and dependent faults. Mathematical Biosciences and Engineering, 20(7), 11785-11804.
[77] Song, S. (2017). Historical Development of Industrial Revolutions and the Place of So called'the Fourth Industrial Revolution'. Journal of Science and Technology Studies, 17(2), 5-40.
[78] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in neural information processing systems, 27.
[79] Tamura, Y., & Yamada, S. (2016). Software reliability model selection based on deep learning with application to the optimal release problem. Journal of Industrial Engineering and Management Science, 1, 43-58.
[80] Teng, X., & Pham, H. (2006). A new methodology for predicting software reliability in the random field environments. IEEE Transactions on Reliability, 55 (3), 458-468.
[81] Tohma, Y., Yamano, H., Ohba, M., & Jacoby, R. (1991). The estimation of parameters of the hypergeometric distribution and its application to the software reliability growth model. IEEE Transactions on Software Engineering, 17(5), 483.

[82] Weibull, W. (1951). A statistical distribution function of wide applicability. Journal of applied mechanics.

[83] Williams, D. P. (2007). Study of the warranty cost model for software reliability with an imperfect debugging phenomenon. Turkish Journal of Electrical Engineering and Computer Sciences, 15(3), 369-381.

[84] Wong, K. L. (1989). The roller-coaster curve is in. Quality and reliability engineering international, 5(1), 29-36.

[85] Wu, C. Y., & Huang, C. Y. (2021). A study of incorporation of deep learning into software reliability modeling and assessment. IEEE Transactions on Reliability, 70(4), 1621-1640.

[86] Yamada, S., & Osaki, S. (1985). Cost-reliability optimal release policies for software systems. IEEE Transactions on Reliability, 34(5), 422-424.

[87] Yamada, S., Ohba, M., & Osaki, S. (1983). S-shaped reliability growth modeling for software error detection. IEEE Transactions on reliability, 32(5), 475-484.

[88] Yamada, S., Ohtera, H., & Narihisa, H. (1986). Software reliability growth models with testing-effort. IEEE Transactions on Reliability, 35(1), 19-23.

[89] Yamada, S., Tokuno, K., & Osaki, S. (1992). Imperfect debugging models with fault introduction rate for software reliability assessment. International Journal of Systems Science, 23(12), 2241-2252.

[90] Yi, M. J., Kim, G. B., Sohn, Y. C., Lee, J. Y., & Lee, K. K. (2004). Time series analysis of groundwater level data obtained from national groundwater monitoring stations. Journal of the Geological Society of Korea, 40(3), 305-329.

[91] Zhang, X., & Pham, H. (1998). A software cost model with error removal times and risk costs. International Journal of Systems Science, 29(4), 435-442.

[92] Zhang, X., Teng, X., & Pham, H. (2003). Considering fault removal efficiency in software reliability assessment. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 33(1), 114-120.

[93] 박동호, 임재학, 남경현, & 정기문. (2015). 신뢰성 이론과 수명분포 응용, 자유아카데미.

[94] 이성경 (1986). 신뢰성 공학의 발전사적 고찰. 한국전자통신연구원, 1(2), 83-90.

[95] 정해성, 박동호, & 김재주. (2003). 신뢰성 분석과 응용, 영지문화사.