ISSN 2288-4866 (Print)
ISSN 2288-4882 (Online)
http://www.jiisonline.org
J Intell Inform Syst 2017 June: 23(2): 1~17 http://dx.doi.org/10.13088/jiis.2017.23.2.001

딥러닝 프레임워크의 비교: 티아노, 텐서플로, CNTK를 중심으로*

정여진
국민대학교 경영대학 경영학부
(*ychung@kookmin.ac.kr*)

안성만
국민대학교 경영대학 경영학부
(*sahn@kookmin.ac.kr*)

양지헌
국민대학교 일반대학원 데이터사이언스학과
(*jihunyang@hanmail.net*)

이재준
국민대학교 일반대학원 데이터사이언스학과
(*leejj4787@gmail.com*)

딥러닝 프레임워크의 대표적인 기능으로는 '자동미분'과 'GPU의 활용' 등을 들 수 있다. 본 논문은 파이썬의 라이브러리 형태로 사용 가능한 프레임워크 중에서 구글의 텐서플로와 마이크로소프트의 CNTK, 그리고 텐서플로의 원조라고 할 수 있는 티아노를 비교하였다. 본문에서는 자동미분의 개념과 GPU의 활용형태를 간단히 설명하고, 그 다음에 logistic regression을 실행하는 예를 통하여 각 프레임워크의 문법을 알아본 뒤에, 마지막으로 대표적인 딥러닝 응용인 CNN의 예제를 실행시켜보고 코딩의 편의성과 실행속도 등을 확인해 보았다. 그 결과, 편의성의 관점에서 보면 티아노가 가장 코딩 하기가 어렵고, CNTK와 텐서플로는 많은 부분이 비슷하게 추상화 되어 있어서 코딩이 비슷하지만 가중치와 편향을 직접 정의하느냐의 여부에서 차이를 보였다. 그리고 각 프레임워크의 실행속도에 대한 평가는 '큰 차이는 없다'는 것이다. 텐서플로는 티아노에 비하여 속도가 느리다는 평가가 있어왔는데, 본 연구의 실험에 의하면, 비록 CNN 모형에 국한되었지만, 텐서플로가 아주 조금이지만 빠른 것으로 나타났다. CNTK의 경우에도, 비록 실험환경이 달랐지만, 실험환경의 차이에 의한 속도의 차이의 편차범위 이내에 있는 것으로 판단이 되었다. 본 연구에서는 세 종류의 딥러닝 프레임워크만을 살펴보았는데, 위키피디아에 따르면 딥러닝 프레임워크의 종류는 12가지가 있으며, 각 프레임워크의 특징을 15가지 속성으로 구분하여 차이를 특정하고 있다. 그 많은 속성 중에서 사용자의 입장에서 볼 때 중요한 속성은 어떤 언어(파이썬, C++, Java, 등)로 사용가능한지, 어떤 딥러닝 모형에 대한 라이브러리가 잘 구현되어 있는지 등일 것이다. 그리고 사용자가 대규모의 딥러닝 모형을 구축한다면, 다중 GPU 혹은 다중 서버를 지원하는지의 여부도 중요할 것이다. 또한 딥러닝 모형을 처음 학습하는 경우에는 사용설명서가 많은지 예제 프로그램이 많은지 여부도 중요한 기준이 될 것이다.

주제어 : 딥러닝 프레임워크, 자동미분, 티아노, 텐서플로, Cognitive toolkit, CNN

논문접수일 : 2017년 2월 27일 논문수정일 : 2017년 2월 27일 게재확정일 : 2017년 3월 13일
원고유형 : 일반논문 교신저자 : 안성만

* 이 연구는 2017년 국민대 교내연구비지원으로 수행되었음

1. 서론

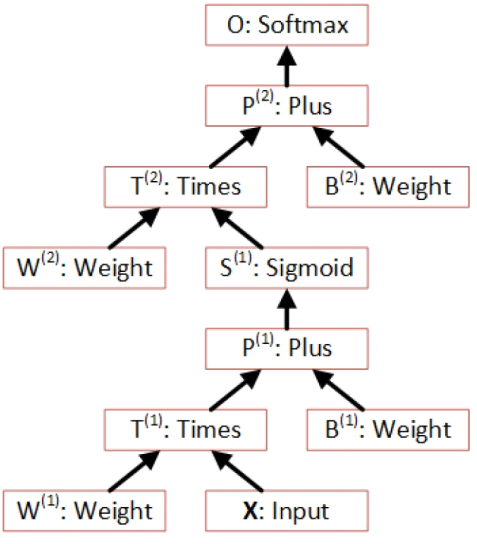
딥러닝 프레임워크는 딥러닝 모형을 쉽고 빠르게 개발할 수 있도록 도움을 주는 소프트웨어로서 대표적인 기능으로 '자동미분'과 'GPU의 활용' 등을 들 수 있다. 대학에서 개발되어 사용되어 왔던 딥러닝 프레임워크로서는 버클리 대학의 '카페(Caffe)', 몬트리올 대학의 '티아노(Theano)'이다. 그리고 연전에 구글이 '텐서플로(TensorFlow)'를 오픈소스로 공개함에 따라 저변을 확대해 오고 있다. 한편, 최근에는 마이크로소프트의 딥러닝 프레임워크인 Microsoft Cognitive Toolkit(CNTK) 2.0이 공개되었는데, C++와 파이썬 등의 프로그래밍 언어와 호환성을 제공하게 되면서 스크립트언어로만 사용하는 제약을 벗어나서 텐서플로나 티아노 같은 딥러닝 프레임워크와 어깨를 나란히 하게 되었다. 이를 보면 기존의 딥러닝 프레임워크는 연구용으로 주로 대학에서 개발되어 저변을 확대해 오다가, 구글과 마이크로소프트가 가세함에 따라 본격적인 주도권 경쟁이 시작된 것으로 보인다. 그러한 추세를 볼 때 구글과 마이크로소프트는 인공지능 분야의 주도권을 가져오기 위해서 딥러닝 프레임워크에 앞으로도 지속적인 투자를 할 것으로 예상이 된다.

이러한 시점에서 본 논문은 파이썬의 라이브러리 형태로 사용 가능한 프레임워크 중에서 구글의 텐서플로와 마이크로소프트의 CNTK, 그리고 텐서플로의 원조라고 할 수 있는 티아노를 비교해 보려고 한다. 그를 위하여 다음 절에서는 각 프레임워크의 특징과 문법을 간단히 설명하고, 그 다음에 대표적인 딥러닝 응용인 CNN의 예제를 실행시켜보고 코딩의 편의성과 실행속도 등을 확인해 보겠다.

2. 딥러닝 프레임워크의 특징

각 프레임워크의 공통되고도 가장 중요한 기능은 자동미분(automatic differentiation)을 하는 기능이다. 이 기능을 구현하기 위하여 연산그래프(computational graph)를 구축하는데, 연산그래프의 특징을 간략히 설명하면 다음과 같다.

CNN, RNN 등과 같은 주요 딥러닝모형은 네트워크의 출력을 계산하기 위하여 일련의 순차적인 연산과정이 필요하다. 이러한 순차적인 연산과정은 알고리듬의 형태로 표현할 수 있으며, 이러한 알고리듬은 방향그래프(directed graph)의 형태인 연산그래프로 표현이 가능하다. 예를 들어 1개의 내부계층을 가지는 신경망은 <Figure 1>과 같이 표현이 된다. 그림에서 각 노드는 연산자를 나타내며 연산자로 연결되는 자식노드는 피연산자를 나타낸다. 자식노드를 가지지 않는 끝노드는 입력값이나 모형변수를 나타낸다.



〈Figure 1〉 A computational graph for a neural network (Yu et al., 2014)

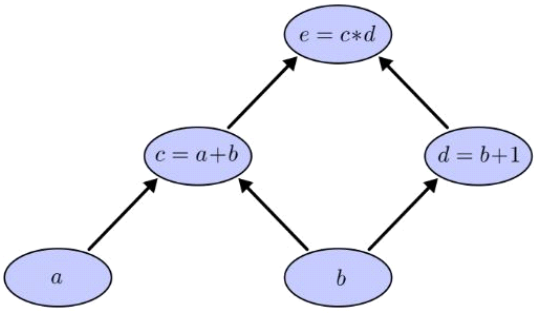
이러한 형태의 연산그래프를 사용하면 CNN이나 RNN과 같은 좀더 복잡한 모형도 표현할 수 있다.

딥러닝 모형을 학습하려면 오류함수의 정의가 필요하고, 경우에 따라서 추가로 조절항(regularization term)도 필요한데 이와 같은 연산도 연산그래프에 표현이 가능하다. 오류함수의 경우는 연산그래프의 최상위 노드로 표현이 되며, 모형의 학습을 위해서 편도함수를 최상위노드에서 역방향으로 계산하게 된다. 자동미분은 최상위노드에서 계산된 오류함수의 편도함수를 자식노드의 피연산자에 대하여 계산하고 그 결과를 다시 자식노드로 전달하면서 재귀적 방법으로 끝노드에 있는 모형변수에 대한 편도함수를 계산함으로써 가능하게 된다.

연산 그래프를 통한 자동미분을 하는 예를 들기 위하여 다음의 표현을 보자

$$e = (a + b) \times (b + 1) \qquad (1)$$

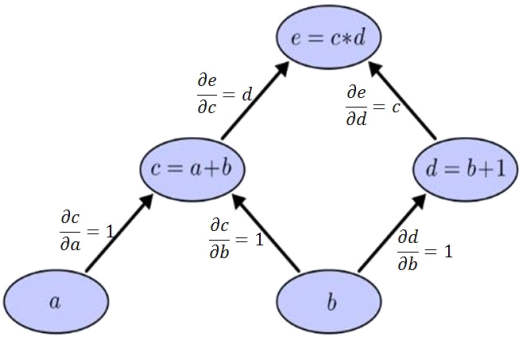
위의 표현에 있는 부분연산에 대하여 $c = a + b$, $d = b + 1$, $e = c \times d$라고 하면, 연산그래프는 <Figure 2>와 같이 작성할 수 있다.



〈Figure 2〉 A computational graph for $e = (a + b) \times (b + 1)$ (H13)

식(1)에서 a와 b의 값이 주어지면 e의 값을 계산할 수 있듯이, <Figure 2>의 연산그래프에서 입력변수(a와 b)의 값을 지정하면 그래프의 화살표를 따라 올라가면서 각 노드에서의 값을 계산할 수 있다. 예를 들어, $a = 2$, $b = 1$이라는 값을 주면 c = 3, d = 2, e = 6으로 각 노드의 값이 계산된다.

연산그래프를 통하여 편도함수의 값을 계산할 때는 먼저 직접 연결되어 있는 노드에 대한 편도함수를 구한다. 즉, 모든 연결선에 대하여 편도함수를 표시한 것이 <Figure 3>이다.



〈Figure 3〉 A computational graph with partial derivatives

<Figure 3>에서는 직접 연결되어 있는 노드에 대한 편도함수만 있다. 직접 연결되어 있지 않은 변수에 대한 편도함수는 연쇄법칙을 이용하여 계산할 수 있다. 예를 들어, e의 a에 대한 편도함수는 연쇄법칙에 의하여 다음과 같다.

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} = d \cdot 1 = b + 1 \qquad (2)$$

그런데 b의 경우는 두 갈래의 경로로 e에 영향을 미친다. 이런 경우는 각 경로에 대한 편도함수를 계산한 뒤에 각 결과를 합하여 편도함수를 구한다.

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b} = d \cdot 1 + c \cdot 1 = (a + b) + (b + 1) = a + 2b + 1 \quad (3)$$

예를 들어, $a = 2$, $b = 1$이라면 $\frac{\partial e}{\partial a} = 2$이고 $\frac{\partial e}{\partial b} = 5$이다. 자동미분에 대한 자세한 설명은 Goodfellow et al.,(2016)과 Yu et al., (2014)에 있다.

한편, 딥러닝 프레임워크의 중요한 기능의 하나인 GPU의 활용은 모든 프레임워크에서 가능하다. 일반적으로 프레임워크를 설치하는 과정에서 GPU버전을 따로 설치하거나(텐서플로, CNTK), 혹은 설치 후에 환경변수를 바꿔주는 방법(티아노)으로 GPU를 사용할 수 있다. 그런데 프레임워크에 따라 차이가 나는 기능은 여러 개의 GPU를 사용하거나 혹은 여러 대의 컴퓨터를 수평적으로 연결하여 실행하는 기능이다. 그런 점에서 티아노는 한 개의 GPU만을 활용하는 수준으로 구현되었으나 현재는 한 컴퓨터에 여러 개의 GPU를 사용할 수 있는 단계로 확장되고 있다(H2). 텐서플로의 경우에는 한 컴퓨터에 여러 개의 GPU를 활용할 수 있도록 설계되어 있으며, CIFAR-10데이터의 경우에 예제 코드도 제공하고 있다(H3). 한편 마이크로소프트에 따르면 CNTK는 다중 서버환경에서 실행 가능한 최초의 프레임워크이고 속도도 타 프레임워크에 비하여 빠르다는 설명이다(H4).

2.1 티아노(Theano)

티아노는 Python과 NumPy의 문법을 이용하여 표현된 딥러닝 라이브러리로서 몬트리올대학에서 개발되어 2008년부터 사용되어 왔다. 티아노는 실행될 때 자동으로 C++ 혹은 CUDA코드로 변환된 뒤에 Python모듈로 로딩된다. Bergstra et al.,(2010)에 소개된 티아노의 예제 프로그램은 <Figure 4>와 같다.

<Figure 4>의 코드는 로지스틱 회귀분석을 하는 예제로서 8~11번 라인에서 네 개의 심볼변수(symbolic variable)를 선언하는데, *x*와 *y*는 데이터를 저장하기 위한 목적이며 *w*와 *b*는 모형변수를 위한 것이다. *w*와 *b*는 티아노의 shared variable로 선언되어 있는데 shared variable은 다른 심볼변수와 달리 값이 초기화가 되고 전역변수(global variable)처럼 값이 유지가 되며 GPU가 있는 경우에는 GPU에 값이 저장되어 사용된다.

12번 라인은 로지스틱 함수의 코드이며 13~14번 라인은 교차엔트로피와 조절항을 사용하여 오류함수를 정의한 것이다. 12~14번 라인을 통하여 연산그래프가 구축되며, 15번 라인에 있는 grad함수가 *w*와 *b*에 대한 편도함수를 계산해서 반환하게 된다.

16번과 18번 라인에 있는 function함수는 심볼변수 *x*와 *y*의 값을 사용하여 출력값을 반환하는 함수를 각각 정의하고 있다. 티아노의 function함수는 실제 데이터와 모형을 연결해 주는 기능을 한다. function함수의 다른 기능은 shared변수의 값을 새로운 값으로 개선해 나갈 수 있게 하는 것이데 이는 18번 라인에 있는 updates인자를 통하여 이루어 진다.

티아노가 일반 프로그래밍 언어와 다른 점은 18번 라인까지 실제 연산이 실행되지 않는다는 것이다. 위의 코드에서 보듯이 실제 데이터는 19번 라인에서 생성이 되며 23번 라인에서 비로소 데이터가 function함수를 통하여 연산그래프에 전달이 되어 연산이 실행된다.

```
 1: import numpy as np
 2: import theano.tensor as T
 3: from theano import shared, function

 4: batch_size = 25
 5: n_samples = 5000
 6: input_dim = 2
 7: output_dim = 2

 8: x = T.fmatrix()
 9: y = T.fvector()
10: w = shared(np.random.randn(input_dim,output_dim-1))
11: b = shared(np.zeros([output_dim-1]))

12: p_1 = 1 / (1+T.exp(-T.dot(x, w) - b))
13: xent = -y*T.log(p_1[:,0]) - (1-y)*T.log(1-p_1[:,0])
14: loss = xent.mean() + 0.01*(w**2).sum()
15: gw, gb = T.grad(loss, [w, b])
16: train = function(inputs=[x, y],
17:                  outputs=loss,
18:                  updates={w:w-0.1*gw, b:b-0.1*gb})

# Helper function to generate a random data sample
19: X_data, y_data = generate_random_data_sample(n_samples, input_dim, output_dim)

20: for i in range(n_samples):
21:     indices = np.random.choice(n_samples, batch_size)
22:     X_batch, y_batch = X_data[indices], y_data[indices,0]
23:     err = train(X_batch, y_batch)
```

〈Figure 4〉 Theano code for logistic regression

티아노는 RNN을 구현하기가 편리하다고 알려져 있다(H6). RNN을 구현하기 위하여 필요한 함수는 scan이다. 티아노의 설명서(H2)에는 scan을 '반복문을 구현하기 위한 함수'로 정의하고 있다. 그런데 일반 프로그래밍 언어에서 제공하는 반복문이라기 보다는 RNN을 구현하는데 주로 사용되도록 만들어져 있어서 사용법이 좀 복잡하다. <Figure 5>는 RNN에 사용된 코드의 일부분이다.

그림에서 scan함수는 7번 라인에서 실행되고 있다. 8~11번 라인에서 scan함수의 주요 인자 4개가 사용되고 있는데 각각의 인자에 대한 설명은 다음과 같다.

fn: scan함수가 반복적으로 실행하는 함수. 3번 라인에 정의된 forward_prop_step으로 지정됨

sequences: fn이 지정한 함수가 실행될 때 사용하는 데이터. 이 데이터가 행렬이면 첫 행부터 행의 개수만큼 반복된다. forward_prop_step의 첫 번째 인자인 x_t로 전달된다.

outputs_info: fn이 지정한 함수가 실행된 후

```
# U, V, W are shared variables
1: x = T.ivector('x')
2: y = T.ivector('y')
3: def forward_prop_step(x_t, s_t_prev, U, V, W):
4:     s_t = T.tanh(U[:,x_t] + W.dot(s_t_prev))
5:     o_t = T.nnet.softmax(V.dot(s_t))
6:     return [o_t[0], s_t]
7: [o,s], updates = theano.scan(
8:     fn=forward_prop_step,
9:     sequences=x,
10:    outputs_info=[None, dict(initial=T.zeros(self.hidden_dim))],
11:    non_sequences=[U, V, W])
12: forward_propagation = theano.function([x], o)
```

〈Figure 5〉 Theano code for scan function (H14)

에 반환되는 값을 나타내며 동시에 직전에 반환된 값이 fn에서 재사용되기도 한다. 초기값을 지정하는 목적으로도 사용함. forward_prop_step의 반환값은 [o_t[0], s_t] 이고 s_t는 두 번째 인자인 s_t_prev로 재사용된다. s_t_prev의 초기값은 T.zeros(self.hidden_dim)이다

non_sequences: fn으로 전달되는 인수 중에서 데이터로 사용되지 않는 변수

scan함수에 의해서 반환되는 변수에는 fn이 반복실행 되면서 반환하는 모든 값이 array로 저장되어 있다. 그래서 마지막에 반환된 값만이 필요한 경우에는 array의 마지막 원소를 지정하여 사용하면 된다. <Figure 5>의 예제에서는 12번 라인의 출력변수가 o이므로 o[-1]이 마지막 원소이다.

2.2 텐서플로(TensorFlow)

2011년 부터 구글 브레인 팀은 첫 머신러닝 시스템으로 DistBelief를 만들었다. 구글에 있는 50개가 넘는 팀과 모회사 알파벳에서 검색, 음성검색, 광고, 구글 포토, 구글 맵스, 스트리트뷰, 번역, 유투브 등과 같은 실제 서비스에 DistBelief의 딥러닝모형이 적용되었다. 텐서플로는 2015년에 오픈 소스로 공개된 구글 브레인 팀의 두 번째 머신 러닝 시스템이다.

텐서플로는 연산그래프를 이용하고 자동미분 기능이 있으며, 또 파이썬 라이브러리의 형태로 그 문법이 티아노를 대체하기 위하여 개발되었다는 말이 있을 정도로 티아노와 상당히 유사하다. 텐서플로가 티아노와 비교해서 장점은 분산 컴퓨팅이 가능하다는 점과 좀더 추상화된 함수를 제공하여 코딩이 빠르다는 점이다. <Figure 6>에 있는 예를 들어서 설명을 해보겠다.

<Figure 6>는 <Figure 4>와 같이 로지스틱회귀를 하는 예제이다. 7~10번 라인에서 심볼변수를 선언하고 있는데, 그 형태가 티아노와 상당히 유사하다. 우선 입력데이터를 위해서는 placeholder라는 이름을 사용하고 있는데, 티아노에서는 해당변수에 대한 데이터의 입력을 위해 function함수를 사용하는 반면에, 텐서플로는 21번 라인에서 보듯이 feed_dict라는 인자를 사용한다는 점이 다르다. 모형변수의 선언을 위해서 티아노의

```
 1: import numpy as np
 2: import tensorflow as tf

 3: batch_size = 25
 4: n_samples = 5000
 5: input_dim = 2
 6: output_dim = 2

# Define placeholders for input
 7: X = tf.placeholder(tf.float32, shape=(batch_size, input_dim))
 8: y = tf.placeholder(tf.float32, shape=(batch_size, output_dim))

# Define variables to be learned
 9: W = tf.Variable(tf.random_normal([input_dim,output_dim]))
10: b = tf.Variable(tf.zeros([output_dim]))
11: z = tf.matmul(X, W) + b
12: loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(z, y))
13: opt = tf.train.AdamOptimizer()
14: opt_operation = opt.minimize(loss)

# Define input data
15: X_data, y_data = generate_random_data_sample(n_samples, input_dim, output_dim)

16: with tf.Session() as sess:
17:     sess.run(tf.initialize_all_variables())
18:     for i in range(n_samples):
19:         indices = np.random.choice(n_samples, batch_size)
20:         X_batch, y_batch = X_data[indices], y_data[indices]
21:         _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

〈Figure 6〉 Tensorflow code for logistic regression

shared함수와 같은 기능을 하는 함수가 텐서플로에서는 Variable함수이다.

11~14번 라인에서 모형의 오류함수와 학습방법을 정의하고 있는데 이 부분이 티아노와 좀 다르다. 티아노의 경우에는 오류함수와 학습과정에 대하여 편도함수의 계산을 제외하고 세부적으로 코딩을 다 해주는 반면에 텐서플로는 더 많은 부분이 추상화되어 있다. 13~14번 라인에서 보듯이 AdamOptimizer와 minimize함수를 사용하여 오류함수를 최소화하고 있는데, 이렇게 함으로써 티아노보다 모형의 구축을 좀더 빨리 할 수 있을 것으로 기대된다.

16번라인에서 볼 수 있는 텐서플로의 또 한가지 특징은 데이터를 입력하여 결과를 계산하기 위하여 Session함수를 사용해야 한다는 것이다. 모형은 연산그래프를 통하여 구축하고 실행은 그와 독립된 세션을 통하여 실행한다는 의미로 이해를 하면 되겠다.

2.3 MS Cognitive toolkit

마이크로소프트는 2015년 4월에 CNTK (Computational Network Toolkit)라는 이름으로 딥러닝 프레임워크를 오픈소스로 공개하였다. 당시에 CNTK는 BrainScript라는 이름의 스크립트언어로만 사용이 가능했으나 2016년 말에 파이썬 API를 제공하면서 이름도 Cognitive toolkit

으로 바꾸어 사용하고 있다. 파이썬 라이브러리로서의 Cognitive toolkit은 텐서플로와 그 기능이 유사하다고 할 수 있으나, 2016년말 현재 다른 프레임워크와의 차별점은 여러 서버를 통한 분산컴퓨팅이 가능하다는 것이다(H4).

파이썬을 사용한 Cognitive toolkit의 특징은 <Figure 7>을 통해서 보도록 하자. 앞의 예제와 마찬가지로 <Figure 7>은 로지스틱회귀를 하는 코드이다. 코드의 전체적인 구성은 텐서플로와 유사하다. 우선 7~10번 라인에서 심볼변수를 선언하고 있는데, 입력데이터를 위해서는 input_variable이라는 함수를 사용하고 있으며, 모형변수의 선언을 위해서 parameter함수를 사용한다. 11~13번 라인에서 오류함수를 정의하고 있으며 이 값들은 17번 라인에 있는 Trainer객체의 인자로 사용된다. 모형의 학습을 위해서 사용되는 Trainer객체는 다음과 같이 정의되어 있다(H5).

class Trainer(model, loss_function, eval_function, parameter_learners)

```
 1: import numpy as np
 2: import sys
 3: import os
 4: import cntk

 5: input_dim = 2
 6: output_dim = 2

# Define the network
 7: X = cntk.ops.input_variable(input_dim, np.float32)
 8: Y = cntk.ops.input_variable(output_dim, np.float32)
 9: w = cntk.ops.parameter(shape=(input_dim, output_dim))
10: b = cntk.ops.parameter(shape=(output_dim))

11: z = cntk.ops.times(X, w) + b
12: loss = cntk.ops.cross_entropy_with_softmax(z, Y)
13: eval_error = cntk.ops.classification_error(z, Y)

# Instantiate the trainer object to drive the model training
14: learning_rate = 0.1
15: lr_schedule = cntk.learning_rate_schedule(learning_rate, cntk.UnitType.minibatch)
16: learner = cntk.learner.sgd(z.parameters, lr_schedule)
17: trainer = cntk.Trainer(z, loss, eval_error, [learner])

# Initialize the parameters for the trainer
18: minibatch_size = 25
19: n_samples = 10000   # originally 20000
20: num_minibatches_to_train = int(n_samples / minibatch_size)
21: X_data, y_data = generate_random_data_sample(n_samples, input_dim, output_dim)

22: for i in range(0, num_minibatches_to_train):
23:     indices = np.random.choice(n_samples, minibatch_size)
24:     X_batch, y_batch = X_data[indices], y_data[indices]
25:     trainer.train_minibatch({X : X_batch, Y : y_batch})
```

〈Figure 7〉 CNTK code for logistic regression

Trainer의 인자 중에서 model, loss_function, 그리고 eval_function은 각각 cntk패키지의 함수이며, 네 번째 인자인 parameter_learners는 learner 패키지의 리스트를 사용하도록 되어있다. 16번 라인에 생성되어 있는 learner는 학습방법과 학습과정에서 필요한 모형변수를 지정하고 있는 객체이다. 17번 라인에서 생성된 Trainer객체는 25번 라인에서 train_minibatch라는 함수를 사용하여 실행된다.

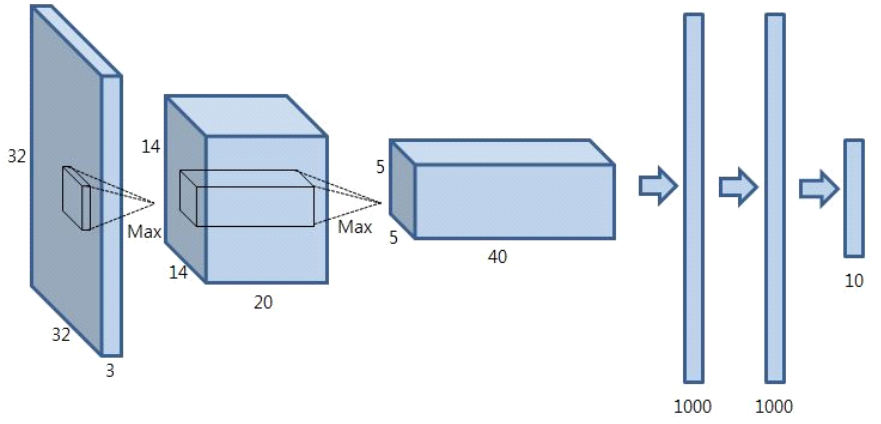
3. CNN모형의 구현

이 절에서는 앞에서 설명한 세가지 프레임워크를 CNN(convolutional neural networks, 합성곱 신경망)을 구현하는데 사용하여 비교해 보고자 한다. 실험을 위해 사용한 데이터는 CIFAR-10이라고 이름 지어진 것으로서 10개의 물체(비행기, 자동차 등)에 대하여 각 6000개씩 총 6만개의 32 × 32크기의 컬러이미지로 구성된 데이터이다(H7). 그 중에서 5만개는 학습용으로 사용하고 만개는 확인용으로 사용하도록 구분이 되어 제공된다. CIFAR-10데이터는 많은 연구자들이 다양한 모형을 적용하여 현재 확인데이터에 대한 인식률이 96%를 넘는 수준을 달성하고 있다(H8).

본 논문에서는 높은 인식률을 목표로 하는 것이 아니므로 <Figure 8>과 같은 비교적 단순한 형태의 CNN모형을 사용하였다. <Figure 8>에 있는 모형에는 두 개의 합성곱계층이 있고 max-pooling 을 한다. 그 다음에 천 개의 뉴런을 가지는 두 개의 내부계층이 연결되며, 마지막으로 softmax출력계층이 연결되는 구조이다. 합성곱의 필터규격은 5 × 5, 진행간격(stride)은 1, max-pooling은 2 × 2크기에 진행간격은 2로 지정하였다. 그리고 모든 계층의 활성화함수는 ReLU를 사용하였다.

원래 CIFAR-10데이터의 픽셀값은 0~255의 RGB값으로 되어 있는데, 그 값을 그대로 두고 ReLU를 사용하면 전혀 학습이 되지 못하는 문제가 발생한다. 그래서 그 값을 0.0~1.0구간의 값으로 변환하여 사용하였다.

티아노를 이용한 CNN코드의 일부분은 <Figure 9>와 같다. <Figure 9>에 있는 코드에서



〈Figure 8〉 CNN model for CIFAR-10

```
from network3 import *
mini_batch_size=50
net = Network([
        ConvPoolLayer(image_shape=(mini_batch_size, 3, 32, 32),
                      filter_shape=(20, 3, 5, 5),
                      poolsize=(2, 2),
                      activation_fn=ReLU),
        ConvPoolLayer(image_shape=(mini_batch_size, 20, 14, 14),
                      filter_shape=(40, 20, 5, 5),
                      poolsize=(2, 2),
                      activation_fn=ReLU),
        FullyConnectedLayer(
            n_in=40*5*5, n_out=1000, activation_fn=ReLU, p_dropout=0.5),
        FullyConnectedLayer(
            n_in=1000, n_out=1000, activation_fn=ReLU, p_dropout=0.5),
        SoftmaxLayer(n_in=1000, n_out=10)],
        mini_batch_size)

train_data, test_data=load_cifar10_shared('../cifar/cifar-10-batches-py/')
net.SGD(train_data, 200, mini_batch_size, 0.01, test_data, test_data)
```

〈Figure 9〉 Theano code snippet for CIFAR-10

생략된 앞부분은 데이터를 읽고 입력데이터를 shared variable로 지정(속도향상을 위하여 GPU로 복사함)하는 함수(load_cifar10_shared)이다.

<Figure 9>는 CNN모형을 정의하고 SGD를 실행하는 과정이다. 코드를 보면 합성곱계층(ConvPoolLayer) 2개와 일반 신경망계층(FullyConnectedLayer)이 두 개 그리고 출력계층(SoftmaxLayer) 한 개가 순차적으로 연결되어 있음을 쉽게 확인할 수 있다. <Figure 9>를 보면 티아노의 코딩이 간결해 보이지만 실제로 Network 클래스(가중치와 편향을 정의하고 학습하는 방법 등)를 직접 구현해야 한다는 번거로움이 있다. Network클래스는 (H1)에 있는 코드를 조금 수정하여 사용하였다. 실험은 intel core i7 CPU, CentOS 7, 그리고 NVIDIA 1080 GPU에 설치된 파이썬 2.7에 추가된 cuDNN 5.1과 티아노(Bleeding-edge)를 사용하였다. <Figure 9>의 마지막 라인에 있는 것과 같이 200 epoch를 실행한 결과 연산시간은 20분 30초 걸렸으며, 확인용 데이터의 최고 인식률은 74.15%로 측정되었다.

한편 텐서플로를 이용한 CNN코드의 일부분은 <Figure 10>과 같다. <Figure 10>에는 티아노의 경우와 마찬가지로 데이터를 읽는 과정은 생략하고 모형을 정의하고 학습을 실행하는 부분만 보여주고 있다. 그리고 두 번째 합성곱계층과 첫 번째 일반 신경망계층의 코드는 지면을 아끼기 위하여 생략되었다. 그럼에도 불구하고 <Figure 9>의 티아노 코드와 비교하면 좀 산만하고 길어 보인다. 그 이유는 티아노의 경우에는 가중치와 편향을 선언하는 부분이 Network클래스에 숨겨져 있는 반면에 텐서플로에서는 그 부분까지 다 포함되어있기 때문이다. 실제로 티아노의 Network클래스 구현부분은 파이썬 코드가 100줄이 넘으며(H1), 그것을 감안한다면 코드의

```
x = tf.placeholder(tf.float32, [None, 32, 32, 3])
y_ = tf.placeholder(tf.float32, [None, 10])
keep_prob = tf.placeholder("float")

W_Conv1 = weight_variable([5, 5, 3, 20])  # 1st convolutional layer
b_Conv1 = bias_variable([20])
Conv1 = tf.nn.relu(conv2d(x, W_Conv1) + b_Conv1)
Pool1 = max_pool_2x2(Conv1)

# 2nd convolutional layer .....

# 1st fully connected layer .....

W_FC2 = weight_variable([1000, 1000])  # 2nd fully connected layer
b_FC2 = bias_variable([1000])
FC2 = tf.nn.relu(tf.matmul(FC1_drop, W_FC2) + b_FC2)
FC2_drop = tf.nn.dropout(FC2, keep_prob)

W_FC3 = weight_variable([1000, 10])  # output layer
b_FC3 = bias_variable([10])
y_conv=tf.nn.softmax(tf.matmul(FC2_drop, W_FC3) + b_FC3)

cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.GradientDescentOptimizer(0.0001).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
sess = tf.Session()
sess.run(tf.initialize_all_variables())
minibatch_size = 50
for i in range(200000):
    j=(i*minibatch_size)%len(x_train)
    x_batch, y_batch = x_train[j:j+minibatch_size], y_train[j:j+minibatch_size]
    train_step.run(session=sess, feed_dict={x:x_batch, y_: y_batch, keep_prob: 0.5})
```

〈Figure 10〉 TensorFlow code snippet for CIFAR-10

양은 텐서플로가 훨씬 적으므로 코딩을 위해 걸리는 시간도 텐서플로가 적다고 할 수 있다.

실험은 티아노와 동일한 환경에 텐서플로(버전 0.11)를 설치하여 수행되었다. 티아노와 마찬가지로 200 epoch를 실행한 결과 연산시간은 20분 걸렸으며, 확인용 데이터의 인식률은 69.08%로 측정되었다. 티아노의 경우와 비교해서 인식률이 낮은 이유는 텐서플로의 경우는 최고 인식률이 아니라 학습을 마친 뒤에 측정된 값이기 때문이다. 그리고 학습률이 좀 다르게 적용되었다.

마지막으로 CNTK를 CNN코드의 일부분은 <Figure 11>과 같다. 앞에서와 마찬가지로 데이터를 읽는 과정은 생략하고 모형을 정의하고 학습을 실행하는 부분만 보여주고 있다. CNTK코드를 텐서플로와 비교할 때, 모형의 정의부분이 좀 간결해 보인다. <Figure 11>에서 CNN모형을 정의하는 부분을 함수(create_basic_model)로 분리해 놓았는데, 텐서플로와 달리 가중치와 편향을 명시하지 않아도 되어서 코드가 간결해 졌고, 그 때문에 코드를 보면 모형의 형태를 쉽게 파악

```
def create_basic_model(input, out_dims):
    net = Convolution((5,5), 20, init=glorot_uniform(), activation=relu, pad=False)(input)
    net = MaxPooling((2,2), strides=(2,2))(net)
    net = Convolution((5,5), 40, init=glorot_uniform(), activation=relu, pad=False)(net)
    net = MaxPooling((2,2), strides=(2,2))(net)
    net = Dense(1000, init=glorot_uniform())(net)
    net = Dropout(0.5)(net)
    net = Dense(1000, init=glorot_uniform())(net)
    net = Dropout(0.5)(net)
    net = Dense(out_dims, init=glorot_uniform(), activation=relu)(net)
    return net

input_var = input_variable((num_channels, image_height, image_width))
label_var = input_variable((num_classes))

z = create_basic_model(input_var, out_dims=10)
ce = cross_entropy_with_softmax(z, label_var)
pe = classification_error(z, label_var)

minibatch_size = 50
lr_per_minibatch = learning_rate_schedule(0.01, UnitType.minibatch)
learner     = sgd(z.parameters, lr = lr_per_minibatch)
trainer     = Trainer(z,ce,pe,[learner])

for i in range(200000):
    print (i)
    j=(i*minibatch_size)%len(x_train)
    x_batch, y_batch = x_train[j:j+minibatch_size], y_train[j:j+minibatch_size]
    trainer.train_minibatch({input_var : x_batch, label_var : y_batch})
```

〈Figure 11〉 CNTK code snippet for CIFAR-10

할 수 있다. 학습을 하는 부분은 1절에서의 예제와 비슷한 형태로 구현하였다.

논문 작성시점에 LINUX CentOS환경에서 설치하기 어려운 제약으로 인하여 CNTK코드의 실험은 윈도우 7 PC(Intel core i7-2637M @1.7GHz)에 설치된 CNTK V2.0 Beta 7 Release에서 수행되었다. GPU가 설치되지 않은 환경이기 때문에 200 epoch를 실행한 결과 연산시간은 15시간 30분정도 걸렸으며, 확인용 데이터의 인식률은 65.6%로 측정되었다. 속도에 있어서 LINUX+GPU환경에서 텐서플로의 경우와 비교하면 50배 가까이 차이가 나는 것이므로 꽤 많은 차이라 할 수 있다. (H9)에 따르면 NVidia Pascal Titan X + cuDNN 에서 실행된 CNN모형은 Xeon E5-2630 v3 CPU에서보다 49~74배 빠르고, 또 GTX1080은 Titan X보다 1.31~1.43배 느리므로, 단순계산으로 GTX1080+ cuDNN의 실행속도는 Xeon CPU보다 대략 34~56배 빠르다고 할 수 있다. 그리고 CNTK가 설치된 컴퓨터가 노트북PC이므로 속도의 차이는 더 느릴 것이라는 것을 감안한다면, 실험에서 측정된 50배 정도의 속도 차이는 CNTK에 의해서 기인한 것이라고 보기는 어렵다. 그리고 확인용 데이터의 인식률이 다른 실험과 비교하여 낮은 것은 가중치를 초기화하는 값이 다르기 때문으로 생각된다. 학습용 데이터의 인식률은 86.0%내외이다.

4. 요약

지금까지 딥러닝 프레임워크의 주요 기능을 살펴보고 또 세 종류의 프레임워크에 대하여 간단한 예제를 통하여 문법적인 특징을 살펴 보고 추가로 CIFAR-10데이터에 CNN모형을 적용한 실험을 통하여 그 결과를 확인해 보았다. CIFAR-10데이터에 대한 실험을 통하여 각 프레임워크에 대한 코딩의 편의성과 실행속도에 대한 요약을 하자면 다음과 같다.

우선 코딩의 편의성은 CNTK, 텐서플로, 티아노의 순서로 우수하다고 할 수 있다. 그 판단 기준은 각 프레임워크에서 제공하는 라이브러리의 직관성이나 학습의 용이성이 아닌 단순히 코딩의 길이에 따른 것임을 밝혀둔다. 그 기준에 따르면 티아노가 가장 코딩 하기가 어렵고, CNTK와 텐서플로는 많은 부분이 비슷하게 추상화 되어 있어서 코딩이 비슷하지만 가중치와 편향을 직접 정의하느냐의 여부에서 차이를 보였다. 한편 추상화된 라이브러리가 많다는 것은 동전의 양면과 같은 것이라서, 코딩의 편의성은 제공한다는 장점은 있지만 코딩의 유연성은 제공하지 못한다는 단점이 동시에 존재한다. 예를 들어, 기존의 모형에서 새로운 기능을 추가하기를 원하거나 혹은 새로운 학습방법을 시험해 보려고 하면 티아노와 같은 형태의 코딩이 필요하다. 또 티아노에서 추상화된 라이브러리를 직접 작성하여 사용할 수도 있는데, <Figure 9>에 있는 Network클래스가 그런 경우이다. 그와 같은 형태로 많이 사용되고 있는 Keras(H10)와 같은 라이브러리(wrapper라고도 부른다)도 있다.

각 프레임워크의 실행속도에 대한 평가는 '큰 차이는 없다'는 것이다. 실행속도에 대한 자료(H11) 에 의하면 텐서플로는 티아노에 비하여 속도가 느리다는 평가이다. 특히 Bahrampour et al.(2016)에서 0.6.0버전을 사용하여 한 실험도 그렇게 보고하였다. 그런데 본 연구의 실험에 의하면, 비록 CNN 모형에 국한되었지만, 텐서플로가 아주 조금이지만 빠른 것으로 나타났다. 본 연구에서 사용한 텐서플로(0.11버전)에서는 속도가 많이 향상된 것으로 보인다. 그리고 CNTK의 경우에도, 비록 실험환경이 달랐지만, 실험환경의 차이에 의한 속도의 차이의 편차범위 이내에 있는 것으로 판단이 되었다.

본 연구에서는 세 종류의 딥러닝 프레임워크를 살펴보았는데, 위키피디아(H12)에 따르면 딥러닝 프레임워크의 종류는 12가지가 있으며, 각 프레임워크의 특징을 15가지 속성으로 구분하여 차이를 특정하고 있다. 그 많은 속성 중에서 사용자의 입장에서 볼 때 중요한 속성은 어떤 언어(파이썬, C++, Java, 등)로 사용가능한지, 어떤 딥러닝 모형에 대한 라이브러리가 잘 구현되어 있는지 등일 것이다. 그리고 사용자가 대규모의 딥러닝 모형을 구축한다면, 다중 GPU 혹은 다중 서버를 지원하는지의 여부도 중요할 것이다. 또한 딥러닝 모형을 처음 학습하는 경우에는 사용설명서가 많은지 예제 프로그램이 많은지 여부도 중요한 기준이 될 것이다.

참고문헌(References)

Bahrampour, S., N. Ramakrishnan, L. Schott, and M. Shah, "Comparative Study of Deep Learning Software Frameworks," arXiv:1511.06435v3 (2016)

Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D.

Warde-Farley and Y. Bengio, “Theano: A CPU and GPU Math Expression Compiler,” *Proceedings of the Python for Scientific Computing Conference (SciPy)* 2010. June 30 - July 3, Austin, TX

Goodfellow, I., Y. Bengio, and A. Courville, *Deep Learning*, MIT Press. 2016

Yu, D., A. Eversole, M. Seltzer, K. Yao, O. Kuchaiev, Y. Zhang, F. Seide, Z. Huang, B. Guenter, H. Wang, J. Droppo, G. Zweig, C. Rossbach, J. Gao, A. Stolcke, J. Currey, M. Slaney, G. Chen, A. Agarwal, C. Basoglu, M. Padmilac, A. Kamenev, V. Ivanov, S. Cypher, H. Parthasarathi, B. Mitra, B. Peng, and X. Huang, “An Introduction to Computational Networks and the Computational Network Toolkit,” *Microsoft Research,* October 1, 2014

(H1) http://neuralnetworksanddeeplearning.com/

(H2) http://deeplearning.net/software/theano/index.html

(H3) https://www.tensorflow.org/

(H4) https://github.com/Microsoft/CNTK

(H5) https://www.cntk.ai/pythondocs/index.html

(H6) https://deeplearning4j.org/

(H7) https://www.cs.toronto.edu/~kriz/cifar.html

(H8) http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

(H9) https://github.com/jcjohnson/cnn-benchmarks

(H10) https://keras.io/

(H11) https://www.microway.com/hpc-tech-tips/deep-learning-frameworks-survey-tensorflow-torch-theano-caffe-neon-ibm-machine-learning-stack/

(H12) https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

(H13) https://colah.github.io/posts/2015-08-Backprop/

(H14) http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Abstract

Comparison of Deep Learning Frameworks: About Theano, Tensorflow, and Cognitive Toolkit

Yeojin Chung* · SungMahn Ahn** · Jiheon Yang*** · Jaejoon Lee****

The deep learning framework is software designed to help develop deep learning models. Some of its important functions include “automatic differentiation” and “utilization of GPU”. The list of popular deep learning framework includes Caffe (BVLC) and Theano (University of Montreal). And recently, Microsoft's deep learning framework, Microsoft Cognitive Toolkit, was released as open-source license, following Google’s Tensorflow a year earlier. The early deep learning frameworks have been developed mainly for research at universities. Beginning with the inception of Tensorflow, however, it seems that companies such as Microsoft and Facebook have started to join the competition of framework development. Given the trend, Google and other companies are expected to continue investing in the deep learning framework to bring forward the initiative in the artificial intelligence business. From this point of view, we think it is a good time to compare some of deep learning frameworks. So we compare three deep learning frameworks which can be used as a Python library. Those are Google's Tensorflow, Microsoft’s CNTK, and Theano which is sort of a predecessor of the preceding two.

The most common and important function of deep learning frameworks is the ability to perform automatic differentiation. Basically all the mathematical expressions of deep learning models can be represented as computational graphs, which consist of nodes and edges. Partial derivatives on each edge of a computational graph can then be obtained. With the partial derivatives, we can let software compute differentiation of any node with respect to any variable by utilizing chain rule of Calculus.

First of all, the convenience of coding is in the order of CNTK, Tensorflow, and Theano. The criterion is simply based on the lengths of the codes and the learning curve and the ease of coding are not the main concern. According to the criteria, Theano was the most difficult to implement with, and

* School of Business Administration, Kookmin University
** Corresponding Author: Ahn, SungMahn
College of Business Administration, Kookmin University
77 Jeongneung-ro, Seongbuk-gu, Seoul 02707, Korea
Tel: +82-2-910-4574, Fax: +82-2-910-4079, E-mail: sahn@kookmin.ac.kr
*** Department of Data Science, Kookmin University
**** Department of Data Science, Kookmin University

CNTK and Tensorflow were somewhat easier. With Tensorflow, we need to define weight variables and biases explicitly. The reason that CNTK and Tensorflow are easier to implement with is that those frameworks provide us with more abstraction than Theano. We, however, need to mention that low-level coding is not always bad. It gives us flexibility of coding. With the low-level coding such as in Theano, we can implement and test any new deep learning models or any new search methods that we can think of.

The assessment of the execution speed of each framework is that there is not meaningful difference. According to the experiment, execution speeds of Theano and Tensorflow are very similar, although the experiment was limited to a CNN model. In the case of CNTK, the experimental environment was not maintained as the same. The code written in CNTK has to be run in PC environment without GPU where codes execute as much as 50 times slower than with GPU. But we concluded that the difference of execution speed was within the range of variation caused by the different hardware setup.

In this study, we compared three types of deep learning framework: Theano, Tensorflow, and CNTK. According to Wikipedia, there are 12 available deep learning frameworks. And 15 different attributes differentiate each framework. Some of the important attributes would include interface language (Python, C ++, Java, etc.) and the availability of libraries on various deep learning models such as CNN, RNN, DBN, and etc. And if a user implements a large scale deep learning model, it will also be important to support multiple GPU or multiple servers. Also, if you are learning the deep learning model, it would also be important if there are enough examples and references.

Key Words : deep learning framework, Theano, TensorFlow, CNTK, computational graph, CIFAR-10

Received : February 27, 2017 Revised : February 27, 2017 Accepted : March 13, 2017
Publication Type : Regular Paper Corresponding Author : Ahn, SungMahn

저 자 소 개

정 여 진
연세대학교에서 경제학 및 응용통계학 복수전공으로 학사를 취득하였으며, 동 대학원에서 응용통계학 석사학위를 취득한 후 Pennsylvania State University에서 통계학 박사학위를 취득하였다. 현재 국민대학교 경영대학 경영학부 조교수로 재직중이다. 주요 연구분야는 비모수분포추정, 모형기반 군집분석, hierarchical linear model, 일반화선형모형, 데이터마이닝 등이다.

안 성 만
현재 국민대학교 경영대학 경영학부교수로 재직하고 있다. 서울대학교 경영대학에서 학사를, George Mason University에서 정보기술학 박사학위를 취득하였다. Johns Hopkins University의 응용수학과에서 박사후과정을 보냈고 ㈜쌍용정보통신에서 근무하였다. 주요 관심분야는 통계적 방법론, 모형선택, 데이터마이닝, 뉴럴네트워크 등이다.

양 지 헌
현재 국민대학교 일반대학원 데이터사이언스학과에서 데이터사이언스전공 박사과정에 재학 중이며,숭실대학교 물리학과에서 학사학위를 취득한 바 있다. 주요 관심 분야는 딥러닝(deeplearning)에 기반한 AI와 머신러닝이다. 전 VTW수석컨설턴트로 재직한 바 있으며 현재 글로벌텔레콤의 IOT분석팀장으로 일하고 있다.

이 재 준
현재 국민대학교 일반대학원 데이터사이언스학과에서 데이터사이언스전공 석사과정에 재학 중이며,국민대학교 수학과에서 학사학위를 취득한 바 있다. 주요 관심 분야는 딥러닝(deeplearning)에 기반한 빅데이터 분석, 감성 분석이다.