임베디드 기기에서 ONNX Runtime 성능 비교와 새로운 Runtime 의 가능성

김성민*, 범정현**, 추현승**
성균관대학교 인공지능학과*
성균관대학교 소프트웨어대학**
semih@ssku.edu, bumjh@skku.edu, choo@skku.edu

Performance comparison of ONNX Runtime on embedded device and possibility of new runtime

Sungmin Kim*, Junghyun Bum**, Hyunseung Choo**
Dept. of Artificial Intelligence, Sungkyunkwan University*
College of Computing, Sungkyunkwan University**

요 약

ONNX 은 인공신경망 모델 교환을 위한 표준 중 하나이다. 인공신경망 모델을 구현하는 연구자 입장에선 ONNX 형태로 모델을 배포함으로써 이질적인 플랫폼 간의 호환성을 보장받을 수 있다. 서로 다른 플랫폼에서 ONNX 표준에선 ONNX 모델을 실행하는 엔진을 ONNX Runtime 이라고 하는데, ONNX Runtime 은 순수 S/W 형태이거나, 다양한 H/W 가속 기술과 결합된 형태가 있다. 본 논문에선 ONNX Backend Scoreboard 에 등록 되어있는 3 종류의 엔진과 본 논문에서 새롭게 제안하는 C-ONNX 의 성능을 윅스테이션과 임베디드 기기에서 비교해보고 임베디드 기기에 특화된 C-ONNX 의 가능성에 대해 알아본다.

1. 서론

ONNX(Open Neural Network eXchange)[1]은 인공신경망 모델을 표준화하기 위해 제안된 규격이다. ONNX 외에 OpenGL 로 유명한 Khronos Group 의 NNEF[2]가 ONNX 에 앞서 표준으로 제안된 바 있다. 현재 ONNX 은 PyTorch 나 TensorFlow 와 같은 다양한 머신러닝 프레임워크와 nVidia 의 Jetson 플랫폼에서 적극적으로 지원되며 사실상의 AI(Artificial Intelligence) 모델 표준으로 인정받고 있다.

ONNX 은 인공신경망 모델의 표준이기 때문에 ONNX 형태로 배포된 모델을 다양한 환경에서 구동하기 위한 시도가 있다. 예를 들어 Microsoft 의 ONNX Runtime [3] [1]은 ONNX 을 고속으로 실행시키는데 최적화된 엔진으로 다양한 AI 가속 기술을 활용해 빠른 실행 속도를 보이는 것이 특징이다. 그 외에도 Google 의 Tensorflow 를 ONNX Runtime 으로 활용하는 onnx-tf[4], 그리고 Intel 의 nGraph[5]가 ONNX Runtime 의 일종이다.

[1] Microsoft 의 ONNX Runtime 은 오픈소스 소프트웨어로 ONNX 에서 정의하는 ONNX Runtime 과 동일한 이름을 사용하고 있다.

2. 임베디드 시스템에서 ONNX Runtime 필요성

인공신경망을 훈련시킬 때는 Tensorflow 나 PyTorch 와 같은 머신 러닝 프레임워크를 활용한다. 따라서 머신러닝 프레임워크가 구동되는 서버 또는 윅스테이션 환경에선 인공신경망 모델을 훈련시킬 뿐만 아니라 실행시키는데도 아무 문제가 없다. 왜냐하면 인공신경망을 훈련시키는 과정엔 인공신경망을 실행시키는 과정이 포함되기 때문이다.

하지만 임베디드 기기의 경우 인공신경망을 훈련시키는 용도로 활용하는 경우가 드물기 때문에 이미 훈련이 완료된 모델을 실행시킬 수 있는 실행에 특화된 엔진이 필요하다. 임베디드 기기에 특화된 대표적인 ONNX Runtime 으론 nVidia 의 Jetson 보드가 있다.

3. C-ONNX – 임베디드 기기를 위한 새로운 Runtime

본 논문에선 임베디드 기기를 위한 새로운 ONNX Runtime 인 C-ONNX[6]을 제안한다. C-ONNX 은 “C 로 구현한 ONNX Runtime”의 약자로 다수의 임베디드 시스템이 C 또는 C++로 구현된 Firmware 만 구동 시킬 수 있다는 점과 GPU 가 없다는 점에 착안해 C 로

ONNX Runtime 을 구현하고, GPU 가 아닌 CPU 에 최적화된 코드를 구현하였다.

C-ONNX 은 IoT(Internet of Things)와 임베디드 기기를 위한 ONNX Runtime 으로 몇 가지 디자인 패턴을 구현하였다. 첫째, C-ONNX 은 ONNX 을 구동하는 엔진으로 모델의 학습보다는 모델의 실행에 초점을 맞춘다. 둘째, C-ONNX 은 C 언어를 기반으로 하여 IoT 기기나 임베디드 기기에 최적화되었고, 다양한 기기에 포팅될 수 있는 최적의 형태를 취하고 있다. 또한 PAL(Platform Abstraction Layer)을 두어 다양한 플랫폼에 최소한의 수정만으로 포팅될 수 있도록 하였다. 셋째, C-ONNX 은 개발자 웍스테이션에서 모델을 배포하고자 하는 기기에 최적화하는 onnx-connx 툴과 임베디드 기기에서 기기에 최적화된 모델을 구동하는 connx 으로 나누어진다. 2 가지 기능을 분리함으로써 코드의 복잡도를 줄이고, 임베디드 기기에 올라가는 라이브러리의 크기를 줄일 수 있다. 넷째, C-ONNX 은 컴파일 시점에 필요로 하는 오퍼레이터를 선택할 수 있도록 함으로써 바이너리의 크기를 줄일 수 있도록 한다. 마지막으로 C-ONNX 은 기기에 최적화된 모델의 특성을 활용해 저장 공간으로부터 인공신경망의 아키텍처와 파라미터를 로딩할 때 별도의 변환과정 없이 기기의 CPU 에 최적화된로 데이터를 로딩할 수 있다. 따라서 다른 기술에 비해 빠른 boot-up 시간을 보장할 수 있다.

4. ONNX Runtime 간 성능 비교

ONNX 프로젝트를 관리하는 리눅스 재단은 'ONNX Backend Scoreboard'라는 툴을 활용해 표준으로서 정의한 ONNX 과 ONNX Runtime 간의 표준 준수성을 점수로 표현한다. ONNX Backend Scoreboard 에 올라온 목록은 Microsoft 에서 만든 ONNX Runtime(이하 ORT), Google 에서 관리하는 Tensorflow 에 ONNX API 를 추가한 ONNX-TF 그리고 Intel 의 nGraph 이다.

본 논문에선 ONNX Runtime 간의 성능 검증을 위해 2 가지 H/W 플랫폼과 2 가지 인공신경망 모델, 그리고 4 가지 ONNX Runtime 을 활용해 성능을 검증하였다. 먼저 H/W 플랫폼으론 서버나 웍스테이션으로 활용되는 CPU 인 Intel 의 Core i7 과 임베디드 시스템에서 활용되는 ARMv7 을 탑재한 Raspberry Pi 3 를 활용하였다. 운영체제는 각각 Ubuntu 20.04 64-bit 과 Raspberry Pi OS 32-bit 을 사용하였다. 인공신경망 모델은 ONNX Model Zoo[7]의 MNIST 와 Mobilenet 을 기준으로 실험을 진행하였다. 2 가지 모델을 선택한 이유는 임베디드 환경에서 실행되기에 최적화된, 크기가 작은 모델이기 때문이다. 마지막으로 ONNX Runtime 으론 ORT, ONNX-TF, nGraph 그리고 본 논문에서 새롭게 제안하는 C-ONNX 을 비교 대상으로 삼았다.

5. 실험 방법과 결과

MNIST 와 Mobilenet 의 반복 실행 시간을 측정하되, MNIST 는 1,000 회 실행 시간을 us 단위로 측정하였고, Mobilenet 은 10 회 실행 시간을 us 단위로 측정하였다. 샘플의 숫자는 각각 60 개이다.

먼저 Intel Core i7(이하 Core i7)에서 실험할 때 ONNX-TF 는 Python 라이브러리 기준으로 tensorflow 1.14, onnx-tf 1.5.0 그리고 onnx 1.7 을 사용하였고, ORT 는 ort 1.3, onnx 1.7 을 사용하였다. 그리고 nGraph 는 ngraph 0.26, ngraph-onnx 0.24, onnx 1.7 을 사용하였고, C-ONNX 은 0.1.90 을 사용하였다. ONNX-TF, ORT, nGraph 모두 실행 시 CPU 만 사용하도록 설정하였고, C-ONNX 은 GPU 가속을 지원하지 않기 때문에 별도의 설정을 하지 않았다.

<표 1> Core i7 에서 ONNX Runtime 간 MNIST 성능 비교

	ONNX-TF	ORT	nGraph	C-ONNX
평균(us)	16,588,980	24,641	55,314	360,026
표준편차(us)	79,309	4,358	9,914	15,986

표 1 에서 볼 수 있듯이 ONNX-TF, ORT, nGraph 그리고 C-ONNX 은 각각 16,588ms, 24ms, 55ms 그리고 360ms 성능을 보였다. ONNX-TF 가 다른 Runtime 에 비해 성능 저하가 심하고, ORT 가 월등한 성능을 보이는 것이 특징이다.

<표 2> Core i7 에서 ONNX Runtime 간 Mobilenet 성능 비교

	ONNX-TF	ORT	nGraph	C-ONNX
평균(us)	132,846,537	28,894	72,450	2,295,651
표준편차(us)	2,412,556	4,049	3,804	9,398

표 2 는 동일한 환경에서 Mobilenet 을 실행하였을 경우이다. Mobilenet 은 10 회만 실행하였음에도 불구하고 ONNX-TF 와 C-ONNX 의 실행 속도가 크게 늘어나는 것을 볼 수 있다. 그 이유는 Mobilenet 의 대부분을 차지하는 활성함수가 2D Convolution 인데, ORT 와 nGraph 에 비해 ONNX-TF 와 C-ONNX 의 2D Convolution 함수가 상대적으로 최적화가 덜 되었기 때문이다. 실행 시간 평균을 보면 C-ONNX 에 비해 ORT 와 nGraph 가 상대적으로 높은 성능을 보이고, ONNX-TF 는 굉장히 낮은 성능을 보이는 것을 확인할 수 있다.

다음으로 임베디드 디바이스에 해당하는 Raspberry

Pi 3(이하 RPi 3)에서 마찬가지로 MNIST 와 Mobilenet 을 수행하는 실험을 진행하였다. Raspberry Pi 에서 실험을 진행할 때는 ONNX-TF 는 tensorflow 1.13.1, onnx-tf 1.5.0, onnx 1.7 을 사용하였고, C-ONNX 은 동일한 버전인 0.1.90 을 사용하였다. ORT 와 nGraph 는 Raspberry Pi 에서 설치가 불가능하여 실험 대상에서 제외되었다.

<표 3> RPi3 에서 ONNX Runtime 간 MNIST 성능 비교

	ONNX-TF	ORT	nGraph	C-ONNX
평균(us)	353,849,864	N/A	N/A	4,666,383
표준편차(us)	1,296,736	N/A	N/A	56,877

RPi3 서 실험을 진행했을 때 표 3 에서 보듯이 ONNX-TF 와 C-ONNX 은 Core i7 에서 각각 21.3 배, 12.9 배 느리게 동작하는 것을 확인할 수 있다. ONNX-TF 와 C-ONNX 간의 성능 차이는 Core i7 에서 46.0 배에서 RPi3 에서 75.8 배로 더욱 벌어진 것을 알 수 있다.

<표 4> RPi3 에서 ONNX Runtime 간 Mobilenet 성능 비교

	ONNX-TF	ORT	nGraph	C-ONNX
평균(us)	704,681,999	N/A	N/A	81,281,101
표준편차(us)	3,493,544	N/A	N/A	263,503

표 4 는 동일한 환경에서 Mobilenet 의 성능을 비교한 것이다. ONNX-TF 는 Core i7 에 비해 5.3 배, CONNX 은 Core i7 에 비해 35.4 배 성능이 느려진 것을 확인할 수 있다. 하지만 여전히 C-ONNX 이 ONNX-TF 에 비해선 성능이 좋은 것을 알 수 있다.

C-ONNX 이 Core i7 에 비해 성능 격차가 보다 크게 난 이유는 2D Convolution 연산 수행 시 C-ONNX 은 Multi-threading 에 크게 의존하는데 Core i7 이 16 개의 Thread 를 사용할 수 있었던데 반해 RPi 3 에선 Thread 를 4 개만 사용할 수 있었던 것이 성능 저하의 폭이 커진 원인이다.

6. 결론

본 논문에선 Intel Core i7, Raspberry Pi 3 에서 ONNX 모델 중 MNIST 와 Mobilenet 의 실행 시간을 측정하는 실험을 진행하였다. 비교 대상은 Tensorflow 를 backend 로 사용하는 ONNX-TF 와, 처음부터 ONNX Runtime 으로 설계된 ORT, ONNX API 를 기본으로 지원하는 nGraph 그리고 본 논문에서 새롭게 제안하는 임베디드 기기에 최적화된 ONNX Runtime 인 C-CONNX 을 비교하였다. Intel Core i7 에선 ORT 가 월등한 성능을 보이지만, C-ONNX 또한 ONNX-TF 에 비해 상대적으로 높은 성능을 보였고, Raspberry Pi 3 에선 우수한 성능을 보이는 ORT 와 nGraph 를 설치할 수 없었기 때문에 ONNX-TF 와 C-ONNX 외엔 실험을 할 수 없었다. Raspberry Pi 3 에선 ONNX-TF 에 비해 C-ONNX 이 월등한 성능을 보이는데, 이는 임베디드 기기에 최적화한 설계의 차이로 해석할 수 있고, C-ONNX 이라는 새로운 ONNX Runtime 의 가능성을 보여주는 사례라 할 수 있다.

ACKNOWLEDGEMENT

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 ICT 명품인재양성 사업(IITP-2020-2051-001), 2020 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.2019-0-00421, 인공지능대학원 지원(성균관대학교))과 2020 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2020R1A2C2008447).

참고문헌

[1] “Open Neural Network eXchange” [Online]. Available: https://onnx.ai/. [Accessed: 21-Sep-2020]

[2] “Neural Network Exchange Format” [Online]. Available: https://www.khronos.org/nnef. [Accessed: 21-Sep-2020]

[3] “ONNX Runtime” [Online]. Available: https://github.com/microsoft/onnxruntime. [Accessed: 21-Sep-2020]

[4] “Tensorflow Backend for ONNX” [Online]. Available: https://github.com/onnx/onnx-tensorflow. [Accessed: 21-Sep-2020]

[5] Scott Cyphers, et al, “Intel nGraph: An Intermediate Representation, Compiler, and Executor for Deep Learning”, arXiv preprint arXiv:1801.08058

[6] “C implementation of ONNX Runtime” [Online], Available: https://github.com/semihlab/connx. [Accessed: 21-Sep-2020]

[7] “ONNX Model Zoo” [Online], Available: https://github.com/onnx/models. [Accessed: 21-Sep-2020]