한국컴퓨터정보학회지
제23권 제2호, 2015. 12.

구글 TensorFlow 소개

• 김종영(한양대학교 컴퓨터공학과)

I. 서 론

최근 들어 글로벌 IT 기업들이 머신러닝(Machine Learning) 경쟁에 뛰어들고 있다. 이들은 주로 기존의 자사 서비스에 머신러닝 기술을 접합하고 있다. 페이스북은 2015년 6월 얼굴인식 서비스인 Moments를 발표했는데 이는 사진 속의 특정 인물을 대상으로 앨범을 구성하게 해 준다. Moments의 핵심기술은 얼굴 인식기술로서 98%의 정확률과 8억건의 사진을 5초안에 처리할 수 있는 성능을 보여준다. 마이크로소프트는 영상통화 서비스인 Skype에 통역 기능을 추가했는데, 모국어로 이야기하면 상대방의 언어로 실시간으로 전환되어 음성으로 전달되게 된다. 현재 영어-독일어, 영어-스페인어 서비스가 제공 중이며 사용하는 사람들의 숫자가 늘어나면서 그 정확도는 더 높아질 것으로 전망된다. 넷플릭스는 다양한 기기로 홈페이지에 접속하는 사용자에게 방대한 콘텐츠 중에서 어떤 콘텐츠를 제시할 것인가라는 문제를 머신러닝 기법을 활용하여 그 기준을 만들었다. 머신러닝이 에너지 사용 최적화에 사용된 사례도 있는데 구글은 뉴럴네트워크를 사용하여 99.6%의 전력 사용 효율을 예측했다. 뉴럴네트워크에 사용된 입력 데이터로는 부하(서버를 비롯한 장비 에너지 사용량, 펌프 스피드, 냉각기 정보), 날씨, 장비 수 등이 사용되었다.

특히 구글은 대부분의 자사 서비스에 머신러닝 기법을 적용하여 서비스 질 향상을 꾀하고 있다. 구글 포토, 구글 드라이브, 구글 번역, 구글 캘린더, 지메일 등 수십여 서비스에 머신러닝 기법이 도입되어 왔다. 구글은 이를 위해 머신러닝에 대한 인적, 물적 투자를 과감히 하고 있으며 구글 데이터 센터를 활용, 16,000개의 CPU와 10억 건 이상의 데이터 연결을 지원하는 뉴럴네트워크를 구축했다. 구글은 구축된 뉴럴네트워크를 사용하여 유투브에서 무작위로 추출한 천만건의 이미지에서 고양이 이미지를 스스로(unsupervised) 찾아내는데 성공했다.

구글이 고양이 이미지 추출에 사용한 기술은 딥러닝이었으며 이는 “Google Brain”이라는 프로젝트에서 진행되었다. “Google Brain”은 구글 연구진과 스탠포드 대학의 Andrew Ng 교수가 모여서 결성된 팀에서 2011년 시작된 프로젝트였다. 이들의 연구 목표는 대규모 딥러닝 소프트웨어 시스템을 만드는 것이었으며 그 결과로서 만들어진 것이 DistBelief였다. DistBelief는 대규모 분산 기계학습 시스템으로서 구글 클라우드 플랫폼에서 실행되는 클라우드 서비스이다. DistBelief의 단점으로는 구글 내부 인프라와 너무 단단히 연결되어 있다는 점, 뉴럴 네트워크만 지원한다는 점을 들 수 있다.

DistBelief가 구글의 1세대 기계학습 인프라였다면 2015년 11월 9일 오픈 소스 형태로 발표한 TensorFlow는 2세대 기계학습 인프라에 해당된다고 말할 수 있다.

II. 관련 분야

1. 자동 미분(Automatic Differentiation)

알고리즘에 의한 미분(Algorithmic Differnetiation)은 프로그램 코드에 의해 기술된 함수의 미분을 방법으로서 자동

미분(Automatic Differnetiation) 또는 계산 미분(Computational Differentiation)이라고도 한다. 알고리즘 미분은 미분 계산이 체인 룰(Chain Rule)을 활용하면 아무리 복잡한 계산식이라도 기본 연산(덧셈, 곱셈)의 순차적인 적용에 의해 이루어진다는 기본 개념에서 출발한다. 알고리즘 미분은 전방향(forward mode) 및 역방향 모드(reverse mode)로 이루어진다. 전방향 모드에서는 특정 독립 변수를 정한 뒤 미분 계산을 수행하는 반면 역방향 모드에서는 특정 종속 변수를 정한 뒤 미분 계산을 수행한다. 뉴럴네트워크 역전파(back-propagation) 알고리즘은 에러 함수의 미분값 계산을 출력 레이어에서 시작하여 입력 레이어로 이동하면서 계산하는 것이 핵심이다. 역전파 알고리즘은 역방향 모드를 활용한 하나의 예로서 n번째 레이어의 미분 값의 계산을 (n-1)번째 레이어의 미분 값을 사용하게 된다.

TensorFlow는 알고리즘 미분 또는 자동 미분 기능을 내장하고 있다. 따라서 사용자가 정의한 예측 모델에 대해 TensorFlow가 gradient descent 방법을 수행할 수 있음을 의미한다. 사용자는 데이터 및 목적 함수(Objective Function)를 정의한다.

2. 딥러닝(Deep Learning)

Gradient 및 역전파 방법에 기반한 기존 뉴럴 네트워크의 문제점이었던 느린 속도는 소위 말하는 “Vanishing gradient problem”에 그 원인이 있었다. Vanishing gradient problem이란 레이어의 수가 많아지면서 gradient가 전파되지 않고 사라지는 것을 말하며 그 결과 지역 최소값에 머무는 결과를 가져오게 된다. Geoffrey Hinton 교수는 이러한 문제를 해결하기 위해 Feedforward Neural Network의 각 레이어에 대해 “사전 트레이닝”(pre-training)을 실시하여 supervised backpropagation에 사용될 수 있게끔 처리하였다. 기계학습 분야에서 overfitting 문제를 해결하기 위해 L1, L2 정규화(regularization)를 사용하는데 최근 딥러닝에 적용되는 정규화 방법으로 dropout 방법이 제안되었다. Dropout이란 훈련 단계에서 무작위로 노드가 일부 선택되어

제외시키는 것을 말한다.

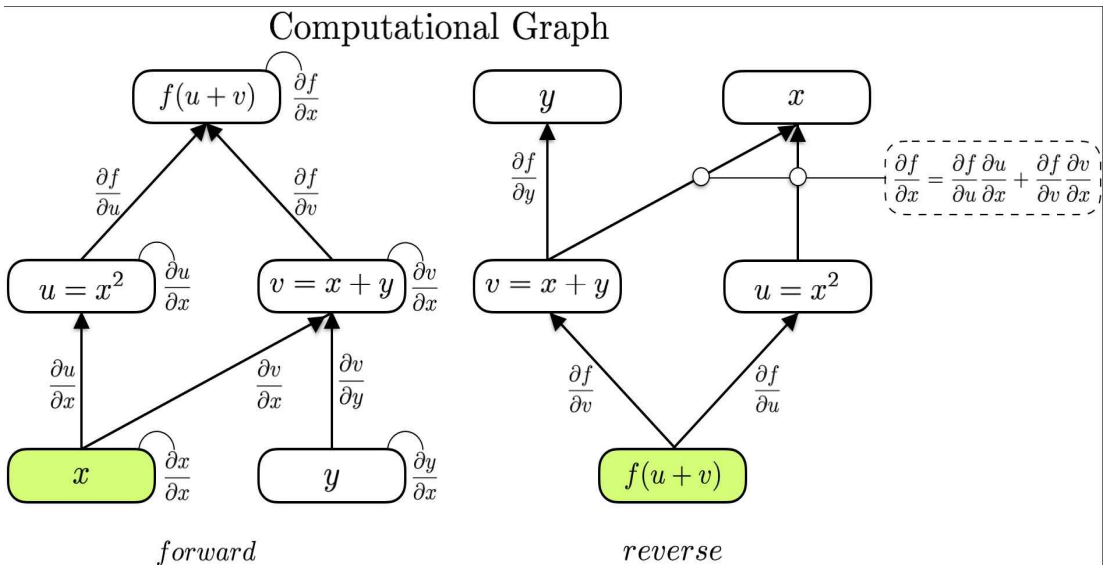

그림 1. Computational Graph

3. GPU 컴퓨팅

2015년 3월 실리콘밸리에서 열린 GTC (GPU Technology Conference)에서 강조된 주제는 바로 딥러닝이었다. 이는 딥러닝에 필요한 연산능력을 기반으로 하고 있는데 기존 CPU만을 활용하는 연산과 GPU를 활용하는 연산 성능이 크게 차이나기 때문이다. 예를 들어, 인텔 제온 16코어 CPU가 43일 걸리는 작업을 NVIDIA의 맥스웰 아키텍처가 적용된 3,072개의 CUDA (Compute Unified Device Architecture) 코어가 포함된 타이탄 X를 사용하면 3일까지 줄어들게 된다. 이는 서버 1,000개를 병렬로 연결한 구글 브레인에 비해 비용이나 전력 소모 면에서 장점을 가지고 있다고 할 수 있다. CUDA는 그래픽 장치의 병렬처리 능력을 C 언어와 같은 산업체 표준 언어를 통해 사용할 수 있도록 하는 기술이다. 비영리 기술 컨소시엄인 크로노스그룹에 의해 관리되는 OpenCL은 개방형 범용 병렬 컴퓨팅 프레임워크이다. 현재 딥러닝 관련 커뮤니티 및 라이브러리들이 대부분 CUDA 기반이므로 OpenCL은 상대적으로 많이 사용되지 않는다.

III. TensorFlow

1.TensorFlow의 특징

TensorFlow의 특징은 다음과 같다

-엔진은 C/C++로 작성되었으며 API는 파이썬 언어(C/C++ 포함)로 제공됨

-딥 러닝 뿐만 아니라 reinforcement learning과 같은 다른 종류의 학습 알고리즘도 지원

-오픈 소스 형태로 공개한 점은 관련 분야 인재 확보를 위한 의도도 있어 보임

-기업용 소프트웨어 성격이라기 보다는 연구 지원 소프트웨어 성격이 강함

-Theano오 마찬가지로 계산 그래프(computational graph)를 생성하며 자동 미분을 실행함

-구글이 발표한 TensorFlow외 구글 내부용으로 사용될 것으로 추측되는 딥러닝 관련 툴이 존재할 것으로 예상

-만약 기업에서 TensorFlow를 도입할 때 가장 먼저 고려해야 할 점은 구글 솔루션에 종속되는지를 먼저 결정해야 함

2. 관련 라이브러리와의 비교

○ TensorFlow

-모델 확장성: TensorFlow에서 행렬 계산 및 comvolution은 심볼릭 그래프를 이용하여 수행하며 사용자는 복잡한 신경망 구조를 역전파(back-propagation) 구현에 구애받지 않고 만드는 것이 가능. 현재 그래프 상에서 반복, 조건 분기와 같은 기능은 구현되어 있지 않은 상태. 이 점은 RNN (Recurrent Neural Network)을 구현할 때 파이썬 언어가 제공하는 반복 메커니즘을 사용해야 한다는 점과 그래프 컴파일러 최적화를 할 수 없다는 점에서 단점으로 작용

-인터페이스: 파이썬과 C/C++ 언어를 기본적으로 제공, 이는 네이티브 코드 또는 속도가 중요시되는 환경에서 장점으로 작용

-범용성: ARM 아키텍처에서도 사용이 가능, 이는 단말에서 서버에 이르기까지 다양하게 사용될 수 있음을 의미. 마이크로소프트 윈도우는 현재 미지원 상태임.

-성능: cuDNN v2(NVIDIA) 기준 Torch에 비해 1.5배 느린 성능을 보이며 배치 사이즈 128 GoogleNet의 경우 메모리 부족 현상을 보임.

-다중 GPU 지원: GPU에서 CPU로 메모리 전송, 다수의 GPU상에서 얻어진 결과 취합 등이 비교적 용이함. 분산 트레이닝은 향후 포함될 예정임

-모델 디버깅: TensorBoard를 사용하여 변수 추적이 용이하도록 설계되어 있음.

○ Theano

-모델 확장성: 거의 대부분의 학습 알고리즘이 구현되어 있으며 심볼릭 그래프를 계산에 도입한 흐름을 만든 사례에 해당. scan이라는 반복 메커니즘을 제공하는데 이는 RNN 구현에 장점으로 작용. 사용자는 이미 만들어져 있는 tensor 연산 프레임워크를 사용하면 되기 때문에 모델 정의와 학습이 단순함

-인터페이스: 파이썬 언어만을 지원

-범용성: 네이티브 언어를 지원하지 않으며 파이썬 인터프리터의 비효율성으로 인해 산업계에서 그다지 많이 사용되지 않음.

-성능: 대규모 네트워크의 경우 Torch7과 동일한 성능을 보임, 하지만 C/CUDA 코드를 컴파일해서 바이너리 코드를

만들어내는 시간이 오래 걸림

-다중 GPU 지원: 네이티브 코드 상에서 지원은 하지 않음. 개발자는 로우레벨 프로그래밍 관점에서 직접 구현해야 함

-모델 디버깅: 없음

○ Torch

-모델 확장성: convolutional 신경망 구현이 우수함, 특히 temporal convolution의 경우 직관적 사용이 용이함. 새로운 레이어 타입을 정의하는 경우 사용자는 포워드, 백워드, 그래디언트(gradient) 입력의 갱신 모듈을 모두 구현해야 함.

-인터페이스: LuaJIT 환경을 기반으로 실행되며 사용자는 심볼릭 프로그래밍에 대한 부담이 없음.

-범용성: LuaJIT 환경에서 실행되므로 API 수분에서 다른 요소와 통합 작업이 용이하지 않음

-성능: 가장 나은 성능을 보임

-다중 GPU 지원: TensorFlow와 Theano의 중간 단계임

-모델 디버깅: 없음

3. TensorFlow 프로그래밍

3.1 개요

TensorFlow 프로그래밍에서 전체 계산은 그래프 구조로 표현한다. 그래프에서 노드는 ops (operations)에 해당되며, op는 입력 값으로 tensor를 받아들여 미리 정의된 계산을 수행한 뒤, 출력 값으로 다시 tensor를 만들어낸다. 여기서 tensor란 타입이 정의된 다차원 배열이다. 예를 들어, 몇 개의 이미지 데이터를 묶어(이를 "mini-batch"라고 함) training/test 목적으로 사용할 경우 tensor는 차원(dimension)이 4인([batch, height, width, channels]) 다차원 배열이다.

TensorFlow에서 계산은 Session을 통해 실행되는데, Session은 실행하고자 하는 그래프 ops를 디바이스(CPU 또는 GPU)에 로드하게 된다.

3.2 "Hello World" 프로그램

```
import tensorflow as tf
hello=tf.constant('Hello, TensorFlow')
s=tf.Session()
print s.run(hello)
```

위 코드에서 먼저 문자열 "Hello, TensorFlow"를내용으로 하는 상수 타입의 텐서를 만든 다음, Session 객체에 TensorFlow ops(이 경우 hello)를 전달한 다음 실행한다.

```
a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])
b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])
with tf.Session() as sess:
    print sess.run(tf.matmul(a,b))
```

위 코드는 상수 타입의 텐서를 만들 때, "모양(shape)"을 지정한 다음 행렬의 곱 계산을 수행한다.

```
import tensorflow as tf
import numpy as np
x = tf.placeholder("float", shape=(1024, 1024))
y = tf.matmul(x, x)

with tf.Session() as sess:
    rand_array = np.random.rand(1024, 1024)
    print sess.run(y, feed_dict={x: rand_array})
```

위 코드는 sess.run의 두 번째 인수 feed_dict를 사용하여 텐서 값의 오버라이딩(x 값 변경) 한다. 1024*1024의 행렬을 생성한 뒤 두 행렬의 곱을 계산한다.

아래 코드에서 변수 x, y는 텐서에 feeding되는 대상이며 dtype은 32비트 정수, 차원은 2*2의 매트릭스이다, 따라서 c 객체를 실행할 때 feeding 되는 실제 텐서는 placeholder에서 정의한 차원과 동일하게 값을 정해줘야 한다. 예를 들어, 텐서 x는 1행의 값이 [2,2], 2행의 값이 [3,3]인 2*2 행렬이다.

```
# tensor 'x' is [[2, 2]], [3, 3]]
x = tf.placeholder("int32", shape=(2, 2))
# tensor 'y' is [[8, 16], [2, 3]]
y = tf.placeholder("int32", shape=(2, 2))

c=tf.pow(x, y)
with tf.Session() as sess:
    print sess.run(c, feed_dict={x: [[2, 2], [3, 3]],y:[[8, 16], [2, 3]]})
```

```python
import tensorflow as tf
import numpy as np

# Make 100 phony data points in NumPy.
x_data = np.float32(np.random.rand(2, 100)) # Random input
y_data = np.dot([0.100, 0.200], x_data) + 0.300

# Construct a linear model.
b = tf.Variable(tf.zeros([1]))
W = tf.Variable(tf.random_uniform([1, 2], -1.0, 1.0))
y = tf.matmul(W, x_data) + b

# Minimize the squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# For initializing the variables.
init = tf.initialize_all_variables()

# Launch the graph
sess = tf.Session()
sess.run(init)

# Fit the plane.
for step in xrange(0, 201):
    sess.run(train)
    if step % 20 == 0:
        print step, sess.run(W), sess.run(b)

# Learns best fit is W: [[0.100  0.200]], b: [0.300]
```

그림 2. TensorFlow 선형모델 예제

3.1 선형 모델

그림 2는 가장 단순한 선형(linear) 모델을 사용한 예제이다. 이 예제에서 학습은 먼저 데이터 준비(training 목적의 데이터), loss function 정의, train 방법의 정의, session을 통한 계산으로 이루어진다. 라인 5에서 구간 [0, 1]사이의 값으로 길이가 2인 column 벡터 100개를 무작위로 만들어낸다. 라인 6에서는 가중치 벡터 [0.1, 0.2]T와 bias 0.3을 사용하여 타겟 값으로 이루어진 row 벡터(1*100)를 만들어낸다. 라인 9, 10에서는 training단계에서 추정할 선형 모델 파라미터 w와 b를 초기화한다. 라인 11에서는 입력 값(x_data)에 파라미터 w와 b를 적용하여 예측 값 y를 계산한다. 이 경우 loss function은 (y-y_data)2으로 정의된다. 라인 15에서 learning rate 0.5인 gradient descent 최적화 방법을 정의했다.

라인 16에서 loss function 최소화를 지정한 다음, 라인 27에서 training을 수행한다. 200여 회의 반복 단계를 수행하고난 뒤 파라미터 w 벡터 및 b의 추정 값은 [0.10001153, 0.20000879]T, 0.29998931이다.

그림 3은 MNIST 데이터를 사용한 예제이다. 라인 4에서, 픽셀 차원이 28*28인 handwritten 이미지 저장 용도의 길이 784의 벡터를 준비한다. 라인 10에서는 가중치에 해당하는 784*10 크기의 행렬을 준비한다. handwritten 이미지 학습은 분류(classification)에 해당되며 따라서 1에서 10까지의 클래스 조건부 확률을 계산하게 된다. 예제에서 사용된 모델은 softmax로서 데이터 x가 클래스 i일 확률을 나타내며 다음의 형태이다.

라인 9에서 x, w, b를 사용하여 모델을 정의했다. x는 라인 24에서 실제 계산을 할 때, feeding을 통해 MNIST 데이터를 전달하게 된다. Softmax를 사용하여 분류작업을 하는 경우 cross-entropy를 loss function으로 정하게 되는데 라인 13이 해당된다. 라인 14에서 learning rate를 0.01로 정의한

```
1  import input_data
2  mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
3  import tensorflow as tf
4  x = tf.placeholder("float", [None, 784])
5  W = tf.Variable(tf.zeros([784,10]))
6  b = tf.Variable(tf.zeros([10]))
7  # placeholder(dtype, shape=none, name=none)
8
9  y = tf.nn.softmax(tf.matmul(x,W) + b)
10 # softmax(logits, name=none)
11
12 y_ = tf.placeholder("float", [None,10])
13 cross_entropy = -tf.reduce_sum(y_*tf.log(y))
14 train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
15 init = tf.initialize_all_variables()
16 sess = tf.Session()
17 sess.run(init)
18 for i in range(1000):
19     batch_xs, batch_ys = mnist.train.next_batch(100)
20     sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
21
22 correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
23 accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
24 print sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
25
```

그림 4. Softmax 분류 예제

다음, 라인 19에서 크기 100의 배치(batch)를 정의했다. 라인 22에서 해당 이미지가 해당되는 숫자와 모델에서 추정한 숫자의 비교를 통해 정확도를 계산한다.

4. 실행환경

TensorFlow 실행환경에서 client는 session 인터페이스를 사용하여 master와 통신한다. 한 개 이상의 worker 프로세스는 CPU나 GPU 디바이스에 접급을 중재하는 역할 및 master가 전달한 그래프 노드를 실행한다(그림 5 참조). 실행환경은 지역(local) 환경 및 분산 환경으로 나뉘는데, 지역 환경이란 client, master, worker가 모두 하나의 디바이스 상에서 실행되는 것을 말한다.

이와 반면에 분산 환경은 client, master, worker가 다른 디바이스 상의 다른 프로세스에서 실행되는 것을 말한다. TensorFlow에서 디바이스는 아주 중요한 요소인데 각각의 디바이스에는 디바이스 타입과 이름이 연결되어 있다. 디바이스 이름에 포함되는 정보로는 디바이스 타입, worker 프로세스 내 인덱스 값 그리고 분산 환경의 경우 worker 프로세스의 job, task 아이디가 포함된다. 예를 들어, "/job:localhost/device:cpu:0" 또는 "/job:worker/task:17/device:gpu:3"과 같은 형식이다. 각각의 디바이스 객체는 디바이스 메모리의 할당 및 해제, 커널 실행 등의 작업을 담당한다.

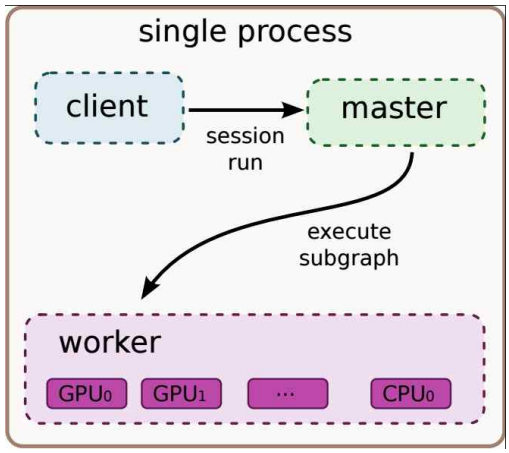

그림 5. TensorFlow 실행 요소

Ⅳ. 결 론

구글이 발표한 TensorFlow는 여러 가지 파급효과가 발생할 것으로 예상된다. 첫 번째로 오픈소스 형태로 발표했기 때문에 향후 기계학습 오픈소스 커뮤니티가 더욱 활성화될 것이라는 점이다. 이는 구글이 안드로이드 운영체제를 오픈소스로 공개한 것과 맥락을 같이한다. 두 번째로 GPU 컴퓨팅 활용이 더 확대됨으로 인해 기계학습 분야의 패러다임이 변화할 것이라는 점이다. 세 번째로 안드로이드 및 iOS와 같은 환경에서 실행이 가능해짐으로 인해 광범위한 하드웨어 상에서 기계학습 솔루션이 채택될 것이란 점이다.

참 고 문 헌

[1] http://www.tensorflow.org/

[2] Jeffrey Dean and Rajat Monga, TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Preliminary White Paper, November 9, 2015.

[3] https://en.wikipedia.org/wiki/Automatic_differention

[4] http://www.itworld.co.kr/news/96498

저자소개

김 종 영

1996: 한양대학교 전자계산학과 공학사.

1998: 한양대학교 컴퓨터공학과 공학석사.

2012: 한양대학교 컴퓨터공학과 박사과정 (박사 수료)

관심분야: Bayesian Optimization