Haskell을 이용한 텐서플로우 프로그램의 성능 평가

김연어[O1], 류샤오[1], 변석우[2], 우균[1,3],
[1]부산대학교 전기전자컴퓨터공학과, [2]경성대학교 컴퓨터공학과, [3]LG전자 스마트 제어 센터
yeoneo@pusan.ac.kr, xiaoliu@pusan.ac.kr, swbyun@ks.ac.kr, woogyun@pusan.ac.kr

Measuring the Performance of the TensorFlow Program Using Haskell

Yeoneo Kim[O1], Xio Liu[1], Sugwoo Byun[2], Gyun Woo[1,3]
[1,3]Dep. of Electrical and Computer Engineering, Pusan National University,
[2]School of Computer Science & Engineering, Kyungsung University,
[3]Smart Control Center of LG Electronics

요 약

딥 러닝 기술은 최근 컴퓨터 공학 분야에서 가장 주목받고 있는 기술 중 하나이다. 이 때문에 다양한 프레임워크가 개발되고 있다. 그 중 구글에서 개발한 텐서플로우는 폭 넓은 사용자를 확보하고 있으며, 여러 프로그래밍 언어에서 사용할 수 있다. 다양한 프로그래밍 언어 중 Haskell은 강력한 타입 추론 시스템을 제공하고 있어 텐서플로우 프로그래밍에 적합한 언어로 판단된다. 이 논문에서는 Haskell을 이용한 텐서플로우 프로그래밍 방법에 대해 살펴보고자 한다. 이를 위해 Haskell 텐서플로우 패키지인 tensorflow-haskell의 구조를 분석하고 손글씨 숫자 인식 예제 프로그램을 살펴보고 이 예제 프로그램을 Python 기반의 프로그램과 성능 비교 실험을 수행한다. 실험 결과 성능이 25% 떨어지는 것을 확인하였으며, Haskell 프로그램의 리스트 문제의 일부 개선을 통해 성능이 12.13% 개선될 수 있음을 보였다.

1. 서 론

최근 컴퓨터 공학 분야에서 가장 주목받고 있는 기술은 인공지능의 한 분야인 머신 러닝(machine learning)으로, 2016년 대한민국에서는 사람과 구글의 인공지능인 알파고와의 대결 이후 더욱 주목받고 있다. 특히 알파고에서 사용된 것으로 알려진 딥 러닝(deep learning) 기술은 컴퓨터 공학을 넘어 다양한 분야에서 연구 및 응용이 이루어지고 있다. 대표적으로 전자 공학에서는 딥 러닝과 같은 기계 학습에서 사용하는 연산에 특화된 하드웨어 연구가 이루어지고 있으며, 비공학 분야에서도 전공 분야에 딥 러닝 기술을 접목하고 있다.

이러한 추세에 맞춰 다양한 기계 학습 프레임워크가 개발되었으며, 그중 구글의 텐서플로우(TensorFlow)가 가장 널리 사용되고 있다. 텐서플로우는 구글에서 개발하고 공개한 기계 학습 오픈 소스 라이브러리이다[1]. 텐서플로우는 C++를 이용하여 구현되어 있으며, 공식적인 인터페이스로 Python과 C++를 제공하고 있다. 특히 많은 텐서플로우 사용자들은 사용이 간편한 Python을 이용하고 있다. 이외에도 비공식으로 Java나 Go, Haskell과 같은 인터페이스도 제공되고 있다.

이 논문에서는 그 중 Haskell을 이용한 텐서플로우 프로그래밍 방법에 대해 분석하고자 한다. Haskell은 함수형 프로그래밍 언어로 지연 계산(lazy evaluation)을 사용하는 대표적인 언어이다. 특히 Haskell은 강력한 타입 시스템을 제공하고 있어 대규모 프로그래밍 시 많은 실행 시간의 오류를 줄일 수 있기에 텐서플로우 프로그래밍에 적합하다. 더욱이 텐서플로우는 그래프(graph)를 생성하고 run 명령어를 만나게 되면 그래프를 평가(evaluation)하는 방식이기 때문에 Haskell의 지연 계산(lazy evaluation)과도 잘 어울릴 것으로 판단된다.

이 논문은 다음과 같이 구성된다. 2장에서는 관련 연구로 Haskell의 특징과 텐서플로우의 계산 과정에 대해 살펴본다. 3장에서는 github[2]를 통해 제공되고 있는 Haskell 텐서플로우의 구조를 살펴본다. 이후 4장에서는 현재 텐서플로우의 인터페이스로 널리 사용되고 있는 Python과 Haskell의 성능을 비교하고 5장에서 Haskell 텐서플로우에 대해 토의하고자 한다. 그리고 마지막으로 6장에서 결론을 맺는다.

2. 관련연구

2.1 Haskell의 특징

Haskell은 지연 계산을 이용하는 함수형 프로그래밍 언어이다. Haskell은 순수성을 유지하는 함수와 순서와 부수 효과(side effect)를 허용하는 모나드(monad)를 제공하며, 두 기능을 이용하여 효과적으로 프로그래밍할 수 있는 언어이다. 그리고 이러한 두 기능 사이의 안전성을 유지해주는 것이 Haskell의 타입 추론 시스템이다.

Haskell의 타입 추론 시스템은 여타 다른 언어보다 강력하다고 알려져 있다. 타입 추론 시스템이 강력하다는 것은 프로그램 실행 도중 발생할 수 있는 오류를 컴파일 시간에 검출할 수 있다는 것이다. 이러한 Haskell의 타입 추론 시스템은 순수한 함수와 모나드를 안전하게 연결하는 역할을 맡고 있다. 이외에도 강력한 타입 추론 시스템을 통해 프로그램 실행 오류를 많이 줄일 수 있으므로 규모가 큰 프로그램이나 안전성이 요구되는 프로그램에 적합하다. 즉 규모가 크며 한번 실행 시 많은 시간이 소요되며 정확성이 요구되는 딥 러닝 프로그램에 적합하다.

2.2 텐서플로우의 계산 과정

텐서플로우 계산 과정은 그래프를 계산하는 과정으로 볼 수 있다. 그래프 내부의 모든 데이터는 텐서(tensor)로 표현되며 계산은 즉시 이루어지지 않는다. 텐서플로우에서 계산은 세션(session)을 통해서만 이루어지며 세션의 run 명령어를 만나게

되면 그래프를 평가하는 방식이다. 그리고 이 과정을 도식화하면 그림 1과 같다.

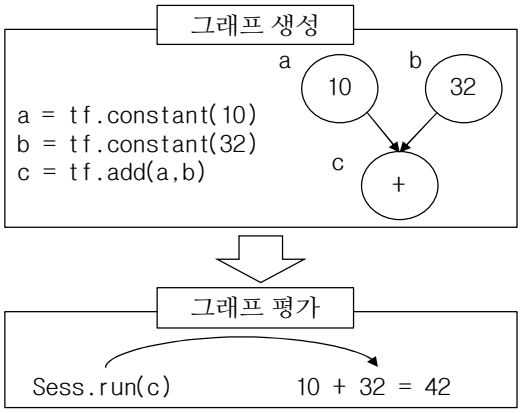

그림 1. 두 텐서를 생성하고 더하는 프로그램의 예

그림 1은 두 텐서를 생성하여 더하는 간단한 프로그램이다. 텐서플로우에서는 평가 전에 실제 계산은 이루어지지 않고 그래프만을 생성한다. 그리고 세션 단계에서 run을 통해 평가되어 실제 계산식으로 바뀌게 된다. 이는 딥 러닝과 같이 많은 연산이 이루어지는 환경에서 그래프 가지치기(pruning)와 같은 기법을 이용하여 불필요한 연산을 줄이기 위해서이다.

이러한 텐서플로우의 계산 과정은 Haskell과 유사하다고 볼 수 있다. Haskell도 모든 수식이 트리와 같은 형태로 표현되어 실제 그 값이 필요한 순간에만 계산이 이루어지는 방식이다. 즉 이러한 Haskell의 계산 특징과 텐서플로우의 계산 특징이 코드에서 효과적으로 연결된다면 큰 상승효과가 나타날 것으로 판단된다.

3. tensorflow-haskell 구조 분석

이 장에서는 Haskell에서 텐서플로우를 사용하기 위한 패키지인 tensorflow-haskell[2]의 특징 및 구조를 예제를 통해 살펴보고자 한다. tensorflow-haskell은 Haskell에서 텐서플로우를 사용하기 위해 제공되는 패키지이다. 이 패키지는 현재 구글에서 공식적으로 제공하는 인터페이스가 아니지만 텐서플로우 github에 같이 연결되어 있다. 2018년 5월을 기준으로 텐서플로우 1.7까지 지원하고 있으며, 설치 방법 및 사용 방법으로는 도커(docker)를 이용한 방법을 제공하고 있다. 그리고 분석을 위해 사용될 예제는 tensorflow-haskell 배포 시 같이 배포되고 있는 손글씨 숫자 인식(mnist: modified national institute of standards and technology)을 이용하고자 한다.

우선 tensorflow-haskell의 전반적인 구조를 살펴보면 텐서플로우와 연결은 FFI(foreign function interface)[3]를 통해 이루어진다. FFI은 Haskell 이외의 다른 언어로 작성된 프로그램과 연결을 위해 Haskell에서 제공하고 있는 방법이다. Haskell의 FFI를 통하면 텐서플로우 라이브러리의 경우 C++로 작성되어 있어서 Haskell 내부에서 바로 호출하여 사용할 수 있다. 그리고 텐서플로우에서 Python을 통해 제공하고 있던 다양한 함수를 Haskell에서 사용할 수 있도록 제공하고 있다.

mnist 예제를 살펴보면 teonsorflow-haskell에서는 프로그래밍을 위해 두 개의 모나드를 제공하고 있다. 첫 번째 모나드는 Build 모나드이다. 이 모나드는 텐서플로우 그래프에서 노드를 생성할 때 수행해야 하는 행동(action)을 정의하는 것이다. mnist의 예제를 살펴보면 코드 1과 같다.

```
 1:  data Model = Model {
 2:        train :: ...
 3:      , infer :: ...
 4:      , errorRate :: ...
 5:      }
 6:
 7:  createModel :: TF.Build Model
 8:  createModel = do
 9:      ...
10:      {- code for layers & training action -}
11:      return Mode {
12:        ...
13:      }
```

코드 1. tensorflow-haskell mnist의 평가 모델 정의

코드 1은 mnist 예제에서 사용할 평가 모델을 정의하는 것이다. tensorflow-haskell에서는 2단계의 레이어를 이용하여 학습하는 방법을 선택하고 있으며, softmax 학습 모델을 이용하고 있 다. 이를 위해 코드 1에서 Model 데이터 타입을 정의하며, 이 타입은 학습 함수와 추론 함수, 정밀도를 측정하는 함수를 가지고 있다. 그리고 createModel 함수에서 Build 모나드를 통해 Model에서 사용될 행동을 정의하고 있다.

그리고 tensorflow-haskell에서 제공하고 있는 두 번째 모나드는 Session 모나드이다. 이 모나드는 텐서플로우 세션을 실행하기 위한 모나드로 프로그램의 전반적인 처리가 실행되는 부분이다. 그리고 이 모나드는 내부적으로 이제까지 실행된 상태를 저장해야 하므로 Reader 모나드[4]를 포함하고 있다. mnist 예제에서 Session 모나드가 사용되는 예는 코드 2와 같다.

```
 1:  main :: IO ()
 2:  main = TF.runSession $ do
 3:      {- code for loading training data image & label -}
 4:      model <- TF.build createModel
 5:      {- code for define functions for gen. batches -}
 6:      forM_ ([0..1000] :: [Int]) $ \i -> do
 7:          ...
 8:          train model images labels
 9:      ...
10:      {- code for Test Error Rate -}
```

코드 2. Haskell mnist 프로그램 실행 및 세션 실행 코드

코드 2는 mnist 전체 프로그램 실행 및 세션을 실행한 예이다. main 함수에서는 바로 세션을 실행하는 함수인 runSession을 호출하며 해당 세션 내부에서 실행할 행동을 정의하고 있다. 코드 2의 세부 내용을 살펴보면 3~5번 라인에서 학습과 실험에 사용할 데이터를 불러오고 평가 모델을 생성 및 배치 작업에 사용할 함수를 정의한다. 그리고 6번 라인에서 1,000번의 반복 학습을 통해 모델의 정확성을 높이고 10번 라인에서 이를 테스트한다.

이처럼 tensorflow-haskell에서는 Build와 Session 모나드 타입을 통해 모델 생성과 세션 실행 영역을 분리하고 있다. 그리고 이를 일반 함수가 아닌 모나드를 통해 제공함으로써 기존 명령형 언어와 유사한 스타일의 프로그래밍을 제공하고 있다. 그리고 타입 추론을 통해 잘못된 타입의 데이터는 텐서플로우 라이브러리에 전달되지 못하기 때문에 프로그램의 안전성 또한 높다고 볼 수 있다.

4. 성능 비교

이 장에서는 3장에서 살펴본 tensorflow-haskell과 Python 인터페이스를 사용한 프로그램의 성능을 비교하고자 한다. 이를 위해 이 논문에서는 Haskell로 작성된 mnist와 같은 평가 모델을 이용하는 Python 프로그램을 작성하였다. 실험을 위해 사용된 환경으로 CPU Intel Q9550, RAM 8GB, OS Ubuntu 16.04, ghc 8.0.2, Python 3.5를 사용하였다. 그리고 실험 결과는 표 1과 같다.

표 1. Haskell과 Python으로 작성된 mnist 실행 결과

프로그램 명	실행 시간(초)	Python 대비비율
mnist_python	37.369	100.00%
mnist_haskell	49.820	75.01%
mnist_haskell_arr	44.430	84.11%

표 1은 Haskell과 Python으로 작성된 mnist 프로그램의 실행 결과이다. 실험 결과 실행 시간은 학습 시간을 포함하여 Python이 가장 빠른 것으로 나타났으며 정확도는 약 94%로 나타났다. 그리고 tensorflow-haskell 예제인 mnist_haskell의 경우 Python보다 약 12초가 느려 Python 대비 약 75% 성능을 보였다.

이 논문에서는 이 차이를 줄이기 위하여 Haskell의 리스트 성능 문제[5]를 일부 개선한 mnist_haskell_arr을 개발하였다. 이 프로그램은 mnist_haskell의 학습 데이터의 자료형을 배열로 변경한 것으로, 프로그램을 실행한 결과는 리스트 프로그램 대비 실행 시간이 약 5초가 줄어 12.13% 개선되었다. 이 프로그램은 tensorflow-haskell의 내부 구조는 수정하지 않고 응용 프로그램에서 학습 데이터 자료형만 변경한 것이다. 즉, tensorflow-haskell 내부에서 이용되고 있는 리스트까지 배열로 변경하면 이 시간은 더 단축될 것으로 판단되다.

5. 고 찰

이 장에서는 tensorflow-haskell을 분석한 결과에 대해 논의하고자 한다. tensorflow-haskell은 Haskell을 이용하여 텐서플로우 프로그램을 작성할 수 있게 하지만 한계점이 명확하다. 우선은 Haskell을 이용하지만 대부분 행렬 연산을 리스트로 사용하고 있다. Haskell에서 리스트의 원소에 접근 명령은 순차적으로 접근하기 때문에 리스트 크기에 의존적이다. 즉, 행렬의 크기가 크면 클수록 속도 저하가 발생한다. 이 때문에 Haskell의 mnist 초기 버전은 Python보다 느린 것으로 나타났다.

그리고 현재 실험에 사용한 mnist는 단순한 예제이기 때문에 Haskell의 타입 시스템을 이용 시 발생하는 이점을 보이기 어렵다. 또한, 데이터를 바로 텐서플로우로 전달하기 때문에 전단부에서 사용한 언어의 장점이 나타나지 않는다. 이 때문에 Haskell의 타입 시스템을 활용할 수 있는 프로그램은 텐서플로우로 데이터가 넘어가기 전 데이터 처리가 발생하는 경우이다. 이러한 경우 C++나 Python의 특성상 실행 도중 오류가 발생하기 쉽지만, Haskell은 컴파일 단계에서 많은 문제를 걸러낼 수 있다. 즉 Haskell이 텐서플로우에서 다른 언어보다 효과적임을 입증하기 위해서는 이러한 응용 분야를 찾는 일이 필요하다.

6. 결 론

이 논문에서는 Haskell에서 텐서플로우 프레임워크를 사용하기 위한 패키지인 tensorflow-haskell 패키지에 대해 살펴보았다. Haskell을 이용하여 텐서플로우 프로그래밍을 진행한다면 타입 추론 시스템의 도움을 받아 안전한 프로그래밍이 가능한 장점이 있다. 이 때문에 논문에서는 tensorflow-haskell의 기본적인 구조를 mnist 예제 프로그램을 통해 살펴보았다. 그리고 Python 버전의 mnist 프로그램과 성능을 비교해본 결과 tensorflow-haskell의 성능이 떨어지는 것을 확인하였으며, 일부 원인은 리스트에 있음을 확인하였다.

향후 연구로는 tensorflow-haskell 코어 내부를 좀 더 분석하여 리스트로 인한 성능 저하를 해결하고자 한다. 그리고 tensorflow-haskell 내부에서 병렬 프로그래밍 기능을 지원 가능 여부를 살펴보고자 한다. 이외에도 Haskell의 지연 계산 기능을 텐서플로우와 연계하는 방안에 대해 모색하고자 한다. 그리고 현재 예제로 사용한 mnist는 Haskell의 강력한 타입 시스템을 많이 활용하지 못하는 예제이다. 향후에는 Haskell의 타입 시스템의 이점을 보일 수 있는 예제를 활용하여 추가 실험을 진행하고자 한다.

ACKNOWLEDGMENT

이 논문은 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (2014-0-00035,매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대 OS 기초연구센터)).
*교신 저자 : 우균(부산대학교, woogyun@pusan.ac.kr).

참 고 문 헌

[1] M. Abadi, et. al, “TensorFlow: A System for Large-Scale Machine Learning,“ In Proceedings of 12th USENIX Symposium on Operating Systems Design and Implementation(OSDI 16), pp. 265-283, 2016.

[2] fkm3, Haskell bindings for TensorFlow, [Online]. Available: https://github.com/tensorflow/haskell.

[3] M. Chakravarty, et. al, The Haskell 98 Foreign Function Interface 1.0 An Addendum to the Haskell 98 Report, [Online]. Available: http://www.cse.unsw.edu.au/~chak/haskell/ffi.

[4] Andy Gill, mtl: Monad classes, using functional dependencies, [Online]. Available: http://hackage.haskell.org/package/mtl-2.2.2.

[5] HaskellWiki, Performance, [Online]. Available: https://wiki.haskell.org/Performance.