석사학위논문

스파이킹 심층 신경망을 위한 오픈신경망교환포맷

2020년

한 성 대 학 교 대 학 원

컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

박 상 민

석 사 학 위 논 문
지도교수 허준영

스파이킹 심층 신경망을 위한 오픈신경망교환포맷

ONNX for the spiking deep neural network

2019년 12월 일

한 성 대 학 교 대 학 원

컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

박 상 민

석 사 학 위 논 문
지도교수 허준영

스파이킹 심층 신경망을 위한 오픈신경망교환포맷

ONNX for the spiking deep neural network

위 논문을 공학 석사학위 논문으로 제출함

2019년 12월 일

한 성 대 학 교 대 학 원

컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

박 상 민

박상민의 공학 석사학위 논문을 인준함

2019년 12월 일

심사위원장 _____(인)

심 사 위 원 _____(인)

심 사 위 원 _____(인)

국 문 초 록

스파이킹 심층 신경망을 위한 오픈신경망교환포맷

한성대학교 일반대학원
컴퓨터공학과
컴퓨터공학전공
박 상 민

스파이킹 신경망(Spiking Neural Network, SNN)은 3세대 신경망으로 1, 2세대 신경망과 다른 메커니즘(Mechanism)으로 동작한다. 비 스파이킹 신경망인 1세대 신경망 단일 계층 퍼셉트론(Single Layer Perceptron, SLP)과 2세대 신경망인 다중 계층 퍼셉트론(Multi Layer Perceptron, MLP)에서는 입력 값에 대해서 생물학적 메커니즘을 고려하지 않은 활성함수를 거쳐 출력 값을 내보낸다. 반면 3세대 신경망인 스파이킹 신경망에서는 기존 활성화함수보다 실제 뉴런의 생물학적 메커니즘과 가장 유사한 뉴런모델을 기반으로 동작한다. 하지만 스파이킹 신경망을 사용하지 않더라도 기존 비 스파이킹 신경망에서도 좋은 성과들이 있었다. 심층 신경망(Deep Neural Network, DNN)의 구조에 따라 성능이 개선되는 것인데 이러한 점에 영감을 받아 스파이킹 신경망에 비 스파이킹 심층 신경망 모델 구조를 가져와 뉴런모델만을 스파이킹 뉴런으로 대체하는 시도들이 있었다. 하지만 인공지능 프레임워크마다 다른 데이터 포맷 때문에 같은 신경망 구조를 각각의 프레임 워크에서 다른 데이터 포맷으로 표현한다. 이러한 문제 때문에 다른 프레임워크로 변환하는 과

정이 필요하게 된다. 이러한 문제를 기존 인공지능 프레임워크에서는 오픈신경망교환포맷(Open Neural Network, ONNX)을 사용하여 해결했다. 다른 인공지능 프레임워크에서 만들어진 모델을 오픈신경망교환포맷으로 변환하고 변환 툴을 사용해서 변환하고자 하는 인공지능 프레임워크에 맞는 데이터 포맷으로 변환하는 방식이다. 즉 오픈신경망교환을 지원하는 프레임워크는 인공지능모델을 간단히 변환하여 사용할 수 있다. 하지만 스파이킹 신경망을 지원하는 프레임워크에서는 현재 오픈신경망교환포맷을 지원하고 있지 않고 있으며 오픈신경망교환포맷에 스파이킹 뉴런에 대한 정의도 되어 있지 않다.

본 논문에서 제안하는 내용은 다음과 같다. 첫 번째 오픈신경망교환포맷을 기반으로 스파이킹 신경망을 표현할 수 있도록 하기 위해 ONNX-SNN을 제안하고 오픈신경망교환포맷에서 ONNX-SNN로 변환방법에 대해 소개한다. 두 번째 ONNX-SNN을 Nengo 프레임워크상에서 사용할 수 있는 모델 생성 및 코드 생성 방법을 제안한다.

【주요어】 퍼셉트론, 스파이킹 신경망, 오픈신경망교환포맷

목 차

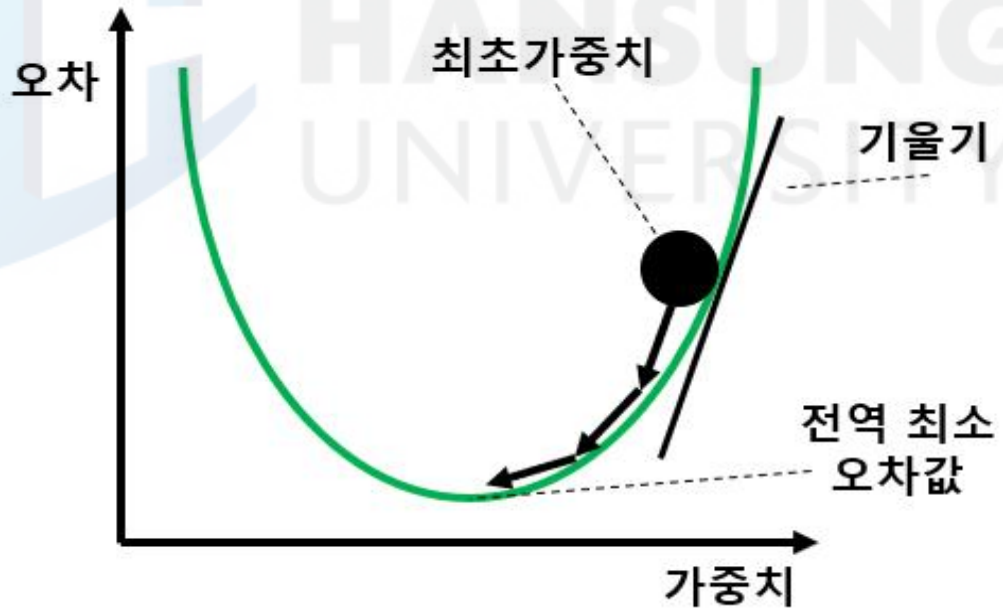
표 목 차

그 림 목 차

제 1 장 서론

제 1 절 연구 내용

최근 전 세계적으로 인공지능에 대한 관심과 투자가 많아지고 있다. 이러한 배경에는 그래픽 처리 유닛(Graphics Processing Unit, GPU)과 컴퓨팅 기술 발전, 다양한 디지털 기기를 통한 많은 량의 데이터가 생성되는 환경이 조성되면서 부터이다(Goodfellow, I., 2016). 복잡한 연산이 가능한 하드웨어가 뒷받침 되면서 인공 신경망(Artificial Neural Network, ANN)중 특히 다중 레이어로 구성되어 있는 심층 신경망(Deep Neural Network)과 학습 알고리즘 대한 좋은 성과가 있었다(LeCun, Y., 1998; Hinton, G. E., 2006).



[그림 1-1] 경사 하강법 예시

심층 신경망은 실제 뇌 구조와 유사한 구조를 가지고 있다. 하지만 실제

뇌에서 동작하는 뉴런(Neuron)과 뉴런사이에 신호 전달 방식과 차이가 있다. 심층 신경망의 경우 뉴런에서 생물학적인 메커니즘을 고려하지 않고 인위적으로 설정한 활성화 함수(Activation function)를 거쳐 다음 뉴런에 값을 전달한다. 그리고 경사 하강법(Gradient descent)과 오차 역전파(Error back-propagation) 통해 오차가 최소가 되는 값을 신경망에 학습시킨다.

반면에 스파이킹 신경망은 생물학적인 메커니즘과 가장 유사하게 동작하는 인공 신경망이다. 최근에는 기존 심층 신경망에 좀더 생물학적인 스파이킹 뉴런을 도입하여 심층 신경망을 만들기 위한 다양한 시도들이 많아지고 있다(Brader, J. M., 2009; Eliasmith, C., 2012; O'Connor, P., 2013; Neftci, E., 2014; Cao, Y., 2015; Diehl, P. U., 2015; Lee, J. H., 2016).

기존 심층 신경망(비 스파이킹 심층 신경망)은 텐서플로(Tensorflow)(Abadi, M., 2016), 케라스(Keras), 파이토치(Pytouch)등 다양한 인공지능 개발 프레임워크가 있다. 하지만 인공지능 개발자가 신경망을 구성함에 있어 같은 신경망을 특정 하나의 프레임워크에서 개발을 하면 다른 인공지능 프레임워크에 바로 가져다 쓸 수 없다. 각각의 프레임워크마다 신경망을 표현하는 포맷이 다르기 때문인데 이 문제를 오픈신경망교환포맷을 통해 다른 프레임워크에 쉽게 변환할 수 있다. 하지만 현재 오픈신경망교환포맷은 뉴런 동작방식(오퍼레이터) 중에서 스파이킹 뉴런의 동작방식을 정의하고 있지 않아 오픈신경망교환포맷으로 스파이킹 신경망을 표현할 방법이 없다.

본 연구에서는 오픈신경망교환포맷에서 스파이킹 신경망에 특화된 ONNX-SNN을 제안하고 기존 오픈신경망교환포맷에서 ONNX-SNN으로 변환하는 방법과 ONNX-SNN로드하여 스파이크 심층 신경망 프레임워크에 사용할 수 있는 모델 생성 및 코드 생성방법을 제안한다.

제 2 절 논문 구성

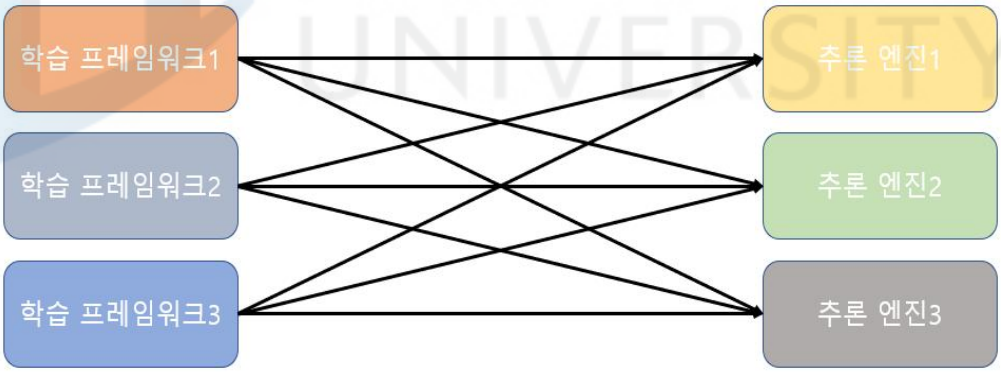
본 논문의 구성은 2장에서 NNEF, ONNX, Protobuf, Nengo(Bekolay, T., 2014), NengoDL(Rasmussen, D., 2019)에 대해 소개하고 본 연구에서 사용한 스파이킹 심층 신경망의 학습과 추론 방법에 대해 설명한다. 3장에서는 ONNX-SNN를 제안하고 ONNX에서 ONNX-SNN으로 변환하는 방법에 대해 소개한다. 4장에서는 ONNX-SNN을 Nengo 프레임워크에서 사용할 수 있는 모델과 코드 변환방법을 제안한다. 5장에서는 3장 4장에서 제안한 방법을 통해 변환한 결과물들을 보여주고 각 활성화함수별 추론결과를 보여준다. 마지막으로 6장에서는 결론 및 향후 개선 방안을 제시한다.

제 2 장 관련연구

제 1 절 신경망교환포맷

1) NNEF

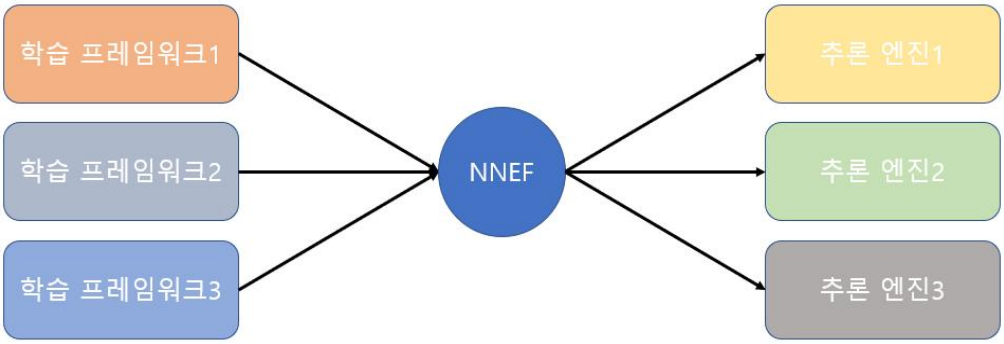
NNEF(Neural Network Exchange Format, NNEF)는 신경망교환포맷으로 Khronos 그룹에서 진행하고 있는 프로젝트다. NNEF는 다양한 범위의 기기와 플랫폼에 걸쳐 응용프로그램에 다양한 신경망 훈련 프레임워크와 추론 엔진을 사용할 수 있도록 하여 인공지능 시스템 배치(deployment)할 발생하는 파편화(fragmentation)를 줄이기 위해 제시되었다. NNEF의 목표는 학습된 인공지능 모델을 다양한 추론엔진으로 쉽게 이식할 수 있도록 하는 것에 초점을 맞췄다.



[그림 2-1] 일반적인 학습 프레임워크와 추론 엔진 간에 변환

대부분의 인공지능 개발 프레임워크(Torch(Collobert, R., 2011), Caffe(Jia, Y., 2014), Tensorflow, Theano(Team, T. T. D., 2016), Chainer, Caffe2, Pytorch, MXNet)에는 자체적인 포맷을 사용하기 때문에 [그림 2-1]과 같이 프레임워크에서 훈련시킨 신경망을 다른 추론엔진에서 사용하기 위

해서 실행할 수 있도록 추론엔진 포맷으로 변환하는 과정이 동반되며 추론엔진이 다수이면 각각의 추론엔진마다 변환과정을 반복해야하는 번거로움이 있다.



[그림 2-2] NNEF를 통해 학습 프레임워크와 추론 엔진 간에 효율적 변환

반면 NNEF를 사용하면 [그림 2-2]와 같이 신경망을 훈련시킨 프레임워크에서 추론엔진마다 변환하는 과정이 필요하지 않다. 즉 많은 자원소비 없이 경제적으로 다양한 추론엔진에 맞는 포맷으로 변환 시킬 수 있다. NNEF는 네트워크와 데이터 파일이 분리되어 있다. 이러한 구조는 신경망 구조와 각각의 파라미터 데이터에 독립적인 접근을 쉽게 만들어 준다. 그리고 여러 개의 파일 세트들을 tar, zip과 같은 컨테이너를 사용하여 선택적 압축 및 암호화 할 수 있다.

2) ONNX

오픈신경망교환포맷(Open Neural Network Exchange, ONNX)은 페이스북(Facebook)과 마이크로소프트(Microsoft)의 합작으로 시작한 오픈소스 프로젝트이다. 오픈신경망교환포맷은 인공지능 개발자들이 인공지능모델 개발에 있어서 다양한 인공지능 개발 프레임워크를 효과적으로 사용할 수 있는 환경을 만들기 위한 시도에서 시작되었다. 오픈신경망교환포맷의 파일 형식은 데이터구조 지향적인 Protobuf를 사용하며 오픈신경망교환포맷은 변환 중간단

계에서 IR(Intermediate Representation)생성과정을 거친다. 프레임워크마다 다른 그래프 표현방식을 사용하는데 오픈신경망교환포맷은 IR을 통해 공통의 그래프형식으로 표현한다. 특정 프레임워크에서 개발된 신경망이 오픈신경망교환포맷으로 변환하면 IR의 공통의 그래프 표현방식 덕분에 다양한 프레임워크에 쉽게 변환할 수 있다.

오픈신경망교환포맷은 현재 두 가지 형태로 나뉜다. 하나는 ONNX(오픈신경망교환포맷)이고 또 다른 하나는 ONNX-ML이다. ONNX-ML은 전통적인 머신러닝(Machine learning, ML)에 필요한 타입정보와 연산자를 가지고 있으며 이러한 두 가지 형태는 심층 신경망 형태뿐만 아니라 전통적 머신러닝에 대한 상호호환성을 높여준다.

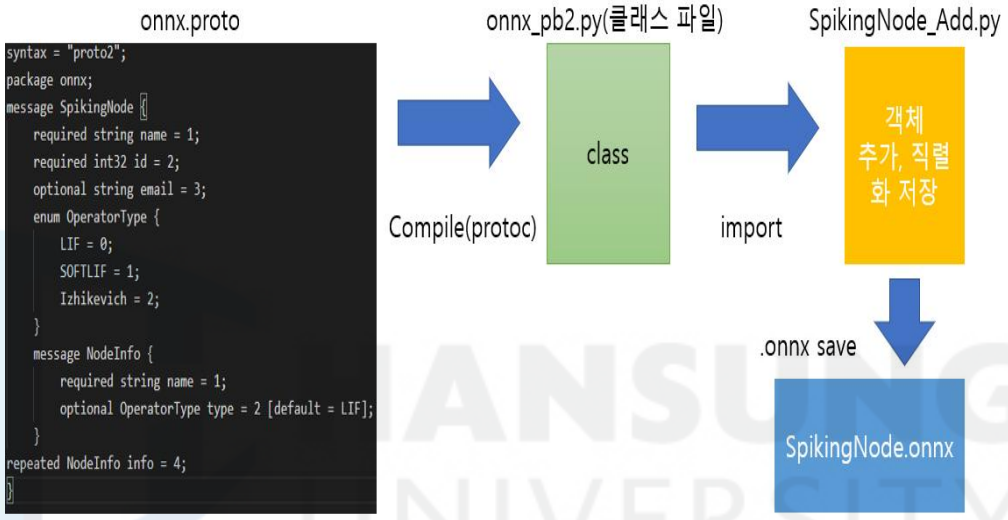
[표 2-1] NNEF와 ONNX비교

	ONNX	NNEF
개발 방식	오픈소스 프로젝트	Khronos
포맷 형식	데이터 구조 지향 Protobuf형식	텍스트 기반 절차형식
데이터	Tensor의 데이터형식 사용	추론 가속화를 위한 유연한 최적화 지원
개발목적	다양한 인공지능 프레임워크에서 빠른 개발 환경 제공	다양한 임베디드 추론엔진에 효과적으로 변환

신경망교환포맷과 오픈신경망교환포맷 모두 탄생 배경은 비슷하지만 프로젝트를 참여하고 있는 단체에 차이가 있다. 오픈신경망교환포맷은 소프트웨어 회사, 하드웨어 회사들뿐만 아니라 모두에게 오픈되어 있다. 많은 참여자들 덕분에 오픈신경망교환포맷은 신경망교환포맷과 비교해 다양한 인공지능개발 프레임워크에서 지원하고 있으며 오픈소스로 프로젝트로 진행되는 만큼 최신 기술 적용이 빠르고 문제점에 대한 해결속도도 빠른 편이다. 뿐만 아니라 오픈신경망교환포맷은 Protobuf포맷을 사용하기 때문에 또 다른 변형 포맷을 만들기에 용의하다. 이러한 오픈신경망교환포맷의 장점이 본 연구에서 진행하는 ONNX-SNN 포맷의 기반으로 사용된다.

제 2 절 Protobuf

Protobuf는 구조화된 데이터를 직렬화(Serialization)를 통해 데이터를 쉽게 주고받을 수 있도록 지원하는 파일저장포맷이다. XML과 유사하지만 더 작은 크기를 갖으며 빠르고 단순하다.



[그림 2-3] Protobuf 포맷기반 ONNX파일 생성과정

[그림 2-3]에서 오픈신경망교환포맷(ONNX) 생성과정을 보여주는 예제이다. Protobuf를 사용하기 위해서는 첫 번째로 proto파일을 생성해야한다. proto파일은 사용하고자하는 데이터 구조(클래스)를 proto파일 형태로 정의한다. [그림 2-3]에서 onnx.proto는 오픈신경망교환포맷을 신경망을 구성하는 컴포넌트들과 속성들을 정의한 모습이다. 그리고 Protobuf의 컴파일러인 protoc를 사용하여 컴파일하면 클래스 파일(onnx_pb2.py)이 생성된다. 클래스 파일은 C++, Python, Java, Go 등 다양한 형태로 생성할 수 있으며 컴파일러를 통해 생성하고자하는 프로그래밍언어를 지정하여 생성할 수 있다. [그림 2-3]에서 SpikingNode_Add.py파일은 onnx_pb2.py파일을 어떠한 방식으

로 사용되는지 보여주기 위해 임의로 만들어낸 파일이다. SpikingNode_Add.py파일에서는 onnx_pb2.py를 import하여 proto파일에서 정의했던 클래스를 이용하여 객체를 만들고 SerializeToString()메소드를 사용하여 문자열을 직렬화 시킨 후 .onnx파일 형식으로 저장하는 모습을 보여준다. 이러한 방식으로 신경망 모델을 .onnx형식으로 저장할 수 있다. 저장한 파일을 다시 읽기위해서는 ParseFromString()메소드를 호출하여 객체에 파싱을 통해 읽어야 한다. 파싱을 사용하지 않고 .onnx파일을 직접 열어 본다면 .proto파일을 참고하지 않고서는 의미를 이해하기 어렵다.

[표 2-2] Protobuf 및 다른 프로그래밍 언어의 데이터 타입 리스트

.proto	C++	Java	Python	Go
double	double	double	float	*float64
float	float	float	float	*float32
int32	int32	int	int	*int32
int64	int64	long	int/long	*int64
uint32	uint32	int	int/long	*uint32
uint64	uint64	long	int/long	*uint64
sint32	int32	int	int	*int32
sint64	int64	long	int/long	*int64
fixed32	uint32	int	int/long	*uint32
fixed64	uint64	long	int/long	*uint64
sfixed32	int32	int	int	*int32
sfixed64	int64	long	int/long	*int64
bool	bool	boolean	bool	*bool
string	string	String	str(Python3)	*string
bytes	string	ByteString	bytes	[]byte

Protobuf(proto2버전)에 사용 할 수 있는 데이터 타입과 Protobuf의 컴파일러로 생성되는 프로그래밍 언어별 데이터 타입은 [표 2-2]과 같고 .proto의 몇 가지 생소한 데이터형에 대해 설명하면 다음과 같다.

uint32와 uint64는 가변 길이 인코딩에 사용한다. sint32와 sint64는 int32와 int64보다 음수 인코딩에 더 효율적이며 가변 길이 인코딩에 사용한다.

fixed32는 2^{28}보다 큰 값을 자주 사용할 때 uint32보다 더 효율적며 4바이트 고정 데이터 타입이다. fixed64는 2^{56}보다 큰 값을 자주 사용할 때 uint64보다 더 효율적며 8바이트 고정 데이터 타입이다. sfixed32는 4바이트 고정 데이터 타입이며 sfixed64는 8바이트 고정 데이터 타입이다.

[표 2-3] Proto파일 예시(*.proto)

ONNX-SNN Proto

```
1:  syntax = "proto2";
2:  package onnx;
3:  message SpikingNode {
4:      required string name = 1;
5:      required int32 id = 2;
6:      optional string email = 3;
7:      enum OperatorType {
8:          LIF = 0;
9:          SoftLIFRate = 1;
10:         Izhikevich = 2;
11:     }
12:     message NodeInfo {
13:         required string name = 1;
14:         optional OperatorType type = 2 [default = LIF];
15:     }
16:     repeated NodeInfo info = 4;
17: }
```

Protobuf는 한 가지 프로그래밍 언어에 종속되지 않고 여러 프로그래밍 언어를 지원하기 위해 종속성이 없는 형태의 데이터 타입([표 2-2]), 클래스를 [표 2-3]와 같이 작성하게 된다. [표 2-3]은 proto2버전을 기준으로 Protobuf문법에 대해 간단히 설명하기 위해 작성한 예시이다. package는 C++의 네임스페이스와 유사한 기능으로 데이터 구조의 충돌방지를 위해 어떤 패키지를 명시하기 위해 사용된다(line 2). message는 데이터 구조의 이름을 나타낸다. [표 2-3]의 line 3, line 12번째에서 데이터 SpikngNode와

NodeInfo라는 데이터 구조가 선언되어 있다. enum은 타입을 열거할 때 사용된다. [표 2-2]의 line 7 ~ line 10에서 오퍼레이터(Operator)타입으로 LIF, SoftLIFRate, Izhikevich가 열거되어 있는 것을 볼 수 있다. 데이터 타입 앞에 명시하는 required, optional, repeated는 수식어(modifier)로서 해당 값이 반드시 필요한지 선택적으로 필요한지 반복되는지를 명시해주는 역할을 한다. default는 값이 지정되지 않은 경우에 어떠한 값을 기본 값으로 할지 설정할 때 사용한다. [표 2-3]의 line 14에서 오퍼레이터 타입이 기본 값으로 LIF로 지정되어 있음을 볼 수 있다.

제 3 절 Nengo와 NengoDL

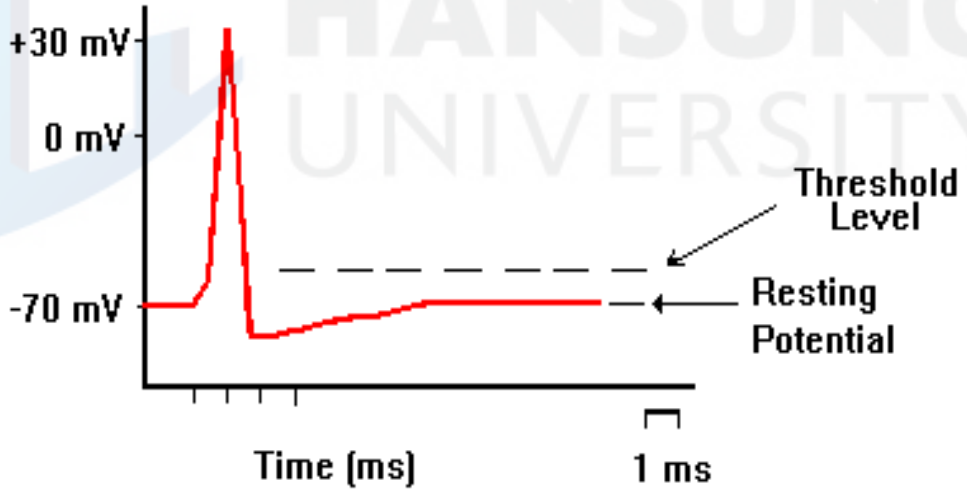
Nengo는 파이썬(Python)기반 뇌를 시뮬레이션하기 위해 만들어진 프레임워크이다. Nengo 프레임워크는 뉴로모픽 하드웨어 환경(Benjamin, B. V., 2014)에서 뿐만 아니라 뉴로모픽 하드웨어가 없는 환경에서도 스파이킹 뉴런을 사용 할 수 있도록 시뮬레이션 환경을 제공해준다.

최근 비 스파이킹 비 스파이킹 심층 신경망분야에서 다양한 심층신경망 구조에 대해 연구되면서 많은 성과가 있었다(Brader, J. M., 2009; Eliasmith, C., 2012; O'Connor, P., 2013; Neftci, E., 2014; Cao, Y., 2015; Diehl, P. U., 2015; Lee, J. H., 2016). Nengo 프레임워크에서는 비 스파이킹 심층 신경망 모델들을 그대로 사용할 수 있도록 NengoDL라이브러리를 제공한다. NengoDL은 Tensorflow기반으로 구현되어 있어 비 스파이킹 신경망에서 사용하던 기본적인 레이어 구조를 Nengo 프레임워크에서 구축 할 수 있도록 한다. Nengo 프레임워크에서 스파이킹 심층 신경망을 구현하기 위해서는 비 스파이킹 심층 신경망에서 주로 쓰이던 합성곱 레이어(Convolution layer), 폴링 레이어(Pooling layer)의 구조는 그대로 사용하고 레이어 안에 일반적인 비 스파이킹 뉴런을 스파이킹 뉴런으로 바꾸어 모델을 구성해야 한다.

하지만 일반적인 스파이킹 뉴런은 미분할 수 없는 특징을 가지고 있다. 이러한 점이 스파이킹 뉴런으로 심층 신경망 학습에 어려움을 준다. 비 스파이

킹 심층 신경망의 경우 경사 하강법(Gradient descent)을 사용하여 비용함수가 최소화되는 최적의 파라미터 값을 찾는다. NengoDL은 이러한 문제를 SoftLIF 뉴런을 이용한 학습 방법을 제시한 논문들(Hunsberger, E., 2015; Hunsberger, E., 2016)에 영감을 받아 스파이킹 심층 신경망 학습방법을 지원한다. (본 논문에서는 STDP(Spike timing dependent plasticity)을 사용한 학습방법은 다루지 않는다.) Nengo와 NengoDL에서 사용 할 수 있는 스파이킹 뉴런모델은 LIF(Leaky Integrate-and-Fire), SoftLIF, Izhikevich 등 다양한 스파이킹 모델이 있다. 보편적으로 사용하는 스파이킹 뉴런모델은 LIF이며 스파이킹 뉴런모델에 대한 자세한 내용은 2장 4절 스파이킹 심층 신경망 학습 및 추론에서 자세히 설명한다.

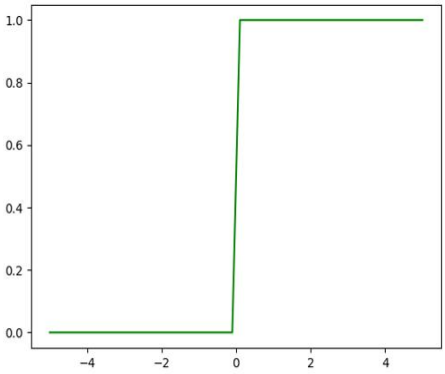
제 4 절 스파이킹 심층 신경망 학습 및 추론



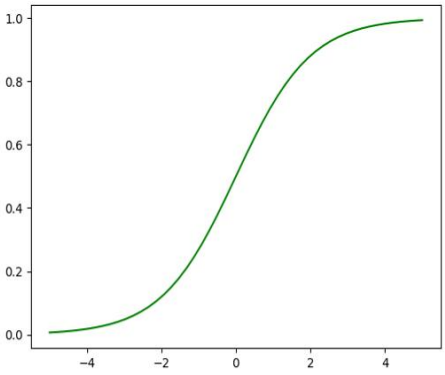
[그림 2-4] Leaky Integrate-and-Fire 예시

스파이킹 심층 신경망이란 기존 비 스파이킹 심층 신경망에서 뉴런의 작동방식을 실제 생물학적인 메커니즘에 더 가깝게 만들어진 알고리즘이다. 대표적으로 [그림 2-4]과 같은 LIF모델이 있다. 온전한 스파이킹 신경망을 사

용하기 위해서는 스파이킹 뉴런 모델을 하드웨어설계로 구현해야 한다.

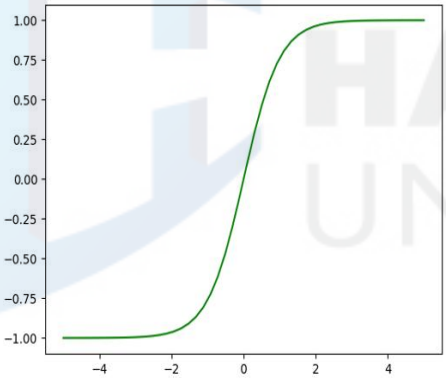


단계 함수 $f(x) = \begin{cases} 0 & (x \le 0) \\ 1 & (x > 0) \end{cases}$

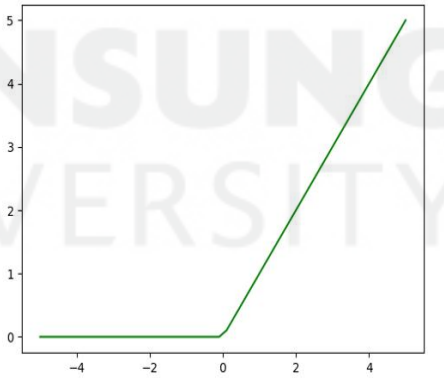


시그모이드 $f(x) = \frac{1}{1+e^{-x}}$

[그림 2-5] 활성화 함수 예시1



하이퍼볼릭 탄젠트 $f(x) = \tanh(x)$

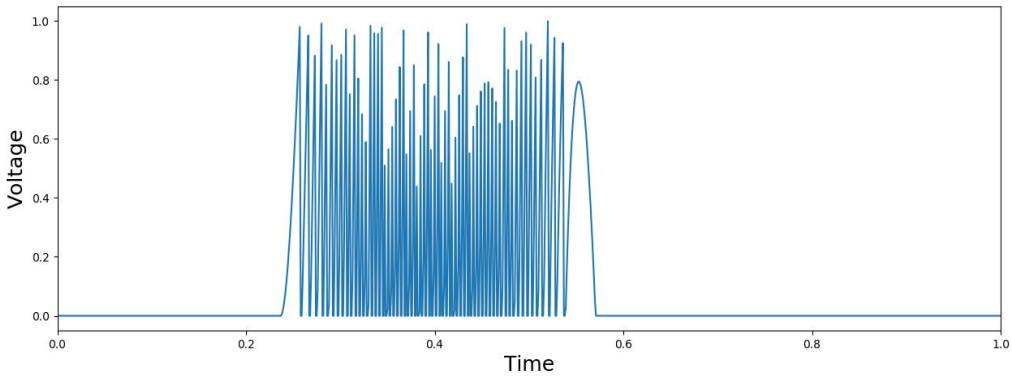


정류된 선형 유닛 $f(x) = \begin{cases} x < 0 & f(x) = 0 \\ x \ge 0 & f(x) = x \end{cases}$

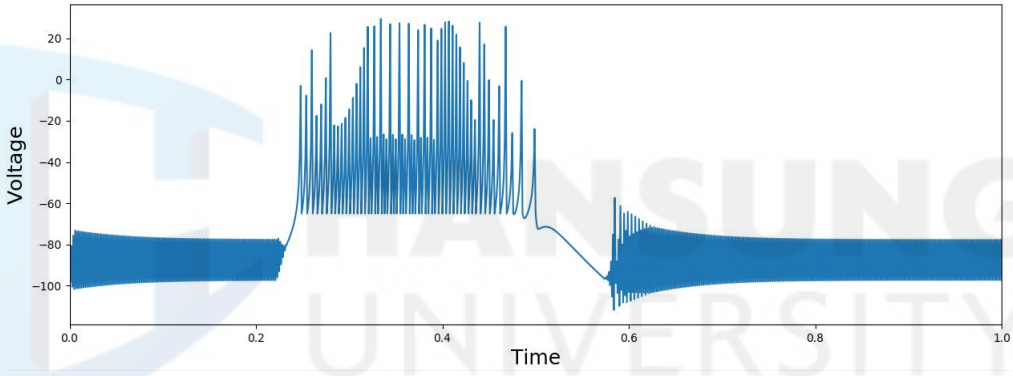
[그림 2-6] 활성화 함수 예시2

이러한 문제 때문에 비 스파이킹 신경망을 학습시키는 과정에서는 [그림 2-5]과 [그림 2-6]와 같은 계단함수(Step function), 시그모이드(Sigmoid), 하이퍼볼릭 탄젠트(Tanh), 정류된 선형 유닛(Rectified Linear Unit, ReLU)와 같은 활성화 함수를 통해 비 스파이킹 신경망을 학습하는데 사용하고 있다.

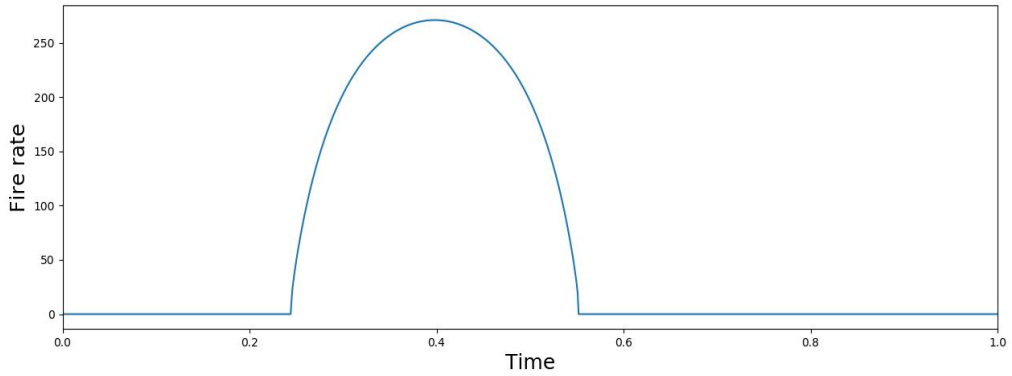
NengoDL은 발화율(Fire rate)기반의 스파이킹 심층망 학습방법을 지원한다. 발화율 기반으로 학습시키는 이유는 다음과 같다.



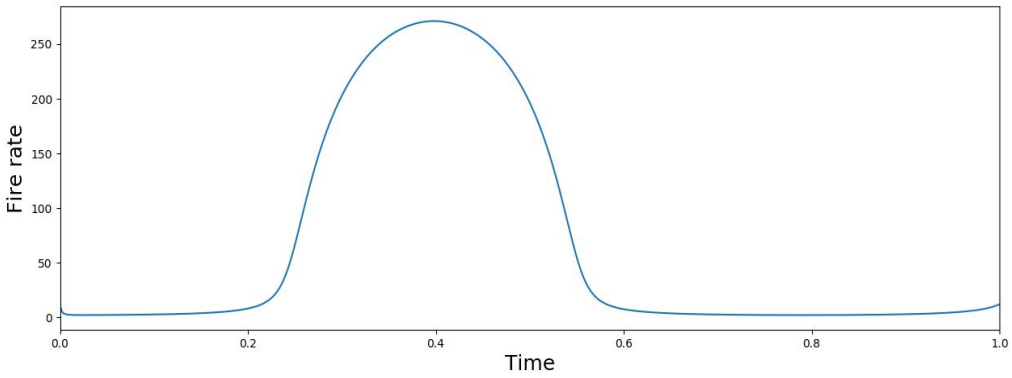
[그림 2-7] LIF뉴런 막전위 그래프



[그림 2-8] Izhikevich뉴런 막전위 그래프



[그림 2-9] LIFRate 발화율 그래프



[그림 2-10] SoftLIFRate 발화율 그래프

[그림 2-7], [그림 2-8], [그림 2-9], [그림 2-10]은 시간 t에 대해서 스파이킹 뉴런에 입력 값을 cos(8t)로 하여 시뮬레이션 한 결과이다. [그림 2-7] LIF뉴런모델이고 [그림 2-8]는 Izhikevich뉴런모델이다. 시간 t에 대해 변하는 입력 값 cos(8t)값에 따라 막전위가 변화한 그래프를 볼 수 있다. 여기서 입력 값은 전류와 같다. 입력 값에 대한 자극으로 입력 전류(값)의 강도에 따라 막전위가 빠르게 오른다. 특정 값 이상이 되면 발화되며 재분극 현상이 온다. 재분극 현상을 통해 뉴런의 막전위 값이 이전 상태로 되돌아온다. [그림 2-7]과 [그림 2-8] 모델을 심층 신경망에서 뉴런 모델로 사용한다면 비용함수 미분화여 최소화시키는 과정에서 학습에 어려움을 준다.

이러한 문제를 단위 시간당 발화한 개수 즉 발화율로 수치화 하여 [그림 2-9]과 같이 부드러운 곡선형태로 만드는 과정이 필요하다. [그림 2-9]과 같이 발화율 기반으로 학습을 진행하면 비선형성을 제공하고 충분히 부드럽게 곡선 형태를 갖기 때문에 스파이킹 뉴런을 충분히 근사(Approximation)했다고 할 수 있다. 하지만 [그림 2-9]을 미분하여 시간에 따른 입력 전압을 0+로 무한대로 접근하는 과정에서 특정 시점에 기울기가 0으로 고정되기 때문에 역전파(Backpropagation)과정에서 문제를 야기할 수 있다.

이러한 문제를 해결하기 위해 NengoDL에서는 SoftLIFRate를 기반으로

스파이킹 심층 신경망을 학습하는데 이용한다. SoftLIFRate는 LIF를 발화율로 수치화 했을때의 문제점을 해결하기 위해 개선시킨 스파이킹 뉴런이다. LIF뉴런을 발화율로 수치화했을때 미분이 불가능한 구간을 부드럽게 곡선 형태로 만들어 미분이 가능하도록 근사시켜 해결하고자 했다. [그림 2-10]은 SoftLIFRate뉴런 모델을 발화율로 수치화하여 사용한 모습이다. [그림 2-9]에서 미분할 수 없었던 시점에서 곡선형태로 변화되면서 미분 할 수 있도록 근사 된 모습이다. NengoDL에서는 비 스파이킹 심층 신경망의 구조에 미분 가능한 스파이킹 뉴런을 사용하지 않더라도 [그림 2-10]과 같이 미분 가능한 뉴런모델로 변환시켜주고 스파이킹 심층 신경망 학습을 진행한다.

반면 추론(Inference)을 할 때에는 스파이크 뉴런 모델을 근사하기 전 모델로 되돌린 후에 근사된 스파이킹 심층 신경망 학습을 통해 얻은 파라미터 값을 적용하여 진행하는 방식을 취하고 있다.

제 3 장 ONNX-SNN

제 1 절 ONNX-SNN 제안

본 연구에서 제안하는 ONNX-SNN은 Protobuf기반의 오픈신경망교환포맷을 기반으로 한다. NengoDL에서 지원하는 스파이킹 심층 신경망의 기본적인 구조는 비 스파이킹 심층 신경망 구조와 동일하고 활성화 함수의 차이만 있기 때문이다. 즉 ONNX-SNN은 오픈신경망교환포맷와 기본적인 구조는 동일하며 스파이킹 뉴런을 위한 오퍼레이터의 종류를 추가된 구조이다. 제안하는 ONNX-SNN의 구조에 대한 설명은 [표3-1], [표3-2], [표3-3]과 함께 한다.

[표 3-1] 모델의 컴포넌트 리스트

명칭	타입
ir_version	int64
opset_import	OperatorSetId
producer_name	string
producer_version	string
domain	string
model_version	int64
doc_string	string
graph	Graph
metadata_props	map<string,string>

[표 3-2] onnx-snn.proto파일에 정의되어있는 OperatorSetId

OperatorSetIdProto

```
1:  message OperatorSetIdProto {
2:    optional string domain = 1;
3:    optional int64 version = 2;
4:  }
```

[표 3-3] onnx-snn.proto파일에 정의되어있는 GraphProto

GraphProto

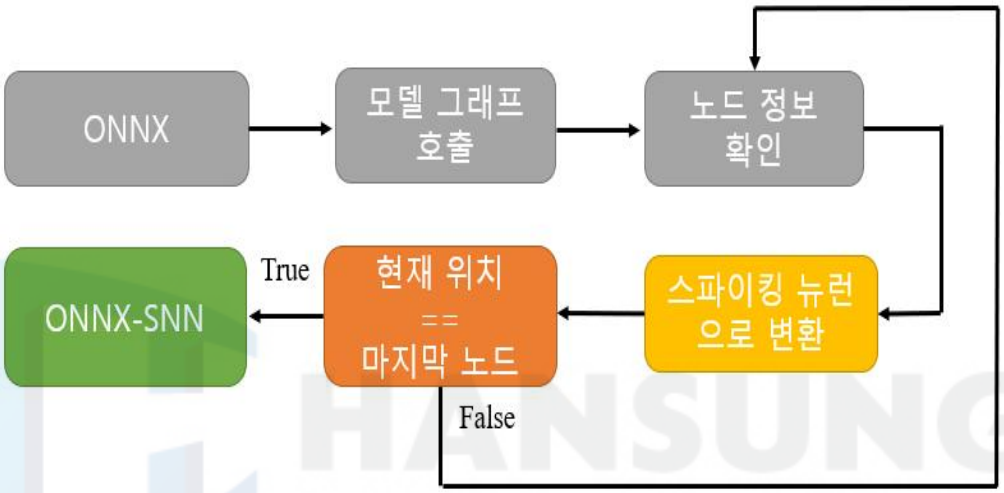
```
1:  message GraphProto {
2:    repeated NodeProto node = 1;
3:    optional string name = 2;
4:    repeated TensorProto initializer = 5;
5:    repeated SparseTensorProto sparse_initializer = 15;
6:    optional string doc_string = 10;
7:    repeated ValueInfoProto input = 11;
8:    repeated ValueInfoProto output = 12;
9:    repeated ValueInfoProto value_info = 13;
10: }
```

ONNX-SNN를 로드하여 모델의 구조를 파악하기 위해서 다양한 컴포넌트들(components)과 속성들(properties)을 사용해야한다. ONNX-SNN가 가지고 있는 컴포넌트는 [표 3-1]와 같다.

ir_version은 ONNX-SNN의 버전 정보를 정수형타입으로 가지고 있다. opset_import는 모델에서 사용할 수 있는 연산자(오퍼레이터) 묶음의 버전 정보를 가지고 있으며 데이터 타입은 OperatorSetId타입으로 되어있다. OperatorSetId은 [표 3-2]과 같이 proto파일에 정의되어 있는 데이터형식이다. opset_import 버전에 따라 지원하는 연산자가 달라진다. producer_name은 모델 생성에 생성된 도구의 이름 정보를 문자열타입으로 가지고 있다. producer_version는 모델 생성 도구의 버전의 버전 정보를 문자열타입으로 가지고 있다. domain는 전통적으로 java에서 사용하던 패키지 지정 방식과 유사하다. 즉 domain정보를 통해 어느 도메인(ex ONNX, ONNX-ML, ONNX-SNN)에 포함되어 있는지를 알 수 있으며 데이터 타입은 문자열이다. model_versions는 ONNX-SNN에 저장된 모델의 버전 정보를 정수형타입으로 인코딩 되어 가지고 있다. doc_string는 이 모델에 대한 설명을 볼 수 있으며 데이터 타입은 문자열이다. graph는 proto파일에 [표 3-3]과 같은 데이터 구조로 정의되어 있다. graph에는 모델을 구성하고 있는 node리스트를 node라는 속성 값으로 가지고 있으며(line 2) 모델의 파라미터들을 initializer라는 속성 값으로 가지고 있다(line 4). node리스트를 구성하는 node안에는

해당 노드의 다양한 속성 값들로 이루어 져 있다. 이 속성 값 중에는 오퍼레이터에 대한 정보도 포함하고 있다. metadata_props는 메타데이터들을 map 데이터 타입으로 key, value의 쌍으로 되어 있다.

제 2 절 ONNX-SNN 변환



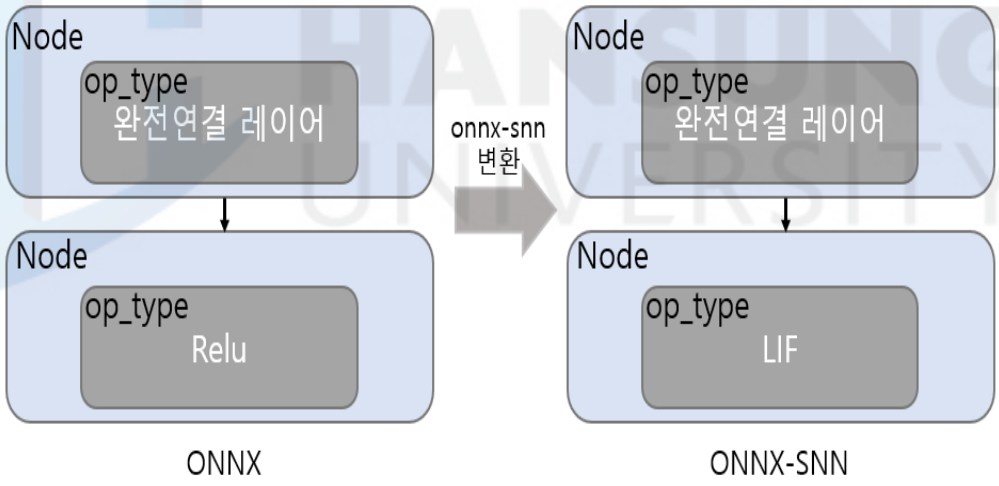
[그림 3-1] ONNX-SNN 변환 과정

[표 3-4] onnx-snn.proto파일에 정의되어있는 NodeProto

NodeProto
1: message NodeProto {
2: repeated string input = 1;
3: repeated string output = 2;
4: optional string name = 3;
5: optional string op_type = 4;
6: optional string domain = 7;
7: repeated AttributeProto attribute = 5;
8: optional string doc_string = 6;
9: }

스파이킹 심층 신경망을 위한 오픈신경망교환포맷 ONNX-SNN을 정의하는 것은 간단하다. 오픈신경망교환포맷과 기본구조는 같고 비 스파이킹 뉴

런에서 사용하는 활성화 함수에 대해서 스파이킹 뉴런모델로 변경해주면 된다. 변환 과정을 [그림 3-1]를 통해 설명하면 다음과 같다. 우선 오픈신경망 교환포맷을 로드하고 모델의 graph컴포넌트를 호출한다. 3장 1절 [표3-3]에서 설명했듯이 graph에는 모델을 구성하는 node리스트를 node라는 속성 값으로 가지고 있다. node의 구조는 [표 3-4]와 같다. 텐서플로와 비교하여 보면 node는 텐서플로의 tensornode와 유사한 기능을 하며 [표 3-4]에서 op_type은(line 5) 텐서플로의 tensor타입과 유사한 기능을 한다. 즉 오퍼레이터 타입정보를 op_type를 통해 알 수 있다. 최상단 노드부터 마지막 노드까지 node리스트를 순회하면서 op_type을 조회하고 op_type의 값을 비 스파이킹 뉴런에서 보편적으로 사용하는 오퍼레이터 중 활성화 함수와 관련된 Relu, Sigmoid, Tanh일 경우에 스파이킹 뉴런모델(LIF, SoftLIFRate)으로 변환하는 방법을 사용한다.



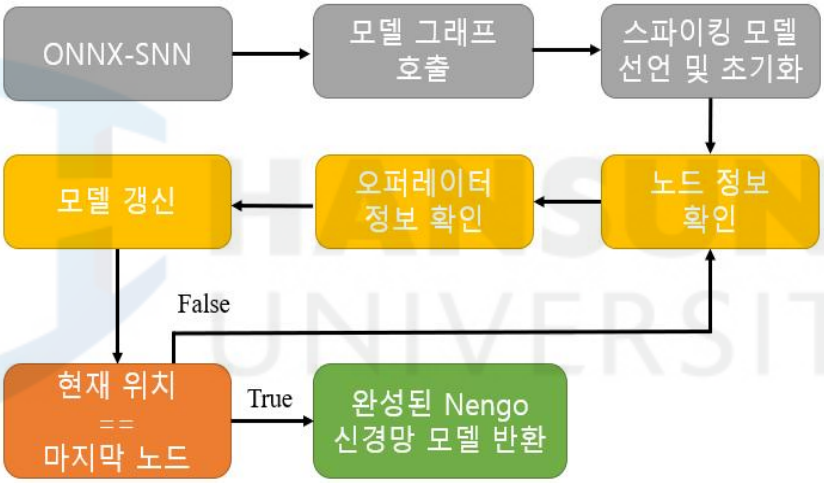
[그림 3-2] ONNX(좌)과 ONNX-SNN(우) 비교

[그림 3-2]는 완전연결신경망(fully connected neural network)모델을 오픈신경망교환포맷과 ONNX-SNN으로 변환 후 두 포맷을 이미지화한 모습이다. [그림 3-2]의 좌측은 오픈신경망교환포맷이고 [그림 3-2]의 우측은 오픈신경망교환포맷을 ONNX-SNN으로 변환시킨 모습이다. [그림 3-2]에서 비 스파이킹 뉴런 오퍼레이터로 사용된 Relu가 스파이킹 뉴런으로 보편적으

로 사용되는 LIF뉴런 모델로 교체되어 ONNX-SNN으로 변환된 모습을 볼 수 있다. [그림 3-2]에서 보여준 LIF뿐만 아니라 다른 스파이킹 뉴런으로도 변경할 수 있다.

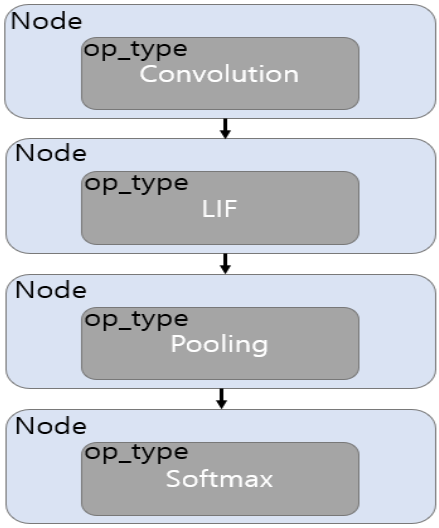
제 4 장 모델 및 코드 생성

본 장에서는 오픈신경망교환포맷에서 ONNX-SNN로 변환된 모델을 로드하여 Nengo 프레임워크에서 동작 할 수 있는 모델 생성 및 Nengo 프레임워크에서 사용할 수 있는 코드 생성 방법에 대해 [그림 4-1]를 통해 설명한다. 첫 번째로 ONNX-SNN으로 변환시킨 모델을 로드한다. 로드가 완료되면 Nengo 프레임워크에 생성할 신경망 모델로 사용할 변수를 선언하고 초기화를 진행한다.



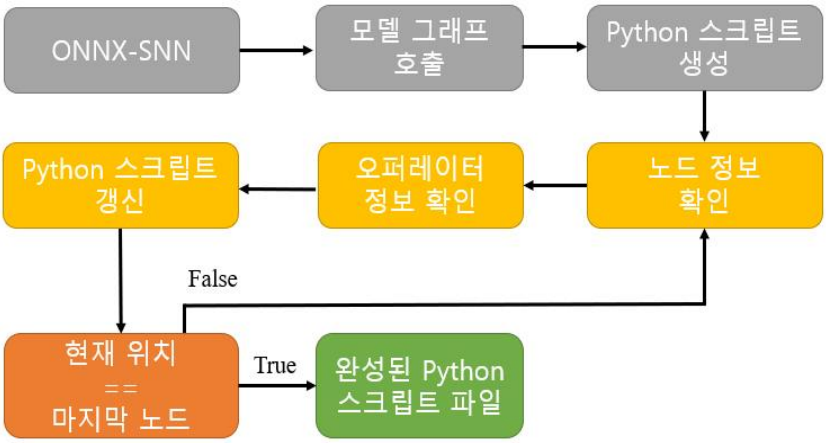
[그림 4-1] 스파이킹 신경망 모델 생성 과정

노드의 정보 확인 과정에서는 로드한 모델의 graph컴포넌트를 호출하고 graph컴포넌트에 있는 node속성 값을 호출한다. node속성을 호출하는 이유는 node속성 값에 node리스트 데이터를 얻기 위함이다. node리스트는 첫 번째 노드부터 마지막 노드까지 순회하면서 노드정보를 확인하는데 사용된다. 노드들을 순회를 하면서 각각의 노드의 오퍼레이터 정보 확인 과정을 거친다. 오퍼레이터 정보는 node의 op_type속성 값에서 확인한다. 1개의 node에는 [그림 4-2]와 같이 1개의 op_type값만이 존재할 수 있다.



[그림 4-2] ONNX-SNN Node 시각화 예시

만약 모델 갱신과정에서 op_type이 합성곱연산이나 폴링연산일 경우 해당 Node의 attribute속성에서 커널형태(Kernel Shape)와 Stride정보도 함께 가져온다. 갱신하는 과정에서 마지막으로 노드로 들어오는 데이터 형태(Input Shape)에 대해서 결과물로 나가는 데이터 형태(Output)를 계산해 주어야 한다. 왜냐하면 데이터 형태를 각각의 노드마다 속성 값으로 가지고 있지 않기 때문이다. 이 같은 과정이 완료되면 수집한 정보를 기반으로 Nengo, NengoDL API(Application Programming Interface)를 사용하여 스파이킹 신경망 모델을 갱신한다. 이 과정을 반복하면서 마지막 노드까지 도달하면 Nengo 프레임워크에서 스파이킹 신경망 모델이 완성된다. 본 연구에서는 이러한 과정 클래스 형태로 구현했다. ONNX-SNN파일 경로 정보만 구현한 변환 클래스로 넘겨주면 자동으로 Nengo 프레임워크에서 사용할 수 있는 스파이킹 신경망 모델로 변환시켜주고 변환된 모델로 바로 학습 및 추론이 가능하다.



[그림 4-3] 스파이킹 신경망 코드 생성 과정

ONNX-SNN을 Nengo 프레임워크에서 사용할 수 있는 코드 형태로 모델을 구현해주는 방법([그림 4-3])도 모델 생성방법과 메커니즘은 동일하다. 다만 변환 툴을 구현하는 과정에서 직접 Nengo 프레임워크에 맞는 Python 스크립트를 생성해주는 방식이다.

본 연구에서 모델과 코드를 생성하는 두 가지 방식을 모두 취하는 이유는 NengoDL에서는 변환된 모델에 노드를 추가하는 것은 가능하지만 이미 들어가 있는 노드를 수정하거나 삭제할 수 없다. 이러한 문제점 때문에 nengo 문법에 맞는 모델을 그대로 구현할 수 있는 코드도 함께 생성 하는 방법을 취하게 되었다.

[표 4-1] 변환 툴을 이용한 모델 사용 예시코드

변환 툴 사용 예시 코드	
1:	import onnxToNengoModel
2:	tool = onnxToNengoModel.toNengoModel(onnx_path)
3:	model = tool.get_model()
4:	model.train(train_set)
5:	model.predict(test_set)

[표 4-1]는 [그림 4-1]과정을 구현한 변환 툴을 이용해 모델로 변환하고 모델을 반환코드이다. line 1에서 import한 onnxToNengoModel.py는 본 연구를 통해 구현한 모델 변환 툴이다. line 2에서는 변환 툴이 ONNX-SNN

경로정보를 받아 get_model()메소드를 통해 모델을 반환받는다(line 3). 그리고 line 4에서 학습 데이터를 반환 받은 모델에 학습하고 line 5에서 학습한 모델에 추론을 진행한다.

[그림 4-4]는 [그림 4-3]과정을 구현한 변환 툴을 이용해 ONNX-SNN으로 변환된 LeNet-5모델을 Nengo 프레임워크에 맞게 코드로 변환한 모습이다. 복잡한 신경망의 경우 변환 툴을 사용하지 않고 직접 NengoDL이 지원하는 API로 다시 신경망을 구성하려면 많은 시간이 소요된다. 변환 툴을 사용하면 빠르게 비 스파이킹 심층 신경망을 스파이킹 심층 신경망으로 변환하여 학습 및 추론을 진행 할 수 있게 된다.

```
with nengo.Network(seed=1000) as net:
    net.config[nengo.Ensemble].max_rates = nengo.dists.Choice([100])
    net.config[nengo.Ensemble].intercepts = nengo.dists.Choice([0])
    default_neuron_type = nengo.LIF(amplitude=0.01)
    nengo_dl.configure_settings(trainable=False)

    inp = nengo.Node([0] * 28 * 28 * 1)
    x = inp

    x = nengo_dl.tensor_layer(x, tf.layers.conv2d, shape_in=(28, 28, 1), filters=4, kernel_size=5, padding="same")
    x = nengo_dl.tensor_layer(x, nengo.LIF(amplitude=0.01))

    x = nengo_dl.tensor_layer(x, tf.layers.average_pooling2d, shape_in=(28, 28, 4), pool_size=2, strides=2)

    x = nengo_dl.tensor_layer(x, tf.layers.conv2d, shape_in=(14, 14, 4), filters=16, kernel_size=5, padding="valid")
    x = nengo_dl.tensor_layer(x, nengo.LIF(amplitude=0.01))

    x = nengo_dl.tensor_layer(x, tf.layers.average_pooling2d, shape_in=(10, 10, 16), pool_size=2, strides=2)

    x = nengo_dl.tensor_layer(x, tf.layers.conv2d, shape_in=(5, 5, 16), filters=120, kernel_size=5, padding="valid")
    x = nengo_dl.tensor_layer(x, nengo.LIF(amplitude=0.01))

    x = nengo_dl.tensor_layer(x, tf.layers.dense, units=84)
    x = nengo_dl.tensor_layer(x, nengo.LIF(amplitude=0.01))

    x = nengo_dl.tensor_layer(x, tf.layers.dense, units=10)
```

[그림 4-4] ONNX-SNN에서 변환된 신경망 코드

제 5 장 실험

제 1 절 실험 환경

[표 5-1] 데스크톱 사양 목록

	사양
CPU	i9-9900k(8코어 16스레드)
GPU	gtx1080 1way
RAM	32Gbyte
저장공간	SSD 1TB

본 절에서는 실험 환경에 대해 설명한다. 실험은 [표 5-1]사양의 개인 데스크톱 환경에서 진행했다. CPU는 8코어 16스레드인 i9-9900k, GPU는 gtx1080, RAM은 32Gbyte, 저장공간은 SSD 1TB를 사용했다.

[표 5-2] 필수 Python라이브러리 목록

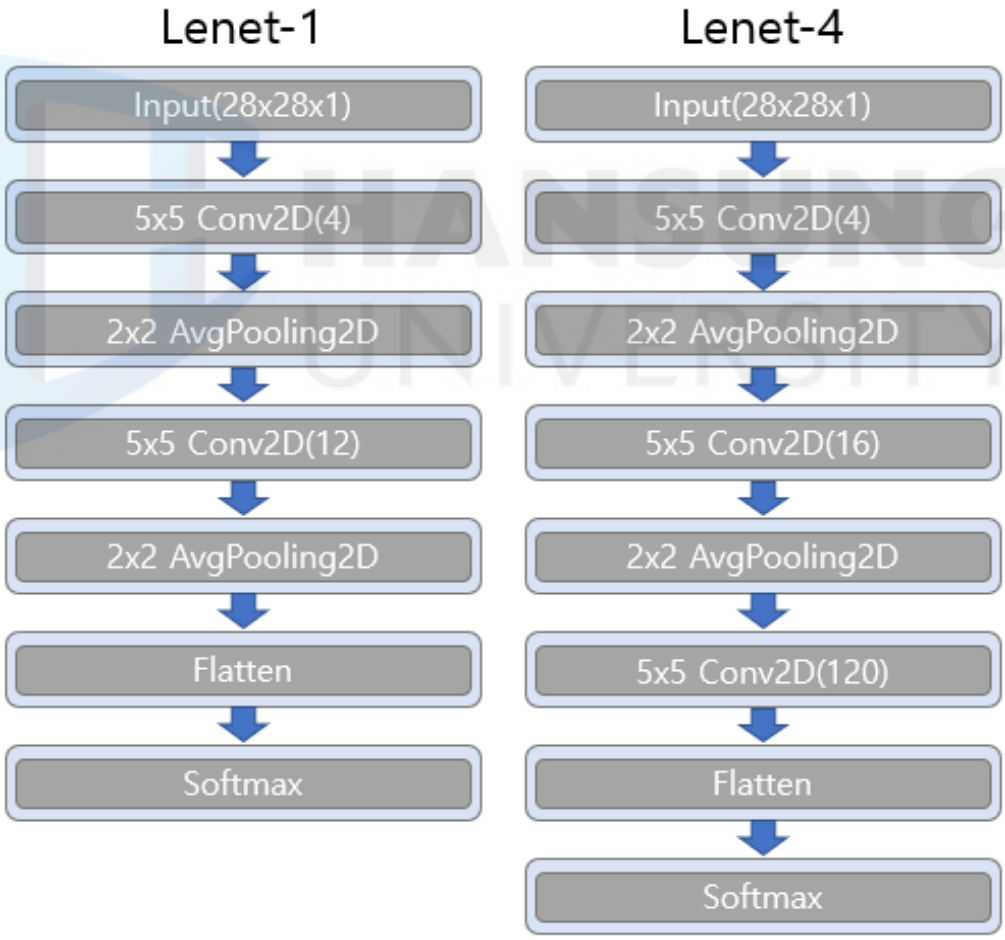
	사양
Tensorflow	1.10.0
keras	2.2.2
keras2onnx	1.6.0
nengo	2.8.0
NengoDL	2.2.0
onnx	1.6.0
numpy	1.14.5
Protobuf	3.6.0
cudatoolkit	9.0
cudnn	7.6.4

구현에 사용된 프로그래밍 언어는 Python(3.5.6버전)이다. 사용된 라이브러리 환경은 [표 5-2]와 같다. NengoDL은 2.2.0버전은 Tensorflow의 2.0버전을 지원하고 있지 않다는 점을 주의해야한다. 또한 ONNX-SNN은 오픈신경망교환포맷 1.6.0버전을 기반으로 한다.

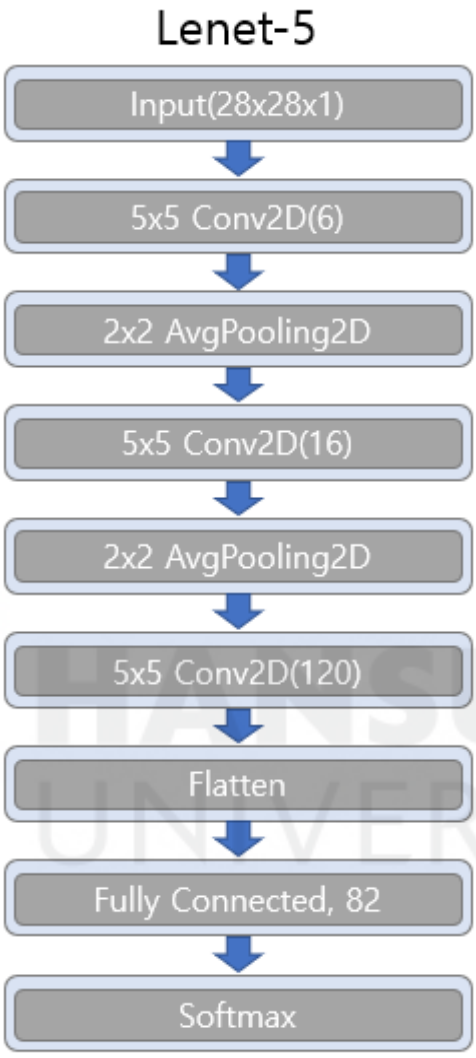
제 2 절 ONNX에서 ONNX-SNN 변환

본 절에서는 ONNX를 ONNX-SNN으로 직접 구현한 변환 툴을 사용해 변환하고 변환한 결과를 보여준다. ONNX-SNN으로 변환하기에 앞서 ONNX로 변환시킬 비 스파이킹 심층 신경망은 [그림 5-1], [그림 5-2]이다.

LeNet-1, LeNet-4, LeNet-5모델을 Keras를 사용하여 신경망 모델을 만들었다. LeNet-1와 LeNet-4구조는 [그림 5-1], LeNet-5구조는 [그림 5-2]와 같다.

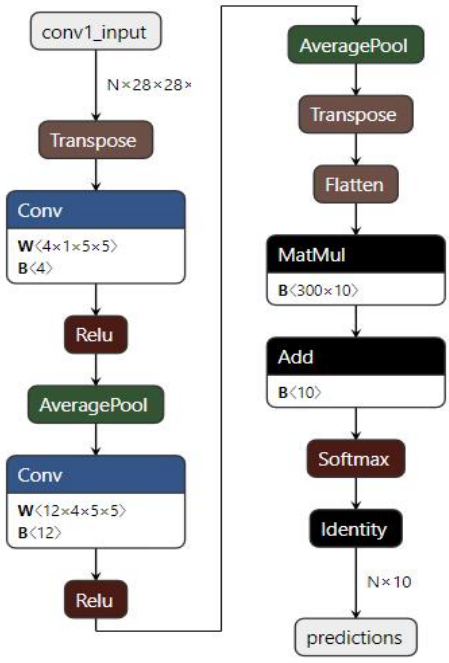


[그림 5-1] LeNet-1, LeNet-4 구조

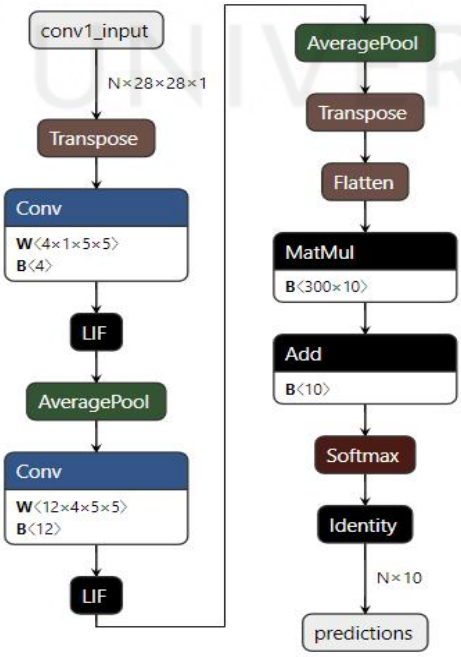


[그림 5-2] LeNet-5 구조

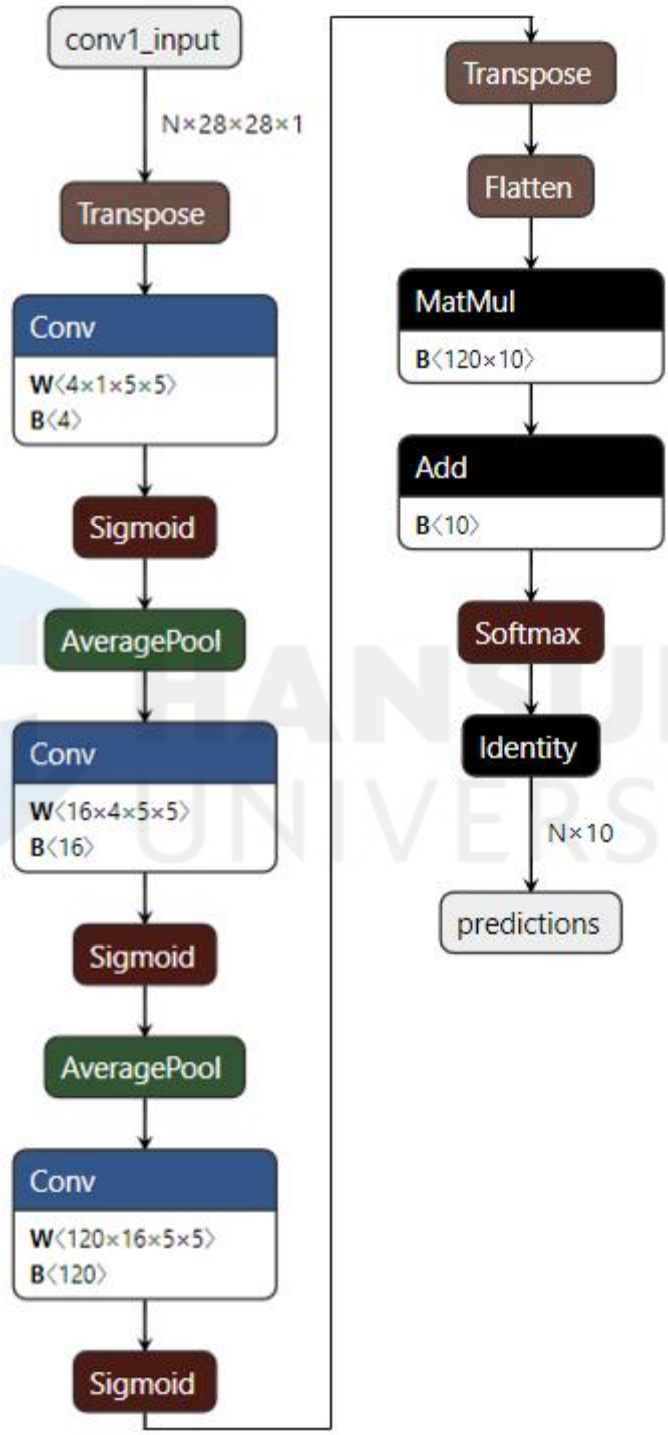
keras는 생성된 모델을 ONNX로 변환시키는 툴을 keras2onnx라이브러리를 통해 지원하고 있다. keras2onnx의 convert_keras메소드를 통해 변환 가능하다.



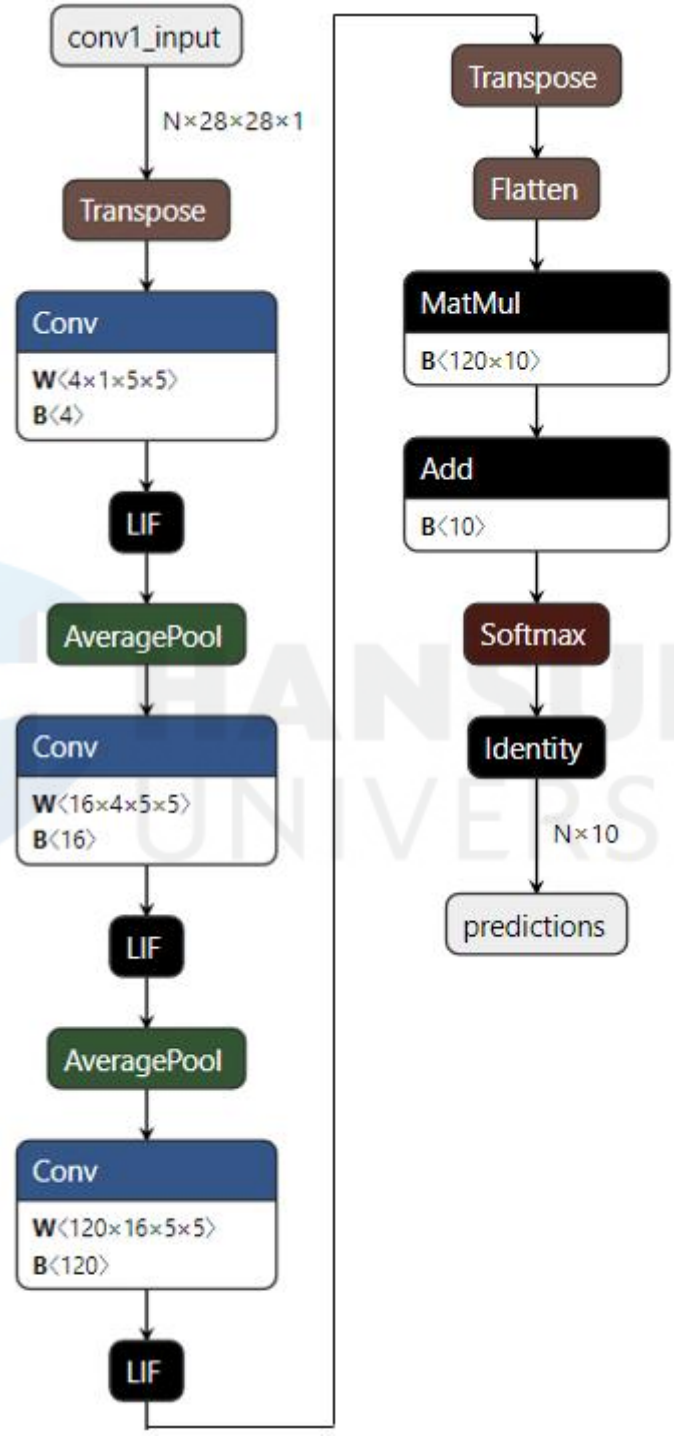
[그림 5-3] LeNet-1(ONNX) 시각화 이미지



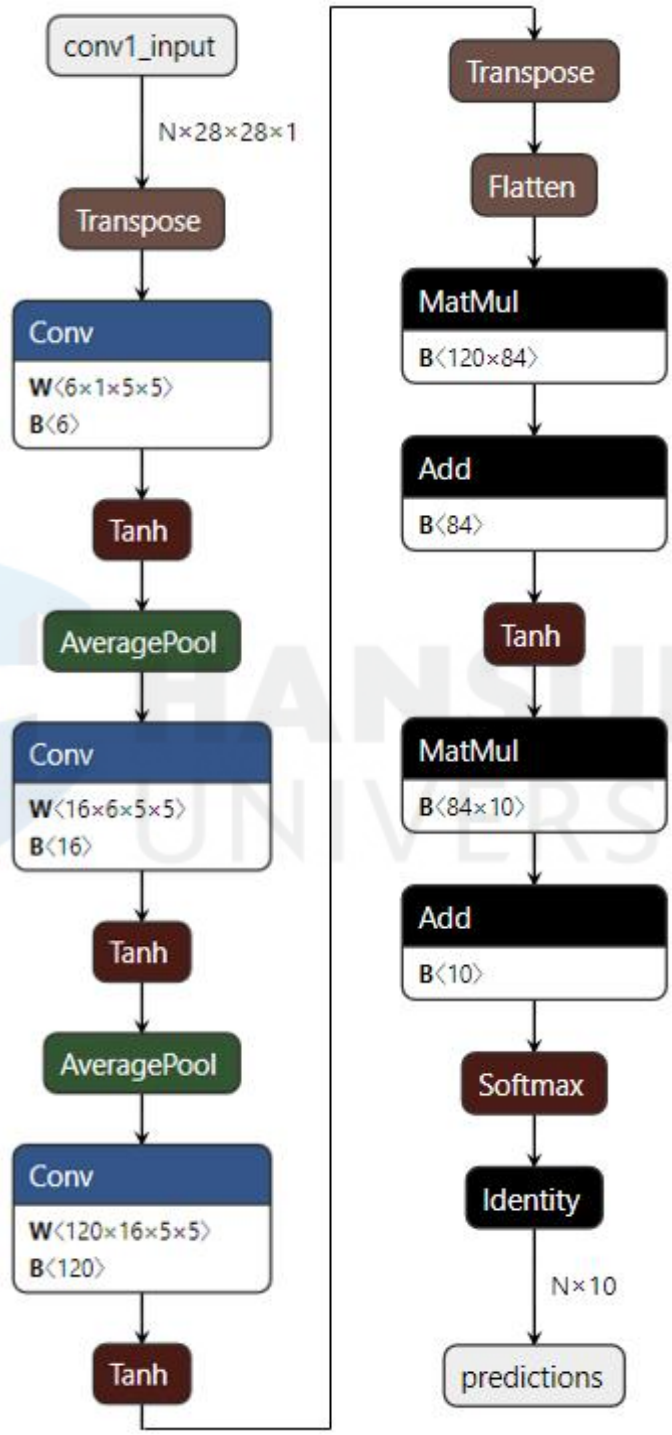
[그림 5-4] LeNet-1(ONNX-SNN) 시각화 이미지



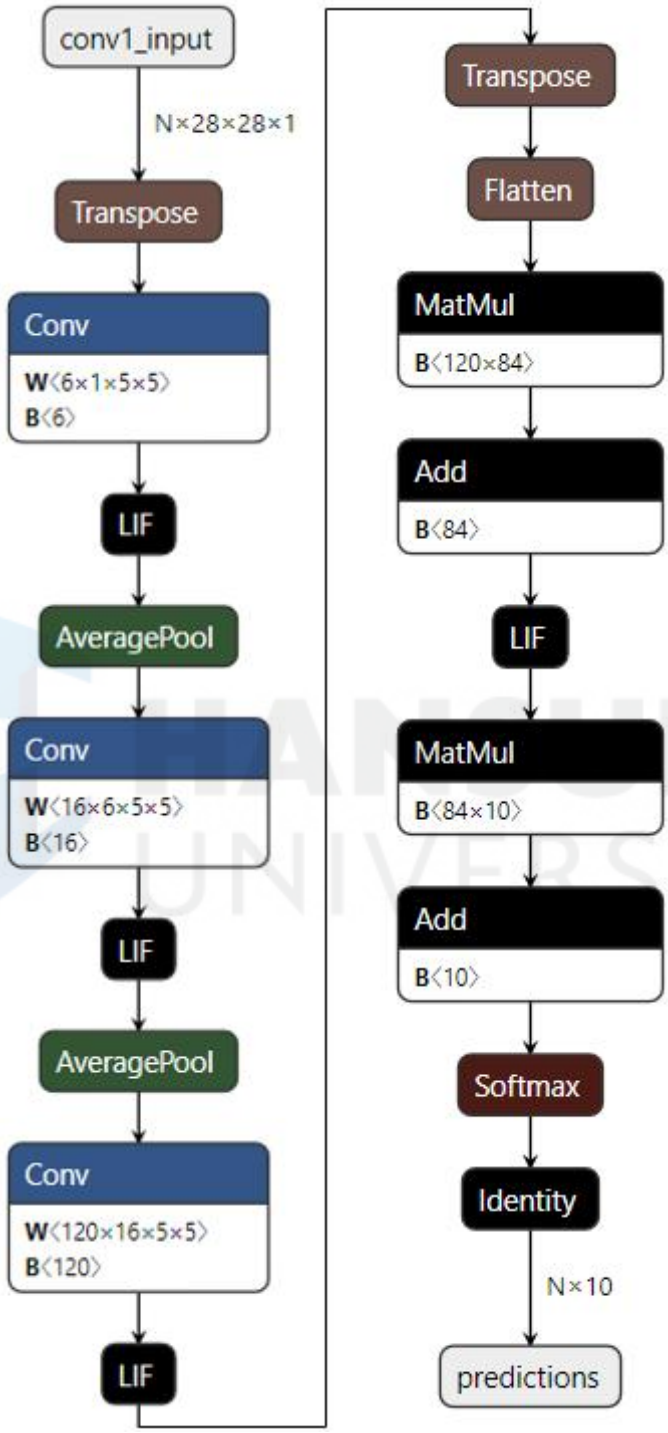
[그림 5-5] LeNet-4(ONNX) 시각화 이미지



[그림 5-6] LeNet-4(ONNX-SNN) 시각화 이미지



[그림 5-7] LeNet-5(ONNX) 시각화 이미지



[그림 5-8] LeNet-5(ONNX-SNN) 시각화 이미지

ONNX-SNN으로의 변환은 3장 2절에서 설명한 방법을 통해서 오퍼레이터를 스파이킹 뉴런모델로 교체한다. ONNX-SNN으로 변환되면서 달라진 부분을 시각화 된 [그림 5-3]과 [그림 5-4], [그림 5-5]과 [그림 5-6], [그림 5-7]과 [그림 5-8]을 통해 알 수 있다. [그림 5-3]의 활성화함수 Relu가 [그림 5-4]에서 LIF로, [그림 5-5]의 활성화함수 Sigmoid가 [그림 5-6]에서 LIF로 [그림 5-7]의 활성화함수 Tanh가 [그림 5-8]에서 LIF로 변환된 것을 확인 할 수 있다. 시각화하기 위해 사용한 응용프로그램은 Netron이다. Netron은 다양한 인공지능 포맷을 시각화해준다.

제 3 절 비 스파이킹/스파이킹 심층 신경망 추론결과 비교

본 절에서는 ONNX-SNN을 Nengo 프레임워크에서 사용할 수 있는 모델 로 변환하여 비 스파이킹 심층 신경망과 스파이킹 심층 신경망과의 추론 정확도 차이를 비교한다. 실험에서 사용할 모델구조는 LeNet으로 진행한다. 학습 및 추론에 사용하는 데이터는 Mnist이고 학습데이터 6만개, 테스트데이터로 1만개를 사용한다.

[표 5-3] 활성화 함수별 오차율

	LeNet-1	LeNet-4	LeNet-5
Tanh	0.06%	0.04%	0.34%
Sigmoid	0.18%	0.23%	0.34%
Relu	0.06%	0.06%	0.06%
LIF	3.58%	1.71%	3.32%
SoftLIFRate	1.07%	1.03%	1.09%

하이퍼파라미터(hyper-parameters)설정은 학습률(learning_rate)은 0.01, epoch은 100, 배치사이즈(batchsize)는 200으로 모두 동일하게 설정하고 [표 5-3] 실험을 진행했다. [표 5-3]은 각 심층 신경망별로 활성화 함수를 다르

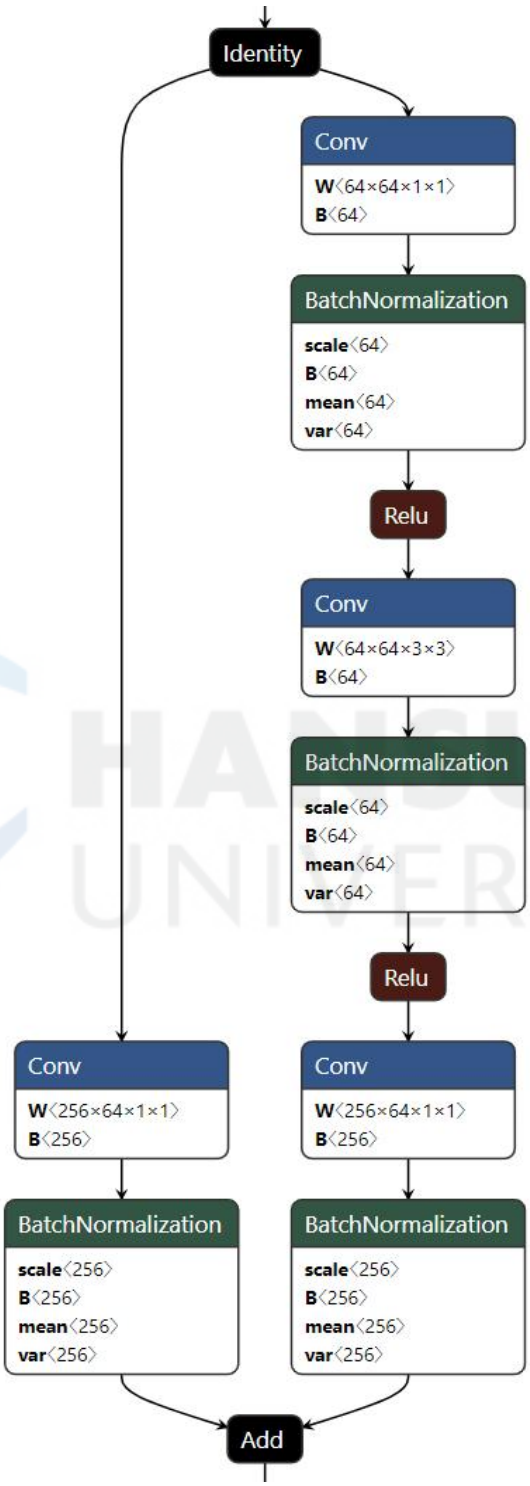
게 하여 실험한 결과이다. 비 스파이킹 신경망에서 사용하는 Tanh, Sigmoid, Relu 활성화 함수와 스파이킹 신경망에서 주로 사용하는 LIF, SoftLIFRate 활성화 함수가 실험 대상이다.

전체적으로 비 스파이킹 뉴런에서 사용하는 활성화 함수를 신경망에 적용했을 때 더 낮은 오차율[표 5-3]이 나왔다. 하지만 Hunsberger, E.,와 Eliasmith, C (2015)에서 Mnist를 대상으로 실험 결과 값과 비교하면 ONNX-SNN 변환 과정을 거쳐 LIF와 SoftLIFRate 활성화 함수를 사용했을 때 오차율은 Hunsberger, E.,와 Eliasmith, C (2015)에서의 실험결과 오차율과 근사치인 것을 확인 할 수 있다. 이를 통해 스파이킹 심층 신경망으로 변환 과정에 문제점이 없다는 것을 확인 할 수 있다. 다만 활성화 함수로 LIF를 사용할 경우 정답률이 낮은 이유는 2장 4절에서 설명했던 이유에서 알 수 있다. NengoDL은 스파이킹 심층 신경망을 학습할 때에 스파이킹 뉴런 모델을 발화율 기반으로 미분가능 하도록 뉴런모델(활성화함수)을 근사시키는 방법을 사용한다(LIF모델 -> 근사된 LIF모델). 추론 과정에서는 근사하기 전 설정한 활성화 함수 상에서 추론이 진행된다. 이 과정에서 LIF로 뉴런모델을 설정한 경우 학습할 때는 발화율 기준으로 근사된 LIF모델을 사용하고 추론 과정에서 LIF모델을 사용한다. 뉴런모델이 달라지면서 SoftLIFRate모델을 사용했을 때보다 오차율이 높아지는 것을 [표 5-3]을 통해 확인 할 수 있다. 반면 SoftLIFRate는 LIF를 근사시킨 뉴런 모델이므로 추론과정에서도 SoftLIFRate모델을 사용하기 때문에 LIF로 뉴런모델을 설정했을 때 보다 낮은 오차율을 보여준다.

제 6 장 결론 및 향후 개선 방안

본 논문은 비 스파이킹 신경망에서 사용하던 신경망 모델을 스파이킹 신경망에서 쉽게 변환하여 사용할 수 있도록 ONNX-SNN을 제안하고 ONNX-SNN을 Nengo프레임워크에서 사용 할 수 있는 모델 또는 코드로 변환 시켜주는 방법에 대해 제안하였다. ONNX-SNN과 변환 툴을 사용하면 프레임워크마다 별도의 변환작업을 할 필요가 없다. 또한 변환 툴을 통해 변환된 모델의 경우 바로 학습 및 추론하는 것에 적합하고 변환된 코드는 모델을 수정하기에 적합하다. 변환 툴 사용자는 단지 하이퍼파라미터를 변경하면서 오차율이 낮아지는 값을 찾아내는 작업만을 수행하면 된다.

향후 개선 방안은 첫 번째 더욱 다양한 신경망 구조에서 변환이 가능해야 한다는 것이다. 본 논문의 실험은 순차적(Sequential)구조의 심층 신경망 LeNet을 대상으로 수행되었다. 순차적 구조 신경망에서 진행한 이유는 NengoDL의 2.2.0버전 까지는 Tensorflow의 concat메소드를 지원하지 않는다. ResNet과 같이 분기가 있는 심층 신경망의 경우 다시 병합([그림 6-1]) 과정이 필요한데 concat메소드를 사용할 수 없어 ResNet과 같이 분기가 있는 심층 신경망은 NengoDL으로 구현 할 수 없다. 이러한 문제점은 NengoDL 향후 concat기능을 구현하거나 concat기능을 지원하는 버전이 배포되면 이에 맞추어 개선할 예정이다. 두 번째 본 논문에서 ONNX에서 NengoDL상에 신경망으로 변환시키는 과정을 제시하였다면 향후 개선방향으로 NengoDL로 만든 신경망 모델을 ONNX으로 변환시키는 변환 툴을 연구할 예정이다.



[그림 6-1] ResNet-50(ONNX) 분기와 병합 일부분 시각화

참 고 문 헌

1. 국외문헌

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016). *Tensorflow: A system for large-scale machine learning*. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16) (pp. 265-283).

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., ... & Eliasmith, C. (2014). *Nengo: a Python tool for building large-scale functional brain models*. Frontiers in neuroinformatics, 7, 48.

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., ... & Boahen, K. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE,* 102(5), 699-716.

Brader, J. M., Senn, W., & Fusi, S. (2007). *Learning real-world stimuli in a neural network with spike-driven synaptic dynamics*. Neural computation, 19(11), 2881-2912.

Cao, Y., Chen, Y., & Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision,* 113(1), 54-66.

Collobert, R., Kavukcuoglu, K., & Farabet, C. (2011). *Torch7: A matlab-like environment for machine learning*. In BigLearn, NIPS workshop (No. CONF).

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S. C., & Pfeiffer, M. (2015, July). *Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing*. In 2015 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

Eliasmith, C. (2012). *A large-scale model of the functioning brain (vol 338, pg 1202, 2012)*. Science, 338(6113), 1420-1420.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT

press.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science,* 313(5786), 504-507.

Hunsberger, E., & Eliasmith, C. (2015). *Spiking deep networks with LIF neurons*. arXiv preprint arXiv:1510.08829.

Hunsberger, E., & Eliasmith, C. (2016). *Training spiking deep networks for neuromorphic hardware*. arXiv preprint arXiv:1611.05141.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). *Caffe: Convolutional architecture for fast feature embedding*. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678). ACM.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE,* 86(11), 2278-2324.

Lee, J. H., Delbruck, T., & Pfeiffer, M. (2016). *Training deep spiking neural networks using backpropagation*. Frontiers in neuroscience, 10, 508.

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., & Cauwenberghs, G. (2014). *Event-driven contrastive divergence for spiking neuromorphic systems*. Frontiers in neuroscience, 7, 272.

O'Connor, P., Neil, D., Liu, S. C., Delbruck, T., & Pfeiffer, M. (2013). *Real-time classification and sensor fusion with a spiking deep belief network*. Frontiers in neuroscience, 7, 178.

Rasmussen, D. (2019). *NengoDL: Combining deep learning and neuromorphic modelling methods*. Neuroinformatics, 1-18.

Team, T. T. D., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., ... & Belopolsky, A. (2016). *Theano: A Python framework for fast computation of mathematical expressions*. arXiv preprint arXiv:1605.02688.

ABSTRACT

ONNX for the spiking deep neural network

Park, Sang-Min

Major in Computer Engineering

Dept. of Computer Engineering

The Graduate School

Hansung University

Spiking Neural Network(SNN) is a third-generation natural network and operations with a different mechanism from the first and second-generation natural networks. The non-spiking neural network Single Layer Perceptron(SLP) and the second generation neural network Multi Layer Perceptron(MLP) export output values through active functions that do not consider biological mechanisms for input values. In contrast, the third-generation neural network, Spiking Neural Networks, operates on a neuron model that is most similar to the biological mechanism of the actual neuron than to the existing activation function. However, even if the network is not used, there have been good results in the existing non-spiking neural network. Inspired by this, there have been attempts to take a non-spiking deep neural network model structure into the spiking neural network and replace only the model of neuron with a spikng neuron. However, due to different data formats for each

artificial intelligence framework, the same neural network structure is expressed in different data formats in each framework. Because of this problem, the process of converting to another framework is necessary. This problem was solved by using ONNX(Open Neural Network Exchange) in previous artificial intelligence frameworks. It converts models made in other AI frameworks into ONNX. and The conversion tool is used to convert to a data format able for the artificial intelligence framework that you want to convert. That is, a framework that supports ONNX. can be used simply by converting an artificial intelligence model. However, the framework that supports the Spike Neural Network does not currently support ONNX, nor does ONNX have a definition of a Spike neuron. The contents proposed in this paper are as follows. First, ONNX-SNN is proposed to enable the expression of the neural network of spikes based on ONNX. In addition, the conversion method from ONNX to ONNX-SNN is introduced. Second, A model conversion and code generation method is proposed that ONNX-SNN can be used in the Nengo framework.

KEYWORD: Perceptron, Spiking Neural Network, Open Neural Network Exchange