**Machine Learning**

**Machine Learning**

목차

[1. Why Machine Learning](http://www.libqa.com/wiki/852)

[2. Data Analytics](http://www.libqa.com/wiki/853)

[3. Big Data](http://www.libqa.com/wiki/854)

[4. 환경 구성](http://www.libqa.com/wiki/855)

[5. 첫번째 모형 - 선형회귀분석](http://www.libqa.com/wiki/858)

[6. Machine Learning이란](http://www.libqa.com/wiki/859)

[7. 파이썬 Basic](http://www.libqa.com/wiki/860)

[8. 파이썬 Advanced](http://www.libqa.com/wiki/861) - 함께 쓰면 유용한 패키지

[9. 텐서플로우 Basic](http://www.libqa.com/wiki/862)

[10. 지도학습](http://www.libqa.com/wiki/863)

[11. 비지도학습](http://www.libqa.com/wiki/864)

[12. 인공 신경망](http://www.libqa.com/wiki/865)

13. 앞으로 우리가 해야할 것들

참고: [체험 인공지능 정기철, 홍릉과학출판사](http://www.yes24.com/24/goods/26792943?scode=032&OzSrank=1), 숭실대학교 정보과학대학원 머신러닝 정기철 교수님 수업 보충 자료

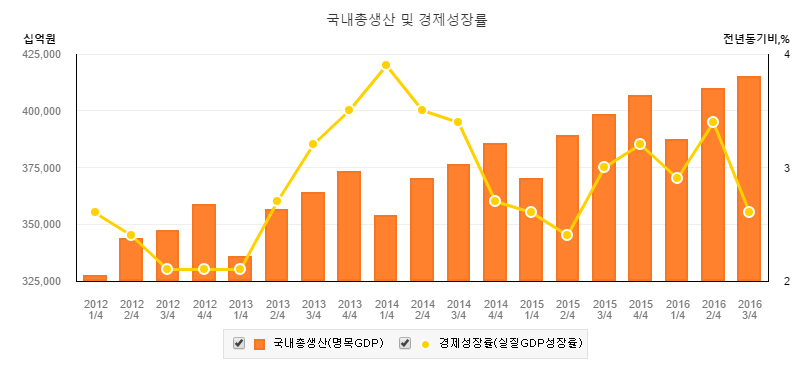
        그리고 안명호님의 Machine Learning Crash Course 자료

        홍콩과기대 김성훈 교수님 자료: <http://hunkim.github.io/ml/>

**1. Why Machine Learning**

왜 머신러닝을 공부해야 할까?

이 문제에 대해서 생각해 보기 전에 먼저 현대 사회를 살펴보자.



지난 경제성장률에 관한 한국은행 자료를 살펴보면 우리나라는 2~3% 성장률을 자랑(?)하고 있다.

2017년 경제성장률 전망도 2.5% 정도이다.

열심히 일해서 나의 생산성이 높아지면 높아질수록... 내 옆 자리를 비워야 하는 경제상황이다. ㅜ\_ㅜ

나만 잘 먹고 잘 살면 되지? 안그래? 아 몰랑~

그런데, 회사에서 나에게 혁신을 요구한다.

창조경제혁신센터???

뭔 놈의 혁신이 이렇게 유행인거야???

혁신이 무엇일까? 검색해 보자!

혁신, 革新

﻿묵은 풍속·관습·조직·방법 등을 바꾸어 아주 새롭게 하는 것.

뭔가 새롭게 바꿔야 한다?. 왜왜왜? 도대체 왜? 지금 잘먹고 잘살고 있는데 왜왜왜? 왜 바꿔!!!

혁신이 필요한 이유는 경제성장률과 관계가 깊다.

사람은 사회적 동물이고 혼자서 잘 먹고 잘 산다고 그것이 지속가능하지 않기 때문인데, 어떠한 혁신이 필요할까?

내가 노력해서 나의 전문성이나 생산성만 늘리는 방법은 현대의 혁신이 아니다.

어떤 좋은 것을 도입해서 비용절감을 도모하는 것도 혁신이 아니다.

예전에 높은 경제성장률을 자랑하던 시절에는 무조건 열심히만 하면 되었고, 생산성을 높이거나 비용 절감을 하는 것이 중요했다.

지난 날의 혁신은 비IT였던 분야를 IT화(자동화)하는 것만으로도 충분한 혁신이였다.

여기저기서 IT 프로젝트를 했고, 수작업하던 일들이 자동화되고 컴퓨터와 기계가 업무를 처리를 했다.

자동화로 인한 일손이 남아도 문제가 되지 않았다.

높은 경제성장률 속에서 새로운 일들이 계속 생겼고, 회사들은 반복적인 일들은 기계한테 시키고 사람은 그들의 핵심역량에 집중했다.

2000년대 초반 BPR (Business Process Re-engineering) 도입이 붐이었다.

은행도 BPR을 도입하면서 창구에서 입력하는 일을 줄였다.

경제성장률이 떨어지기 시작한 것은 이 때 부터였던 것 같다. (실제로는 IMF를 겪으면서 거품이 빠지기 시작했고, 경제성장률은 점차 낮아지기 시작했다.)

BPR이 도입되면 창구 직원들의 일손이 감소하고, 결국엔 짤리게 된다고 걱정했다.



변화의 속도가 엄청 빠르다. 이제는 모든 사람들이 컴퓨터 한 대씩 들고 다니는 시대가 되었다.

내 iPhone 용량이 128G다. 내 생애 첫 번째 컴퓨터는 94년도에 샀던 486DX2 50 이었다. 하드 디스크가 256M 였다.

500배가 넘는 용량의 컴퓨터를 들고 다니는 시대가 된 것이다.

현대의 혁신은 어떤 혁신일까?

비IT를 IT화 하는 것이 예전의 혁신이었다면, 현대는 IT화 안된 분야가 거의 없다.

혁신하기가 너무 힘든 세상이다.

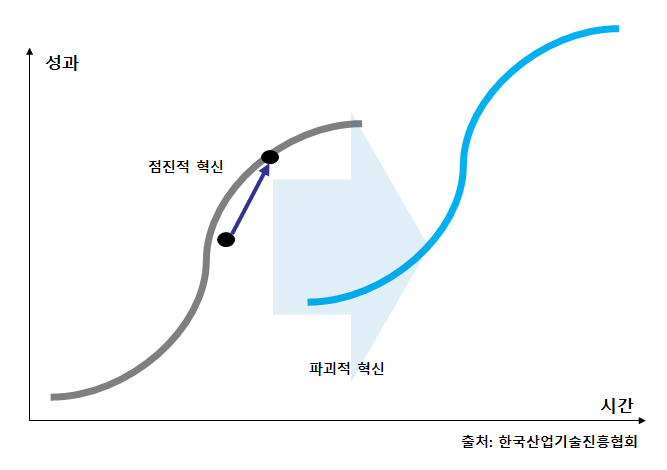
비IT를 IT화 하는 것도 생산성을 높이거나  비용절감 차원으로 접근한다.

이런 혁신은 안하니만 못하다.

**앞으로의 혁신은 파괴적 혁신!**

앞으로의 혁신은 새로운 시장을 창출해 내는 것이여야 한다. 새로운 일자리를 만들어 내는 혁신이라야 한다.

이러한 혁신을 파괴적 혁신이라 부른다.



위 그림에서 나타내듯이 기존의 혁신은 점진적 혁신이었다. 앞으로의 혁신은 파괴적 혁신을 해야 하는데,

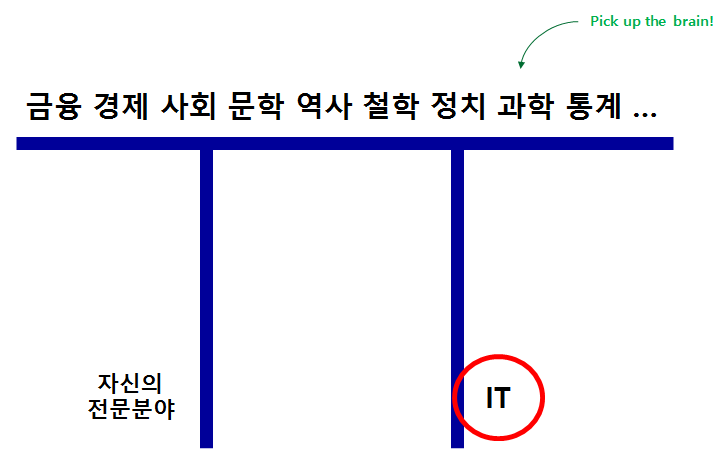
그림에서 보듯 S성장곡선이 우상승하는 쪽으로 이동하는 혁신이 파괴적 혁신이다.

새로운 분야를 창출해서 일자리를 창출하고 부를 증가시키는 창조적인 혁신을 말한다.

현대는 모든 사람들이 컴퓨터를 적어도 한 대씩 들고 다니는 초 디지털 사회이다. 이런 사회에서 IT를 모르면서 혁신을 한다는 것은 있을 수 없다.

IT만 알아서도 혁신을 할 수 없다. 자신의 전문 분야를 적어도 한 개는 가지고 있어야 그 분야에서 만큼은 IT를 이용한 혁신이 가능하다.

그래서 요즘 사회에서 원하는 인재상은 예전의 **T자형** 인재에서 **ㅠ자형** 인재이다.



넓고 얉은 다양한 분야에 대한 이해와, 자신의 깊은 전문 분야 한가지!

그리고 다른 깊은 전문 분야 한 가지는 바로 IT로 삼아야 한다. (내가 IT를 하는 사람이어서 말하는 소리이긴 하다.)

IT를 모르고서 자신의 전문 분야에서 성과를 나타낼 수는 있겠지만, 파괴적 혁신을 하지는 못할 것이다.

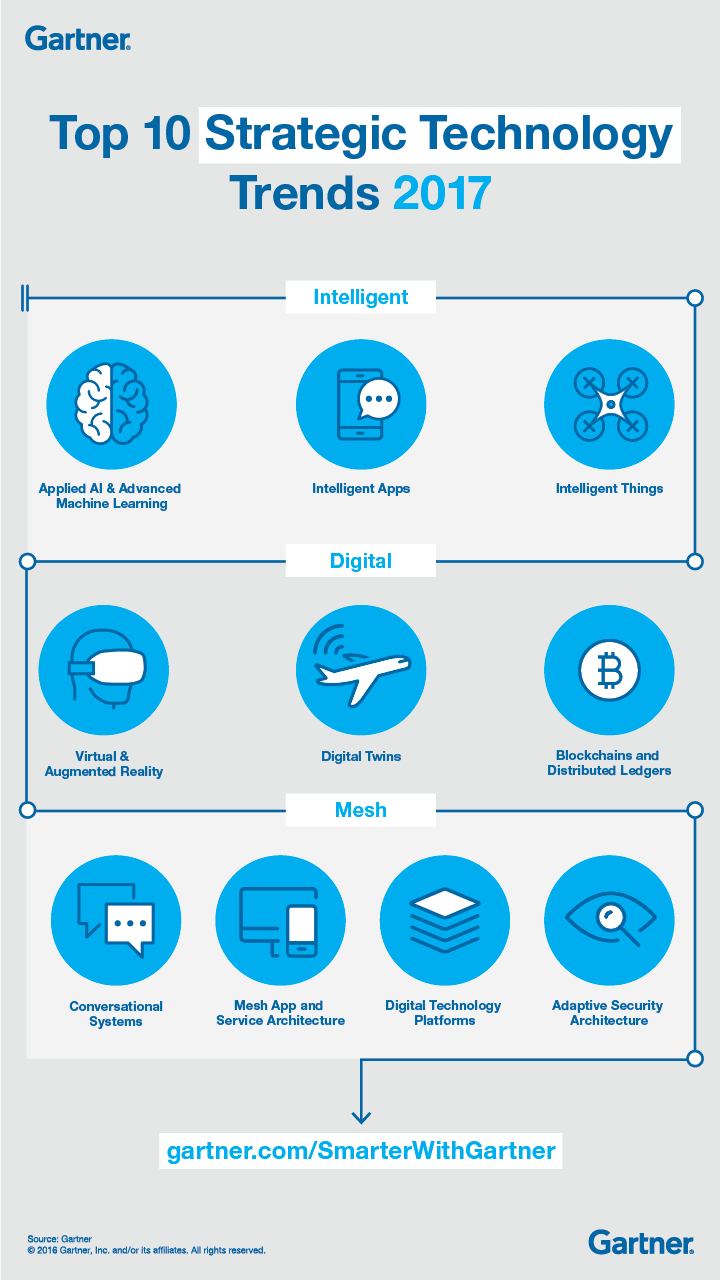
점진적 혁신만으로는 초경쟁 사회에 빠질 수 밖에 없고, 옆 자리에 앉은 동료를 밟고 올라가야만 살아남을 것이기 때문이다.

넓고 얉은 다양한 분야는 Pick up the brain! 할 수 있는 수준까지만! 즉, 해당 분야의 전문가의 지식을 읽고(또는 듣고) 이해할 수 있는 수준은 되어야 한다.

그런데, IT분야도 정말 넓고 넓다. 어떤 IT를 내 전문 분야로 삼아야 할까?

Pick up the brain을 해보자!

다음은 가트너의 2017년 10대 전략 기술이다.



한글로 정리하면 다음과 같다.

1. **인공지능과 고급 머신 러닝**
2. **지능형 앱**
3. **지능형 사물**
4. **가상 현실 및 증강 현실**
5. **디지털 트윈**
6. **블록체인과 분산 장부**
7. **대화형 시스템**
8. **매시 앱 및 서비스 아키텍처**
9. **디지털 기술 플랫폼**
10. **능동형 보안 아키텍처**

이 기술들은 모두 IT기술이다. 2016년까지 3D 프린팅 기술이 전략기술로 한 자리를 차지하고 있었지만 어느새 빠져버렸다.

그 기술이 별볼일 없어졌다는 뜻이 아니라 이미 성숙해 졌다는 뜻이다.

인공지능과 머신러닝은  2016년에는 중간 정도(5번)에 있었다. 2017년에는 어느 덧 1번을 자리를 떡하니 차지하고 있다.

각각에 대한 설명은 스터디 모임을 통해서 말씀드리는 것으로 하겠다.

우리(스터디그룹: 뜻밖의 모임)가 저 10개 중에 하나를 붙잡고 깊게 판다면 과연 혁신할 수 있을까? (그러기엔 우린 너무 늙었다...)

이번 주제가 머신러닝이니까 전략기술이 1번이 머신러닝이라며 그래서 머신러닝을 공부해야 한다고 주장하는게 맞을까?

우리가 Pick up the brain 해야 하는 것은 10대 기술 자체를 전문분야로 삼고자 하는 것이 아니다.

10대 기술에 숨어 있는 인사이트를 얻는 것이 중요하다.

10대 전략 기술을 관통하는 한 가지 인사이트가 있는데 그것은 바로 **"Data Analytics"** 이다.

즉, 데이터를 읽고 이해할 수 있는 능력을 갖춰야 한다. 인공지능과 머신러닝도 데이터가 있어야 가능하다.

지능형 앱, 지능형 사물, 대화형 시스템, 디지털 기술 플랫폼도 데이터를 통해서 가능하다.

가상현실 및 증강현실은 데이터를 어떻게 보여줄까의 문제이고, 디지털 트윈은 데이터를 생성해내는 사물의 컨트롤 부분이다.

능동형 보안 아키텍처는 데이터를 어떻게 통제하고 보호할 것인가의 문제이다.

우리 주위에는 수많은 데이터가 존재한다. 그 데이터는 점점 더 많아 질 것이다.

인간이 이해하기 어려운 수준의 수많은 데이터들...

엑셀로 처리하기 어려운 비정형이 데이터들...

IoT 사물로 부터 쏟아져 나오는 센서 데이터들...

그러한 데이터를 읽고 분석할 수 있는 능력이 현대 사회에서 요구하는 핵심 능력이다.

이러한 분석 능력을 갖추려면 IT가 아닌 자신의 전문 분야를 하나정도는 깊은 이해와 경험을 해야 한다.

그래서 ㅠ자형 인재가 현대 사회에서 요구하는 인재상인 것이다.

머신러닝도 데이터의 분석 관점에서 바라보았으면 좋겠다.

우리가 알파고를 만들고자 머신러닝을 공부하는 것이 아니다.

로보어드바이저를 만들고자 채팅봇을 만들고자 머신러닝을 공부하는 것이 아니다.

저런 것을 만드는 전문가는 수 없이 많다! 수많은 인공지능 석박사들이 달려들고 있는 분야이다.

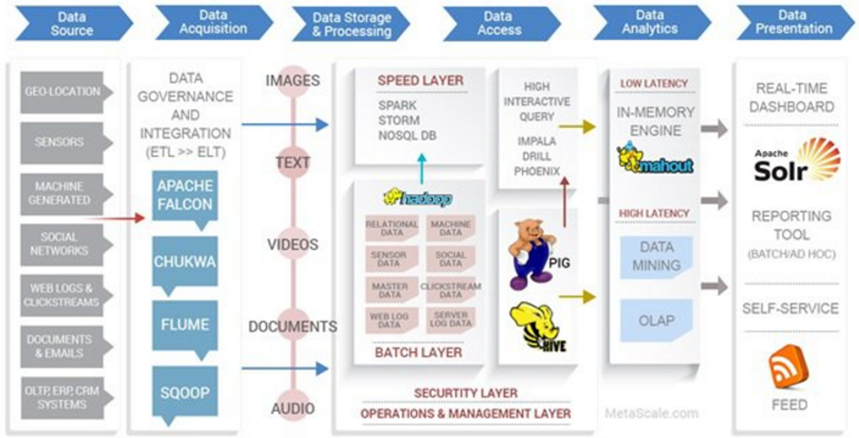
머신러닝을 공부해야 하는 이유는 우리의 전문 분야에서 다루고 있는 데이터를 이해해서 우리 전문 분야를 한 단계 혁신 시키기 위해서다.

데이터 분석을 통해 우리 전문 분야에 숨겨져 있는 새로운 시장을 창출하는데 활용하기 바란다.

# 2. Data Analytics

Data Analytics란 무엇일까요?

다음은 Data Lake Reference Architecture 입니다.



호수와 같이 모든 데이터를 한 곳에 모아서 담아 놓고 필요할 때 꺼내쓴다는 개념입니다.

다양한 데이터 소스로 부터 데이터를 취합하고 데이터 처리를 한 다음 저장해 놓고나서 데이터 분석을 시작합니다.

그리고, 분석 결과를 시각화하는 것까지 데이터 처리의 흐름을 한 눈에 알 수 있는 아키텍처입니다.

Flume이니 Sqoop, Spark, Storm 등 다양한 기술들이 접목되어 해당 아키텍처가 만들어 지는데요.

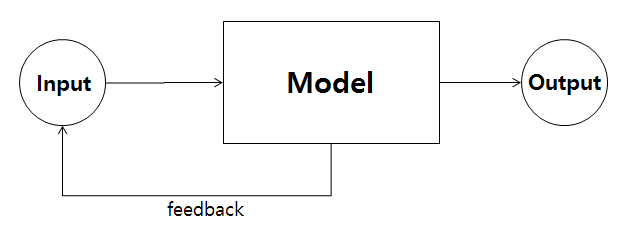
Data Analytics를 위해 이러한 기술들을 익혀야 할까요?

알아두면 좋겠지만 그러한 것들은 해당 전문가들에게 맡기자구요.

우리는 데이터가 있고 그 데이터를 어떻게 분석해서 어떤 결과를 만들 것인지에 집중하면 됩니다.

그럼 무엇을 하면 될까요? 즉, Data Analyst가 하는 일이 무엇일까요?

데이터 분석이라는 것은 바로 **모형**을 만드는 것입니다.



Input도 데이터를 가지고 모형을 만듭니다. 모형이 Input 데이터를 잘 설명하는지 확인해서 이상하면 다시 Input을 바꿔서 모형을 만들고... (과정반복)

그리고나서 모형을 통해 Output을 만드는 것이죠.

Model 자체가 데이터 분석의 Output 이기도 합니다.

데이터 분석이라는 것은 어렵지 않습니다. 대신 왜 해야하는지와 같은 명확한 목표가 필요합니다.

**데이터 분석은 모형을 만드는 것** 다시 한번 기억하세요.

# 3. Big Data

데이터를 분석한다는 것은 모형을 만드는 것이다.

우리가 잘 아는 평균, 표준편차도 모형이다.

데이터들이 어떤 값을 중심으로 모여있는지 보는 것이 평균이라는 모형이다.

데이터들이 평균으로 부터 얼마나 떨어져 있는지 알아 보는 것이 표준편차라는 모형인 것이다.

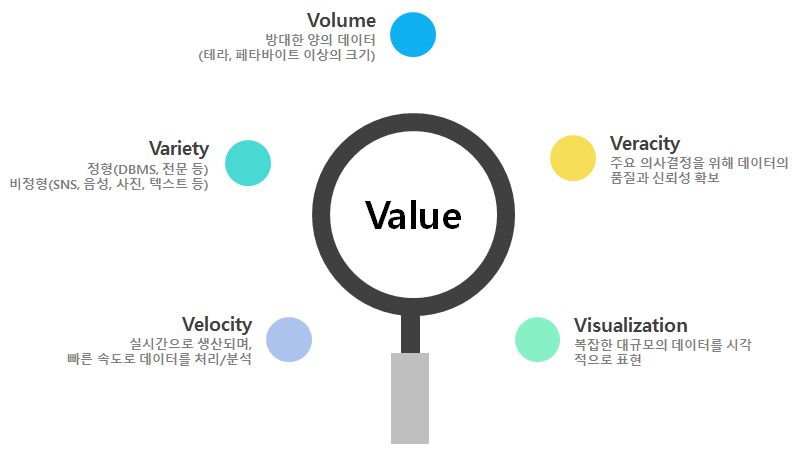
모형에 대해서 본격적으로 알아보기전에 먼저 현대의 데이터 특징에 대해서 알아야 한다.

현대의 데이터는 여러분 들도 익히 들었을 것이다. 바로 **빅데이터(Big Data)**이다.



빅데이터를 이야기할 때 보통 3V로 이야기한다. Volume, Variety, Velocity이 바로 3V이다.

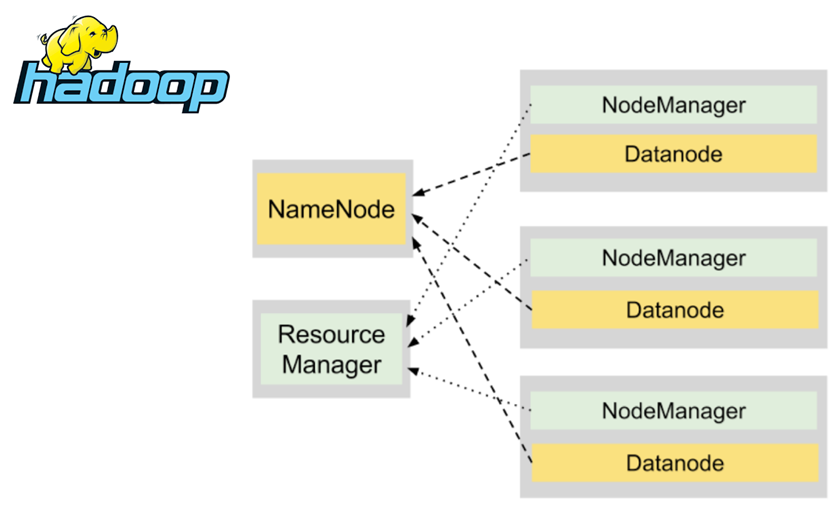
이 개념이 이미 6V로 확장되었다. 자세한 설명은 6V로 하자.



이런 빅데이터 개념이 왜 중요할까?

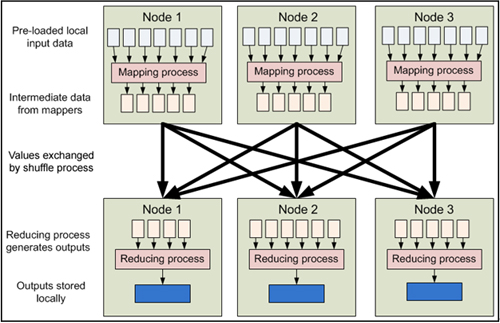
바로 빅데이터를 다루기 위해서는 생각하는 법을 바꿔야 한다.

빅데이터로 생각하기 첫번째 - **분산 처리!!**

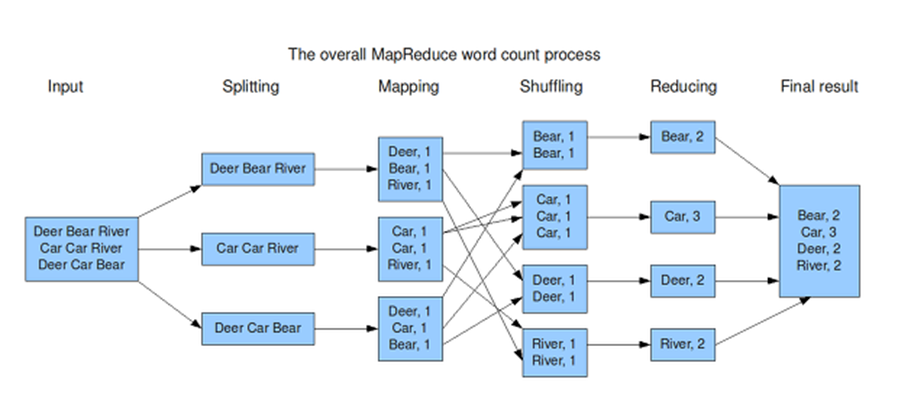


빅데이터하면 Hadoop이고, 하둡은 기본적으로 여러개의 노드로 구성된다. 특히 Datanode의 개수만큼 분산처리가 이뤄지는 구조이다.

하둡의 분산처리 프로그래밍의 핵심은 MapReduce이다.



각각의 노드에서 데이터를 Mapping 프로세스로 잘게 쪼개서 계산을 수행하고, Reducing 프로세스를 통해 결과를 취합하는 구조이다.

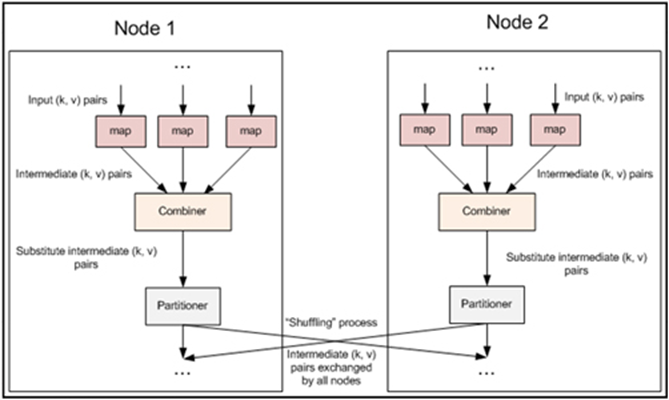


워드 카운트 예제의 전체적인 맵리듀스 프로세스를 나타낸 그림이다.

여기에서 매우 중요한 개념이 있는데 그것은 **"연산은 교환법칙과 결합법칙이 성립해야 한다."** 이다.

교환법칙: A + B = B + A

결합법칙: ( A + B ) + C = A + ( B + C )



Reducing 하는 과정에서 노드간 네트워크 트래픽이 많이 발생할 수 있으므로, Combiner라는 것을 둬서 취합 후 전달할 수 있다.

컴바이너는 성능향상과 교환법칙과 결합법칙이 성립하지 않는 연산을 위해서 필요하다.

빅데이터를 통한 머신러닝을 수행하기 위해서는 분산병렬처리와 MapReduce를 꼭 기억하자.

**4. 환경 구성**

지금 부터 텐서플로우 환경을 셋팅한다.

[**https://github.com/golbin/TensorFlow-Tutorials**](https://github.com/golbin/TensorFlow-Tutorials)

**텐서플로우를 기초부터 응용까지 단계별로 연습할 수 있는 소스 코드를 제공합니다**

텐서플로우는 구글 브레인팀에서 머신 러닝 및 딥러닝 개발을 목적으로 만든 오픈 소스 라이브러리이다.

텐서플로우는 기본적으로 Python과 C++ 언어로 프로그램을 작성한다. 하지만 SWIG 인터페이스를 통해 JAVA, Ruby 등 다른 언어를 사용하는 것도 가능하다.

텐서플로우는 다음과 같은 순서로 설치한다.

1. 아나콘다 설치

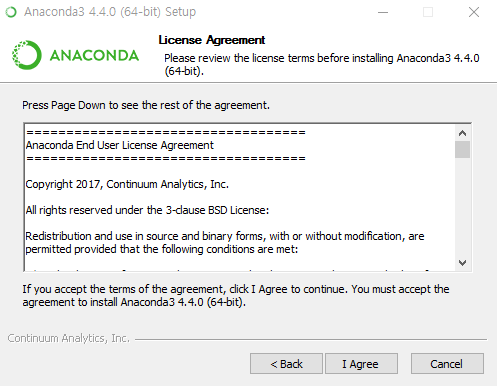
    프로그램 다운로드 URL: <https://repo.continuum.io/archive/>

    URL에서 자신의 OS에 맞는 프로그램을 다운로드 받아 설치한다.

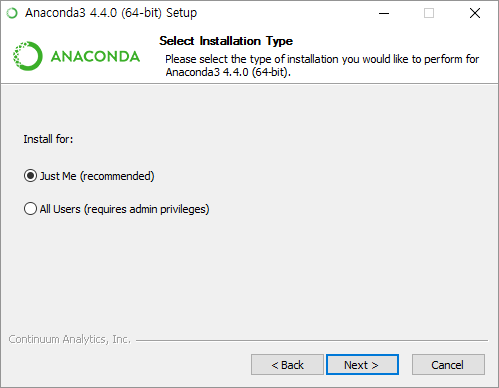
    현재 이 글을 작성하고 있는 PC는 Window10 64bit이므로 [**Anaconda3-4.4.0-Windows-x86\_64.exe**](https://repo.continuum.io/archive/Anaconda3-4.4.0-Windows-x86_64.exe) 프로그램을 설치할 것이다.



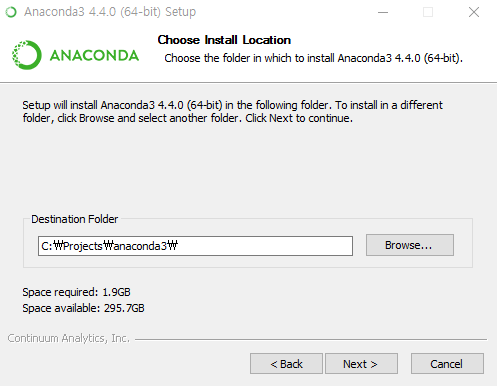
    Next 버튼을 누른다.



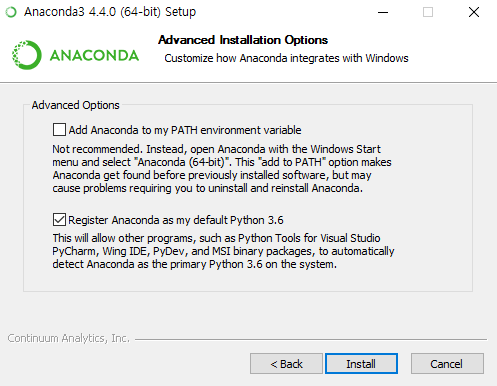
    I Agree 버튼을 누른다.



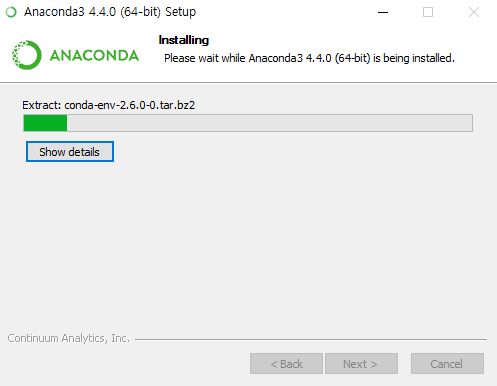
    Next 버튼을 누른다.



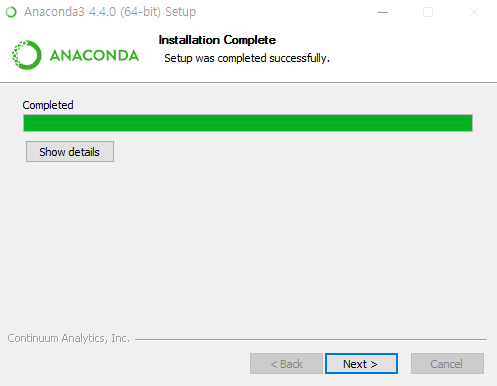
    설치 경로를 지정하고 Next 버튼을 누른다.



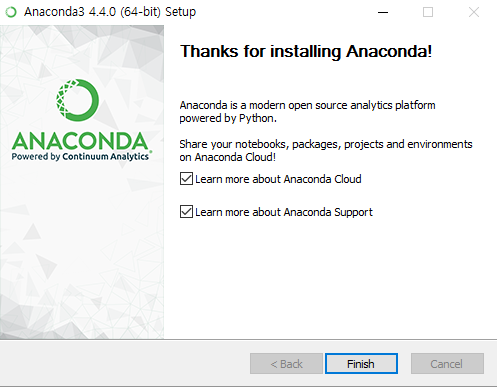
    Install 버튼을 누른다.



    설치가 진행된다.



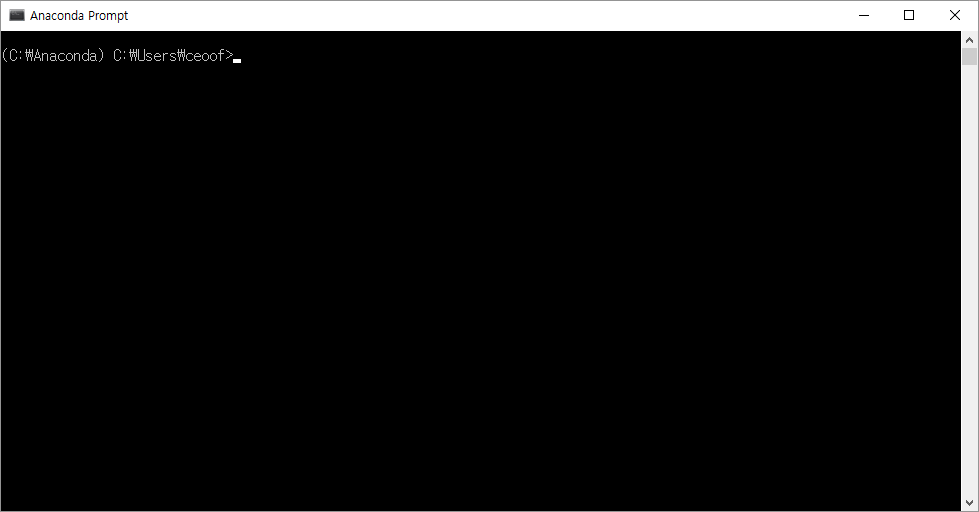
    Next 버튼을 누른다.



   Finish버튼을 눌러 설치를 완료한다.

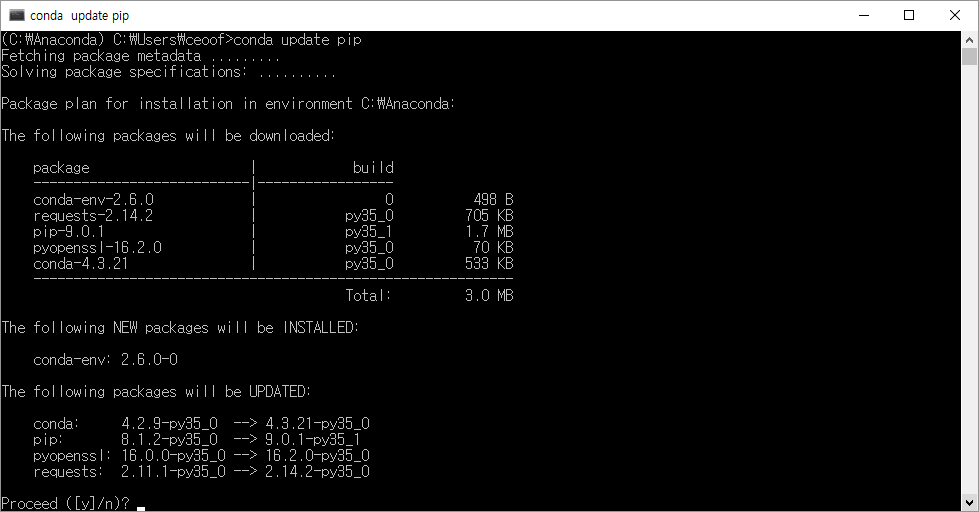
2. 아나콘다 설치 후, 파이썬 최신 패키지로 업데이트 한다.

    Anaconda Prompt를 실행한다.



    먼저 pip 를 업데이트 한다.

**conda update pip**



    이와 같은 방식으로 다음의 명령을 수행한다.

**conda update matplotlib**

3. 텐서플로우 설치

    같은 Anaconda Prompt 창에서 텐서플로우를 설치할 수 있다.

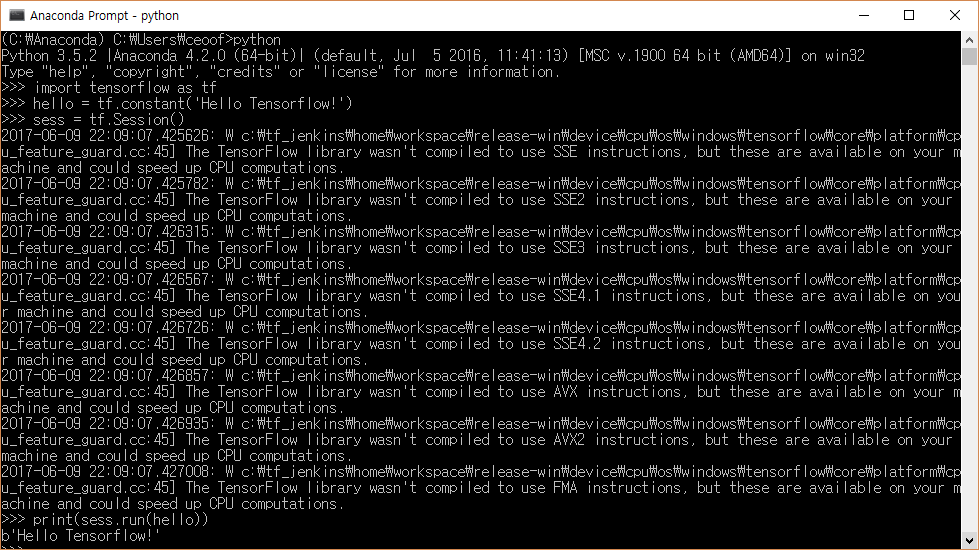
**pip install tensorflow**

    를 실행하여 텐서플로우를 설치한다.

    텐서플로우가 설치되면 제대로 설치되었는지 확인한다.

    Anaconda Prompt에서 python을 입력한 후 다음의 코드를 실행한다.

    import tensorflow as tf  
    hello = tf.constant('Hello Tensorflow!')  
    sess = tf.Session()  
    print(sess.run(hello))



   텐서플로우 설치가 완료 되었다.

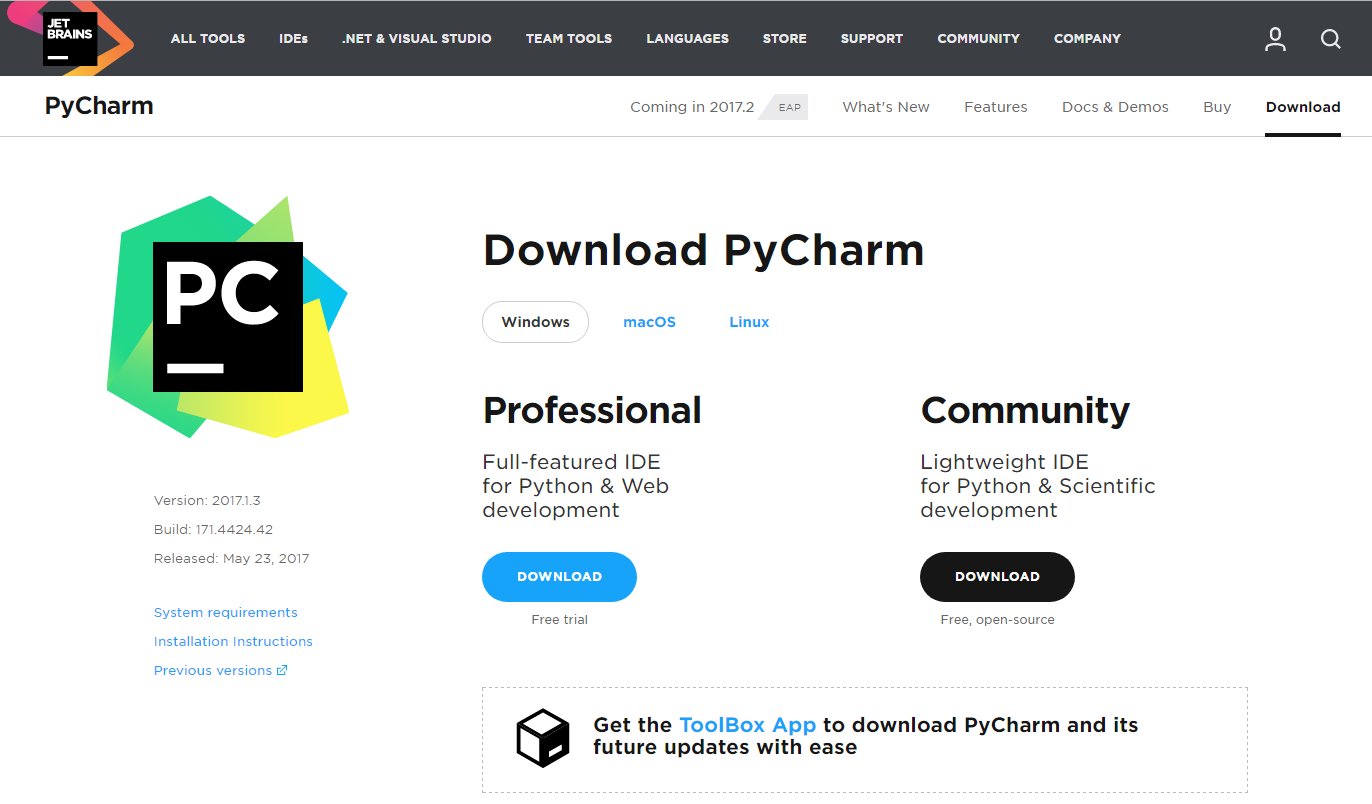
4. 파이참 설치

   이번에는 텐서플로우 작업을 위한 IDE를 설치한다.

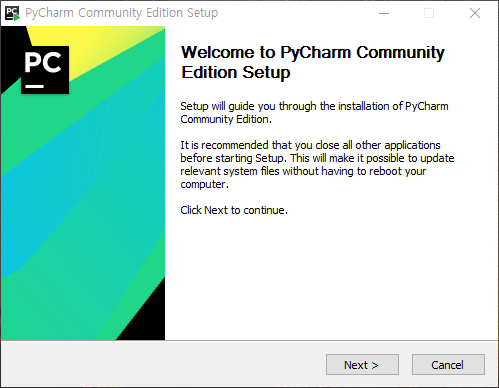
   많은 Python 개발자들이 사용하는 IDE인 파이참을 설치해 본다.

<https://www.jetbrains.com/pycharm/download/>

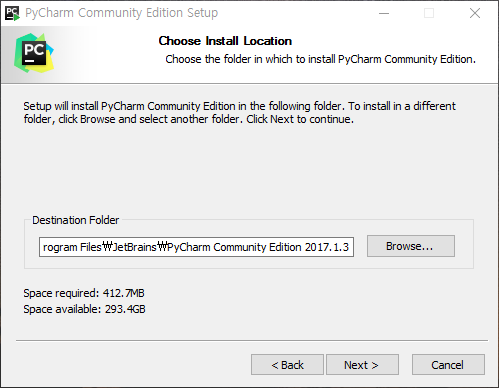
   사이트에 접속해서 자신의 OS에 맞는 파이참을 설치한다.



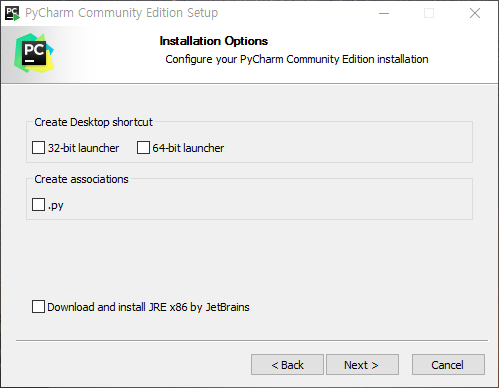
    무료버전인 Community 버전을 설치한다. 다운로드 받은 설치파일을 실행한다.



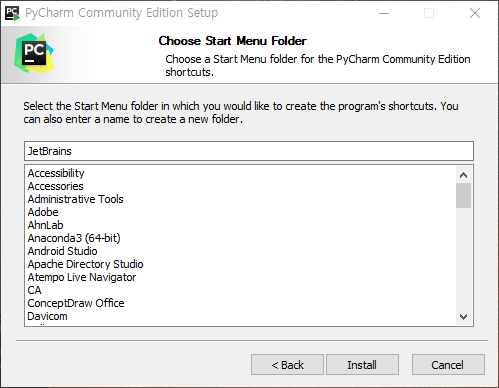
    Next를 클릭한다.



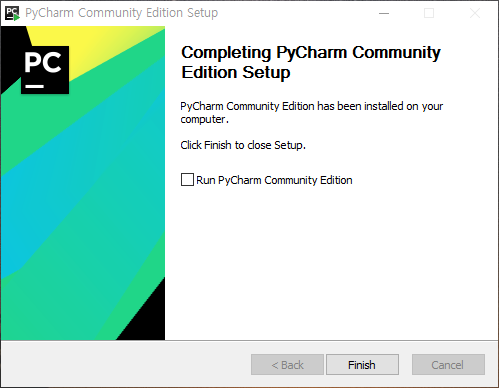
    설치 위치를 지정한 후 Next 버튼을 클릭한다.



    본인이 원하는 옵션을 선택한 후 Next 버튼을 클릭한다.



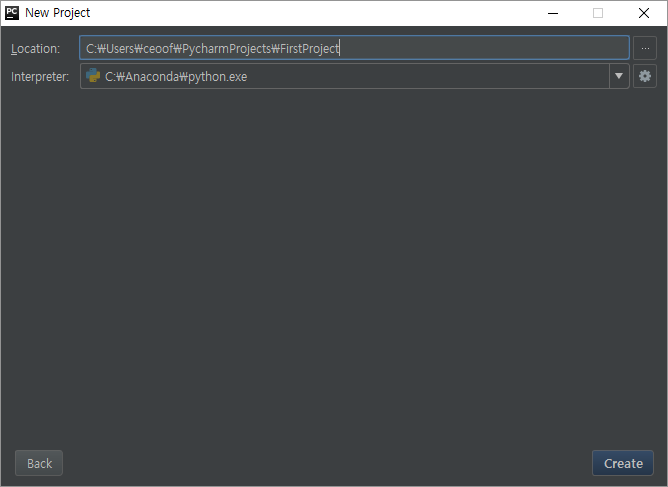
    Install 버튼을 클릭한다.



    Finish를 눌러 설치를 완료한다.

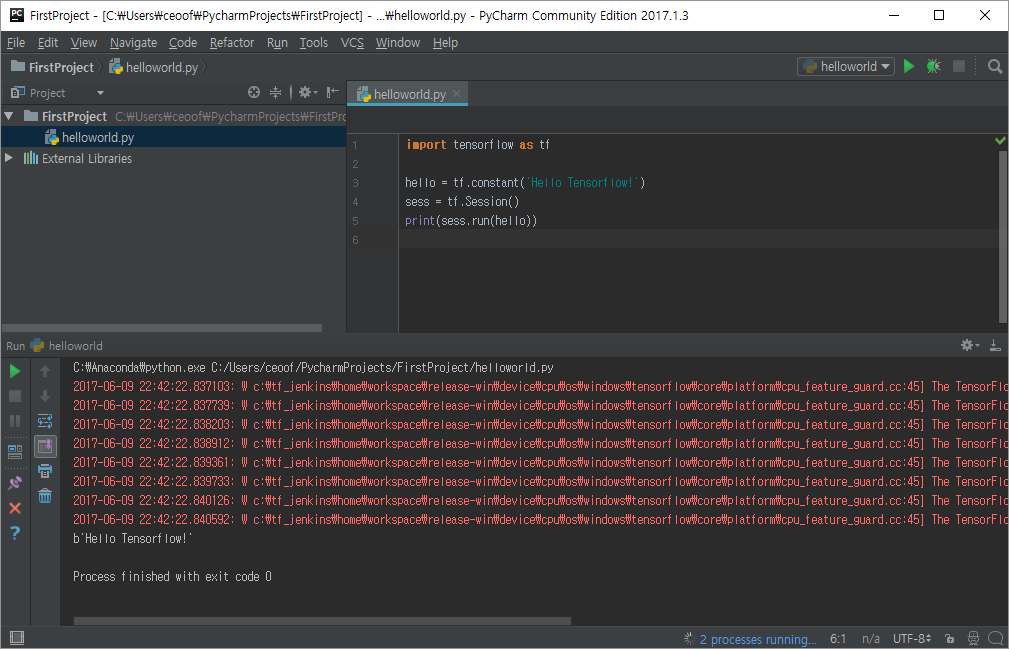
5. 텐서플로우 프로젝트 생성

    파이참을 실행 한 후 **Create New Project**를 선택한다.



    Location 란에 프로젝트 이름을 입력 한 후 Create 버튼을 클릭한다.

    helloworld.py 파일을 만든 후 위에 텐서플로우 테스트 코드를 실행해 본다.



이상으로 환경 구성을 마친다.

# 5. 첫번째 모형 - 선형회귀분석

본격적인 머신러닝을 수행하기 전에 가장 단순하면서도 파워풀한 모형을 살펴보자.

바로 선형회식분석 모형이다.

회사에 월별 매출액 및 광고비 데이터가 있다고 가정하자.

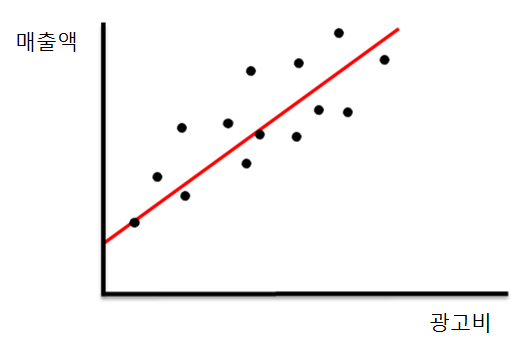
데이터를 분석해 보니 결과가 다음과 같다.

년 평균 매출액: 400억

년 평균 광고비: 5억

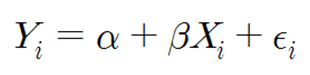
그런데 갑자기 여러 분에게 사장님이 다음과 같이 물어본다.

**"OO 과장! 올해 광고비 얼마써야 할까?"**

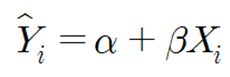


선형회귀분석 (Linear Regression Model)은 위와 같은 질문에 답할 수 있게 데이터를 설명하는 선(Line)을 찾는 모형이다.

데이터를 나타내는 식은 다음과 같다.



여기에서 데이터를 설명하는 선에 대한 식은



Y가 매출액, X가 광고비로 놓으면

**매출액 = 300 + 20 \* 광고비**

와 같은 식을 얻을 수 있다.

저 공식이 바로 데이터를 설명하는 모형이고, 분석가가 만들어야 하는 것이다.

일단 저와 같은 공식을 구했다는 가정에서 조금만 더 깊게 들어가 보자.

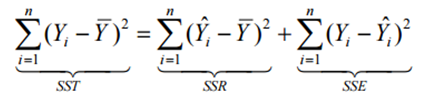
광고비 10억을 집행하면 예상 매출액은 500억이다.

그런데 실제 매출액이 600억이라면 예측을 100억이나 잘못한 것이다.

뭐야? 회귀분석이라고 별거 없네? 라고 할지도 모르겠다.

만약에 우리가 저 회귀식을 몰랐다고 가정하면 우리의 의사결정은 평균으로 밖에 하지 못한다.

년 평균 매출액이 400억이니까 올해는 평균 수준은 되지 않을까요? 400억이라고 예측할 수 밖에 없다.



해당 식을 통해 위의 상황을 정리해 보면

ST = 600억(실제) - 400억(평균) = 200억

SR = 500억(예상) - 400억(평균) = 100억

SE = 600억(실제) - 500억(예상) = 100억

여기에서 이해하면 좋은 것이 하나가 있는데, 회귀식을 몰랐다면 오차가 200억이 발생한 것이다.

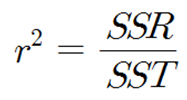
그런데 회귀식으로 인해 오차가 100억이 줄어든 것이다. 즉, 평균으로 의사결정할 때 보다 회귀식을 이용하면 오차를 50%나 줄일 수 있다.

거꾸로 얘기하면 50%나 더 잘 설명했다고 말할 수 있는 것이다.

그리고, ST = SR + SE 가 된다.

위의 공식 SST에서 ST를 제외한 맨 앞의 S는 Sum 을 가리킨다. 즉, 모든 경우의 수를 계산한 것이 SST이고, 저 위의 상황만 계산해 본게 ST이다.

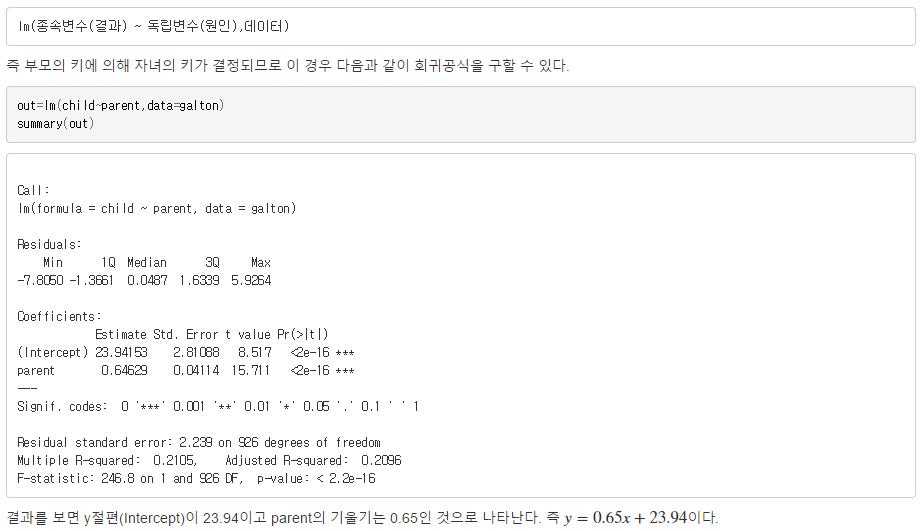
그래서 회귀분석에서는 결정계수 R2 (알 스퀘어)라는 것이 있는데



로 표현한다.

그래서 결정계수가 50이면 이 회귀식으로 50% 확률로 예측할 수 있다고 말할 수 있는 것이다.

그럼 지금까지 알고있는 이론을 가지고 R이라는 언어로 회귀분석을 해보자.

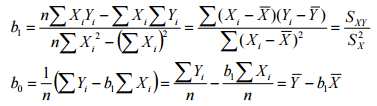


출처: <http://rstudio-pubs-static.s3.amazonaws.com/189354_277dfb3a83a34a2abaae855b90fcf269.html>

너무 쉽게 구할 수 있다. 이렇게 데이터가 있고, 도구의 사용법만 알면, 이런 작업은 쉽게 할 수 있는 것이다.

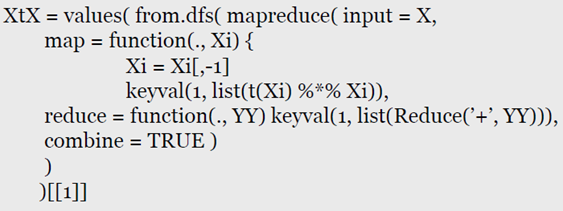
이제는 빅데이터로 생각해 보자. 3장에서 빅데이터로 생각할 때는 분산병렬처리와 MapReduce를 기억해야 한다고 했다.

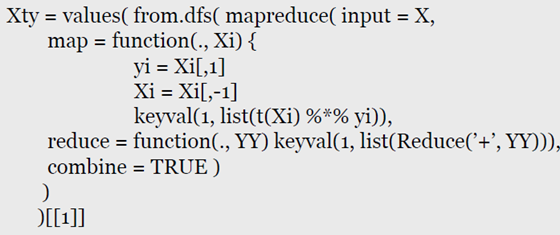
실제 수학으로 회귀식을 구하는 공식은 다음과 같다.



b1이 기울기 이고 b0가  y절편이 된다. b1을 구하면 b0는 평균을 이용해서 구하면 된다.

Hadoop에는 RHive라는 R을 지원하는 프레임워크가 있다. RHive를 이용한 회귀식 구하는 방법은 다음과 같다.





를 구한 다음 Solve(XtX, Xty)를 수행하면 b1을 구할 수 있다.

빅데이터가 아니면 lm이라는 함수 하나면 회귀분석이 끝나지만, 빅데이터로 처리하려니 뭔가 어렵고 복잡하다.

**Spark vs. Tensorflow**

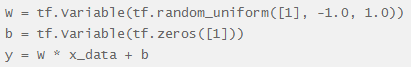
위의 선형 회귀분석을 Spark로 처리하면 다음과 같다.



Spark는 Hadoop 기반 언어이기 때문에 자연스럽게 빅데이터 처리를 지원한다.

하지만 Spark는 **Deep Neural Network를 지원하지 않는다.**

위의 선형 회귀분석을 Tensorflow로 처리하면 다음과 같다.

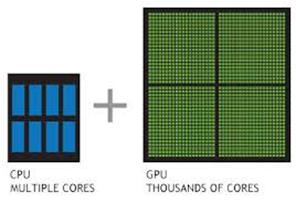


http://libqa.com/imageView?path=/resource/temp/10/2017061012327200148365099.png

http://libqa.com/imageView?path=/resource/temp/10/2017061012327205634309769.png

아쉽게도 아직 Tensorflow는 **Hadoop을 지원하지 않는다**. 하지만 Deep Neural Network를 모두 지원한다.

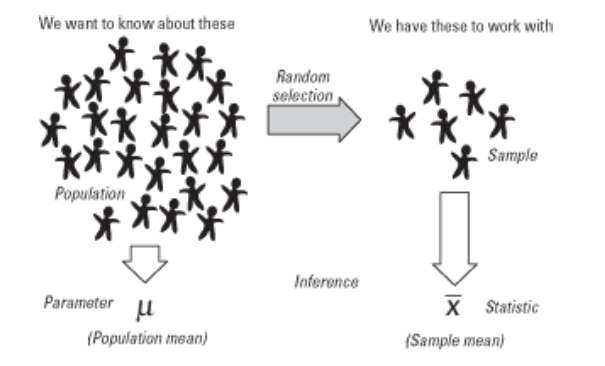
Spark와 Tensorflow 모두 GPU를 지원한다.



GPU는 빠르게 처리할 수 있을 뿐, 빅데이터를 지원하지 않는다. ㅜ\_ㅜ

빅데이터 시대에서 빅데이터(Hadoop 즉, 분산병렬처리 & MapReduce)를 지원하지 않으면 한계가 있는 건 아닐까?

그래서 우리는 공부를 해야한다. 통계학을 공부하면 빅데이터의 도구에서 자유로워 질수도!



그래서 이번 뜻밖의 모임 스터디 그룹에서는 Tensorflow로 머신러닝을 진행해 본다!!

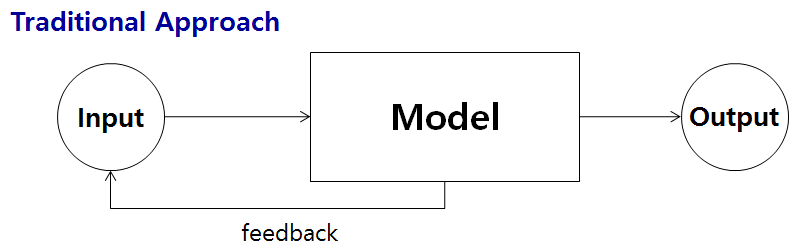
**6. Machine Learning이란**

머신러닝도 모형을 만드는 Data Analytics의 한 분야이다.

좀 더 자세히 설명하면,

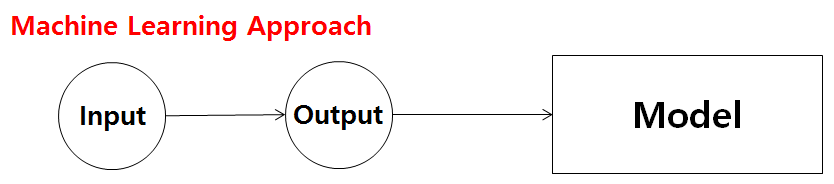
머신러닝 또는 기계학습은 인공지능의 한 분야로, 컴퓨터가 학습할 수 있도록 하는 알고리즘 과 기술을 개발하는 분야이다.

전통적인 모형을 만드는 방법은



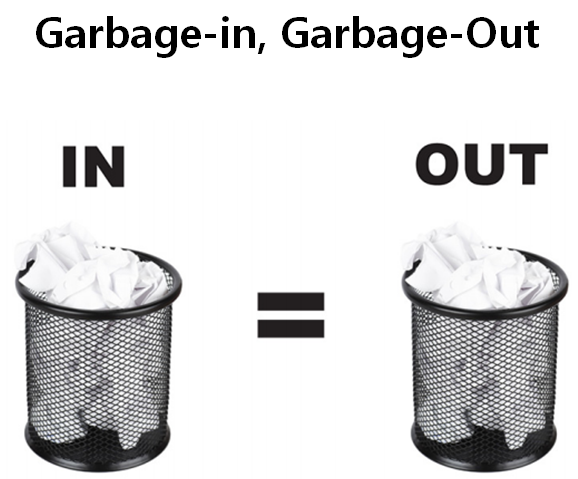
Input을 통해 모형을 만들고, 우리가 원하는 Output을 모형을 통해 얻는 방식이다.

머신러닝의 접근법은 다음과 같다.



선형회귀분석도 머신러닝의 방법이다. Input 광고비, Output 매출액을 통해 회귀식인 모형을 구하는 방식이기 때문이다.

머신러닝에서 제일 중요한 것은 데이터이다.



머신러닝의 장단점은 다음과 같다.

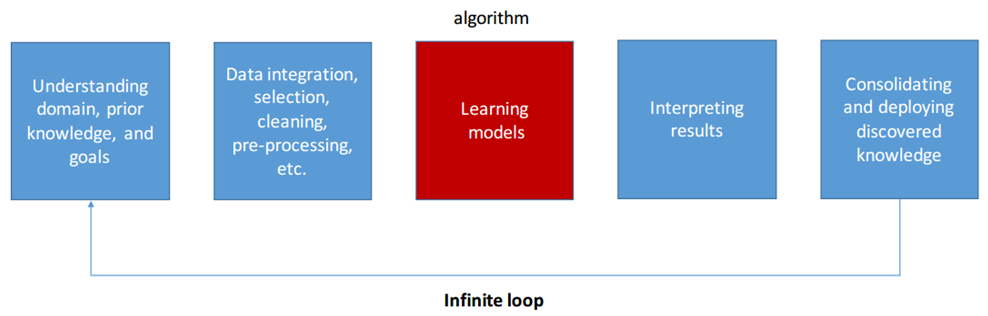
장점

* 학습을 위한 지식 표현이 필요 없음
* 사람이 만든 모델보다 좋은 결과를 보여줄 수 있음
* 고도의 수학적 지식이나 프로그래밍 능력을 요구하지 않음
* 자동화 가능
* 저렴하고 유연
* 프로그램을 이용해 원하는 대로 재사용

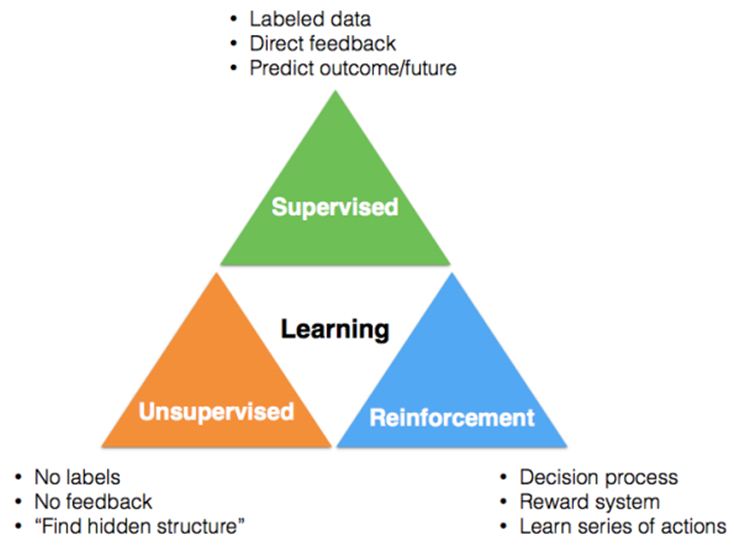
단점

* 데이터 준비에 많은 노력이 필요
* 오류 발생이 쉽다
* 생성된 모델이 블랙박스이기 때문에 해석하기 어렵다. (특히, 신경망)
* Over-fitting 문제 발생

머신러닝 프로세스



머신러닝의 종류



**7. 파이썬 Basic**

현재 공개된 텐서플로우의 예제들이 파이썬 언어로 작성되어 있기 때문에 파이썬을 이용해서 머신러닝을 공부하는 것이 좋다.

파이썬은 네덜란드의 개발자 Guido Van Rossum이 개발한 언어로 다른 언어와 비교해서 문법이 쉽고 간결하다.

동적언어이며 인터프리터 언어이다.

Hello World

a = "Hello World"  
print(a)  
print("Hello World!")

자료형은 다른 언어와 같다. int, float, bool, str 등

그러나 파이썬은 선언 과정이 없기 때문에 입력된 값에 맞춰 자료형을 판단한다.

a = 1  
b = 1.0  
sum = a + b  
print(a)  
print(b)  
print(sum)

또한 파이썬은 데이터 자료형이 다를 때 자동 형 변환이 된다.

**자료구조**

|  |  |  |  |
| --- | --- | --- | --- |
| 분류 | 자료형 | 설명 | 예 |
| 리스트 | list | 파이썬 객체들의 모음을 표현하는 자료형으로 '목록'처럼 순서 있는 값들을 나열한다. | [1,2,3,4], ['a','b','c','d'],[[1,2],[3,4]], [1,2,'a','b'] |
| 튜플 | tuple | 파이썬 객체들의 모음을 표현하는 자료형으로 순서있는 값들을 나열한다. 내용을 바꿀 수 없다. | ('a','b','c','d'). (1,2,(3,4)), (1,2,[3,4])  #괄호을 빼도 된다. |
| 딕셔너리 | dict | 파이썬 객체들의 모음을 표현하는 자료형으로써 순서가 없는 값들을 나열합니다.  '사전'처럼 미리 키(key)를 등록해 놓고 그 키를 통해 값을 이용할 수 있다. | {'one':1, 'two':2}, {1:1,2:2},  {'one':'un', 'two':'deux'}  #키는 고정된 값이라서 리스트와 사전이 올 수 없다. |
| 셋 | set | 집합을 표현할 수 있다. 순서가 없고 중복된 자료를 넣으면 안된다. | {'one','two'n'three','four'} |

**List 예제**

a = []  
b = [1, 2, 3]  
c = [4, 5, 6]  
d = [['a', 'b', 'c'], ['A', 'B', 'C']]  
  
print(b)  
print(b[0])  
print(b[1])  
print(b[-1])  
  
b[0] = 10  
print(b)  
print(b+c)  
print(d)  
print(d[0])  
print(d[1])  
print(d[0][1])  
print(d[1][1])

실행결과

[1, 2, 3]

1

2

3

[10, 2, 3]

[10, 2, 3, 4, 5, 6]

[['a', 'b', 'c'], ['A', 'B', 'C']]

['a', 'b', 'c']

['A', 'B', 'C']

b

B

**리스트 연산**

a = [1, 2] \* 5  
print(a)

실행결과

[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]

[1,2]가 5번 반복되는 결과가 나온다.

그래서

a = [1, 2]  
**for** i **in** range(2):  
 a[i] \*= 5  
print(a)

실행결과

[5, 10]

**리스트 슬라이싱**

a = [7, 9, 8, 5, 6]  
print(a[2:])  
print(a[:])  
print(a[2:4])

실행결과

[5, 10]  
[8, 5, 6]  
[7, 9, 8, 5, 6]  
[8, 5]

**리스트 슬라이싱2**

a = [7, 9, 8, 5, 6]  
print(a[::-1])  
print(a[::2])  
print(a[::3])

실행결과

[6, 5, 8, 9, 7]  
[7, 8, 6]  
[7, 5]

**리스트의 메소드**

a = [5, 2, 1, 4, 3]  
a.append(6)  
print(a)  
a.remove(1)  
print(a)  
**del** a[3]  
print(a)  
print(a.index(6))  
a.sort()  
print(a)  
a.sort(reverse=**True**)  
print(a)  
a.append(6)  
print(a)  
print(a.count(6))

실행결과

[5, 2, 1, 4, 3, 6]  
[5, 2, 4, 3, 6]  
[5, 2, 4, 6]  
3  
[2, 4, 5, 6]  
[6, 5, 4, 2]  
[6, 5, 4, 2, 6]  
2

**튜플의 인덱싱 및 슬라이싱**

a = (7, 9, 8, 5, 6)  
print(a[1])  
print(a[2:])

실행결과

9  
(8, 5, 6)

튜플과 리스트의 차이는 튜플은 변경이 안된다는 것!

즉, 리스트는 몇 개의 요소가 들어있는지 모르고 변경이 자주 일어날 때 사용되며,  튜플은 요소들이 몇 개가 있는지 아는 상황에서 그 요소의 위치(인덱스)가 중요할 때 사용한다.

**딕셔너리**

dict = {'A': 'attack', 'M': 'move', 'P': 'patrol'}  
print(dict)  
  
dict = {}  
dict['A'] = 'Apple'  
dict['B'] = 'Banana'  
dict['M'] = 'Melon'  
print(dict)

실행결과

{'M': 'move', 'A': 'attack', 'P': 'patrol'}  
{'M': 'Melon', 'A': 'Apple', 'B': 'Banana'}

**셋**

A = {'one', 'two', 'three'}  
B = { 'one', 'three', 'five'}  
print(A & B)  
print(A.intersection(B))  
print(A | B)  
print(A.union(B))  
print(A - B)  
print(B - A)  
print(A.difference(B))

실행결과

{'three', 'one'}  
{'three', 'one'}  
{'five', 'three', 'two', 'one'}  
{'five', 'three', 'two', 'one'}  
{'two'}  
{'five'}  
{'two'}

**자료구조 비교**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 변경 | 삭제 | 순서 | 중복 |
| 리스트 | O | O | O | O |
| 튜플 | X | X | O | O |
| 딕셔너리 | O | O | X | X |
| 집합 | O | O | X | X |

**제어문**

**조건문 if**

파이썬에서 조건문은 다른 언어와 비슷하지만 문장을 구분하는 방법이 인덴트(들여쓰기) 이다.

그리고, 조건식의 끝에 콜론(:)이 필요하다.

a = 10  
b = 20  
**if** a < b:  
 print(b)  
**else**:  
 print(a)

실행결과

20

**조건문 if...elif**

a = 20  
**if** a > 0:  
 print('plus')  
**elif** a < 0:  
 print('minus')  
**else**:  
 print('zero')

실행결과

plus

**in, not in 문**

lis = [1, 2, 3]  
print(1 **in** lis)  
print(1 **not in** lis)

실행결과

True  
False

**반복문 While**

num = 0  
**while** num < 10:  
 print(num)  
 num += 1

실행결과

0  
1  
2  
3  
4  
5  
6  
7  
8  
9

**break 사용**

num = 0  
**while True**:  
 print(num)  
 **if** num == 5:  
 **break** num += 1

실행결과

0  
1  
2  
3  
4  
5

**continue 사용**

num = 0  
**while** num < 10:  
 num += 1  
 **if** num % 2 == 0:  
 **continue** print(num)

실행결과

1  
3  
5  
7  
9

**반복문 for**

num = 10  
sum = 0  
  
**for** x **in** range(num):  
 sum += x  
print(x)

실행결과

9

**리스트 컴퓨리핸션(List Comprehension)**

listComprehension = [v **for** v **in** range(0, 30) **if** v % 10 == 0]  
listComprehension2 = [[v, x, v \* x] **for** v **in** range(1, 10) **for** x **in** range(1, 10)]  
print(listComprehension)  
print(listComprehension2)

실행결과

[0, 10, 20]  
[[1, 1, 1], [1, 2, 2], [1, 3, 3], [1, 4, 4], [1, 5, 5], [1, 6, 6], [1, 7, 7], [1, 8, 8], [1, 9, 9], [2, 1, 2], [2, 2, 4], [2, 3, 6], [2, 4, 8], [2, 5, 10], [2, 6, 12], [2, 7, 14], [2, 8, 16], [2, 9, 18], [3, 1, 3], [3, 2, 6], [3, 3, 9], [3, 4, 12], [3, 5, 15], [3, 6, 18], [3, 7, 21], [3, 8, 24], [3, 9, 27], [4, 1, 4], [4, 2, 8], [4, 3, 12], [4, 4, 16], [4, 5, 20], [4, 6, 24], [4, 7, 28], [4, 8, 32], [4, 9, 36], [5, 1, 5], [5, 2, 10], [5, 3, 15], [5, 4, 20], [5, 5, 25], [5, 6, 30], [5, 7, 35], [5, 8, 40], [5, 9, 45], [6, 1, 6], [6, 2, 12], [6, 3, 18], [6, 4, 24], [6, 5, 30], [6, 6, 36], [6, 7, 42], [6, 8, 48], [6, 9, 54], [7, 1, 7], [7, 2, 14], [7, 3, 21], [7, 4, 28], [7, 5, 35], [7, 6, 42], [7, 7, 49], [7, 8, 56], [7, 9, 63], [8, 1, 8], [8, 2, 16], [8, 3, 24], [8, 4, 32], [8, 5, 40], [8, 6, 48], [8, 7, 56], [8, 8, 64], [8, 9, 72], [9, 1, 9], [9, 2, 18], [9, 3, 27], [9, 4, 36], [9, 5, 45], [9, 6, 54], [9, 7, 63], [9, 8, 72], [9, 9, 81]]

**함수**

**사용자 정의 함수**

**def sum**(a, b):  
 **return** a + b  
  
a = 2000  
b = 17  
c = sum(a, b)  
print(c)

실행결과

2017

**매개변수가 유동적인 함수**

**def sum\_many**(\*args):  
 sum = 0  
 **for** i **in** args:  
 sum = sum + i  
 **return** sum  
result = sum\_many(2000, 10, 7)  
print(result)

실행결과

2017

**2개의 반환값을 가지는 함수**

**def sum\_and\_mul** (a, b):  
 **return** a+b, a\*b  
result = sum\_and\_mul(1008, 2)  
print(result)

실행결과

(1010, 2016)

**변수의 유효 범위**

a = 1  
  
**def test**(a):  
 a = a \* 1000  
 print(a)  
  
test(a)  
print(a)

실행결과

1000  
1

**전역변수 global**

a = 1  
**def test**():  
 **global** a  
 a = a\*1000  
 print(a)  
print(a)  
test()  
print(a)

실행결과

1  
1000  
1000

이상으로 파이썬 기초 부분을 마칩니다.

**8. 파이썬 Advanced**

Data Analytics에서 파이썬은 상당히 유용한 패키지들을 많이 제공한다.

지금부터 파이썬과 함께 쓰면 유용한 패키지들에 대해서 알아본다.

앞으로 진행하게 될 텐서플로우도 마찬가지다.

이번 장에서 살펴 볼 패키지들은 텐서플로우를 사용하면서도 많이 활용될 것이다.

**NUMPY**

Numerical Python의 줄임말로 다차원 배열인 ndarray(N-dimensional array)와 선형대수, 난수 발생 등의 수학적 계산에 많이 사용된다.

파이썬의 리스트 자료구조는 다양한 자료형을 저장할 수 있어 매유 유연하지만 메모리 점유 공간이 많고 속도가 느려지고, 프로그램 오류를 발생시키는 경우가 많다.

이러한 문제를 해결하기 위해 numpy의 ndarray를 사용한다.

**numpy의 ndarray 사용하기**

**import** numpy **as** np  
  
a = [1, 2, 3, 4, 5, 6]  
npa = np.array(a)  
print(type(a))  
print(type(npa))  
print(npa.dtype)  
  
b = [1.0, 2.0, 3.0, 4.0, 5.0]  
npb = np.array(b)  
print(npb.dtype)  
  
print(npa[0])  
npa[0] = 10  
print(a)  
  
npc = np.array([[1, 2], [3, 4]])  
print(npc[0, 0], npc[0, 1])

실행결과

<class 'list'>  
<class 'numpy.ndarray'>  
int32  
float64  
1  
[1, 2, 3, 4, 5, 6]  
1 2

numpy의 ndarray는 리스트를 사용하는 방법과 유사하게 사용할 수 있다.

**많이 쓰이는 함수**

**import** numpy **as** np  
  
print(np.zeros(5))  
print(np.zeros((3, 3)))  
print(np.ones(5))  
print(np.random.rand(2, 2))

실행결과

[ 0.  0.  0.  0.  0.]  
[[ 0.  0.  0.]  
 [ 0.  0.  0.]  
 [ 0.  0.  0.]]  
[ 1.  1.  1.  1.  1.]  
[[ 0.58613583  0.20208349]  
 [ 0.85597352  0.49402706]]

* zeros(): 0으로 초기화된 배열을 만든다.
* ones(): 1로 초기화된 배열을 만든다.
* random.rand(): 랜덤값으로 초기화화된 배열을 만든다.

**arange()와 linespace()**

**import** numpy **as** np  
  
a = np.arange(10)  
b = np.arange(3, 10, 2)  
c = np.linspace(0, 2, 5)  
  
print(a)  
print(b)  
print(c)

실행결과

[0 1 2 3 4 5 6 7 8 9]  
[3 5 7 9]  
[ 0.   0.5  1.   1.5  2. ]

파이썬의 rang()는 빌트인 함수로 리스트를 반환하고 arange()함수는 배열을 반환한다는 점이 다르다.

linespace()함수는 일정한 구역을 등간격으로 배열을 생성한다.

**arrary.shape & array.dtype**

**import** numpy **as** np  
  
npArr1 = np.array([1, 2, 3, 4, 5, 6])  
print(npArr1.shape)  
print(npArr1.dtype)  
  
npArr2 = np.array([[1, 2], [3, 4], [5, 6]])  
print(npArr2.shape)  
  
npArr3 = np.array([[[1], [2], [3]], [[4], [5], [6]]])  
print(npArr3.shape)

실행결과

(6,)  
int32  
(3, 2)  
(2, 3, 1)

npArr1은 1차원만 가지고 있고, 6개의 값을 가지고 있다는 뜻이고, npArr2는 1차원에 3개의 값, 2차원에 2개의 값을 가지고 있다는 뜻이다.

npArr3은 1차원에 2개, 2차원에 3개, 3차원에 1개의 값을 가지고 있다는 내용을 나타낸다.

**reshape()**

**import** numpy **as** np  
  
array1 = np.arange(6)  
print(array1)  
  
array2 = np.arange(6).reshape(2, 3)  
print(array2)  
  
array3 = np.arange(12).reshape(2, 2, 3)  
print(array3)

실행결과

[0 1 2 3 4 5]  
[[0 1 2]  
 [3 4 5]]  
[[[ 0  1  2]  
  [ 3  4  5]]  
 [[ 6  7  8]  
  [ 9 10 11]]]

reshape()함수는 배열의 차원을 변경할 수 있는 함수이다.

**ndarray의 연산**

**import** numpy **as** np  
  
data1 = np.array([1, 2, 3, 4, 5])  
data2 = np.array([10, 20, 30, 40, 50])  
  
print(data1 \* 10)  
print(data1 \* data2)  
  
print(data1 + 2)  
print(data1 + data2)  
  
print(data1 \*\* 2)

실행결과

[10 20 30 40 50]  
[ 10  40  90 160 250]  
[3 4 5 6 7]  
[11 22 33 44 55]  
[ 1  4  9 16 25]

nparray끼리의 사칙연산은 각 항목끼리 이뤄진다. 빌트인 자료형인 리스트와는 다르게 연산되는 점에 주목할 필요가 있다.

**브로드캐스팅(broadcasting)**

**import** numpy **as** np  
  
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])  
print(x.shape)  
  
v = np.array([10, 20, 30])  
print(v.shape)  
  
y = x + v  
print(y)

실행결과

(4, 3)  
(3,)  
[[11 22 33]  
 [14 25 36]  
 [17 28 39]  
 [20 31 42]]

브로드캐스팅은 numpy에서 shape이 다른 배열사이에서 산술 연산이 가능하게 해주는 기능이다.

브로드캐스팅 규칙은 다음과 같다.

행렬A와 벡터V의 연산에서 벡터V의 크기가 행렬A의 행또는 열과 동일한 경우

* m\*n 행렬과 m\*1 벡터 간의 연산
* m\*n 행령과 1\*n 벡터 간의 연산

열벡터와 행벡터간의 연산

* m\*1 벡터와 1\*n 벡터 간의 연산

이러한 경우 **한 쪽의 크기가 다른 쪽의 크기로 확장된 후** 요소간 연산을 수행한다.

즉, **비교 대상인 두 개의 차원이 서로 같은 값이거나, 2개 중에 1개가 1이라면 두 개의 차원이 동일하다고 간주한다**

그리고, **연산 결과 배열의 차원은 최대 크기**가 된다.

**import** numpy **as** np  
  
a = np.arange(12).reshape(4, 3)  
print(a)  
  
b = np.array([0, 1, 2])  
print(b)  
  
print(a + b)  
  
b = np.array([0, 1, 2, 3]).reshape(4, 1)  
print(b)  
print(a + b)

실행결과

[[ 0  1  2]  
 [ 3  4  5]  
 [ 6  7  8]  
 [ 9 10 11]]  
[0 1 2]  
[[ 0  2  4]  
 [ 3  5  7]  
 [ 6  8 10]  
 [ 9 11 13]]  
[[0]  
 [1]  
 [2]  
 [3]]  
[[ 0  1  2]  
 [ 4  5  6]  
 [ 8  9 10]  
 [12 13 14]]

**sum()**

**import** numpy **as** np  
  
x = np.array([[1, 2], [3, 4]])  
  
print(np.sum(x))  
print(np.sum(x, axis=0))  
print(np.sum(x, axis=1))

실행결과

10  
[4 6]  
[3 7]

**ndarray의 슬라이싱**

**import** numpy **as** np  
  
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])  
print(a)  
  
b = a[:2, 1:3]  
print(b)  
  
print(a[0, 1])  
b[0, 0] = 20  
print(a[0, 1])

실행결과

[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]]  
[[2 3]  
 [6 7]]  
2  
20

이러한 numpy에서의 슬라이싱은 텐서플로우에서 빈번하게 사용된다.

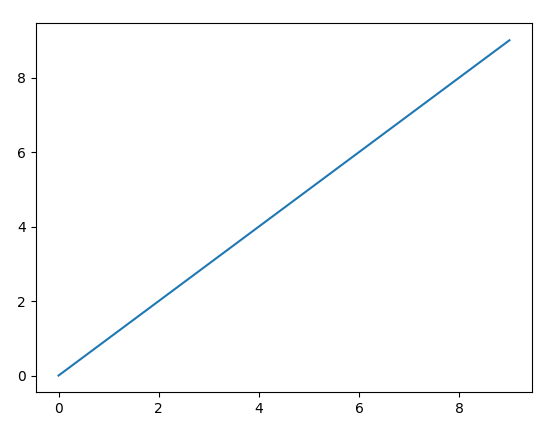
**MATPLOTLIB**

matplotlib는 그래프를 그려주는 파이썬의 시각화 라이브러리이다.

**직선 그리기**

**import** matplotlib.pyplot **as** plt  
  
x = range(10)  
y = range(10)  
  
plt.plot(x, y)  
plt.show()

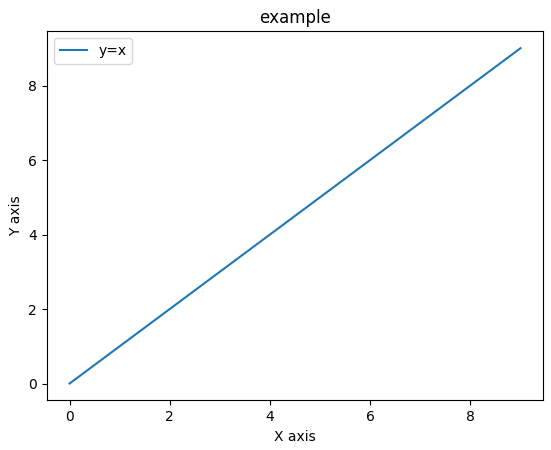
실행결과



**그래프 꾸미기**

**import** matplotlib.pyplot **as** plt  
  
x = range(10)  
y = range(10)  
  
plt.plot(x, y, label="y=x")  
plt.title("example")  
plt.xlabel("X axis")  
plt.ylabel("Y axis")  
plt.legend()  
plt.show()

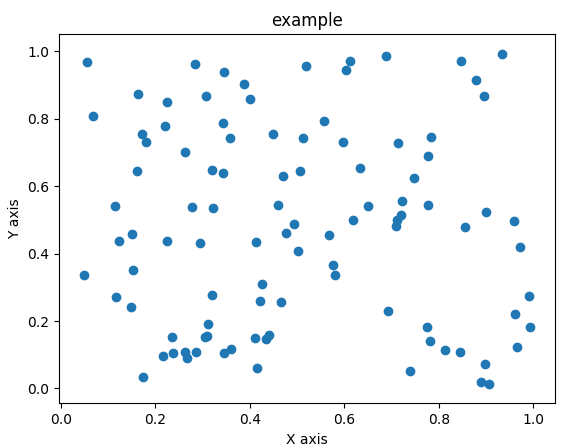
실행결과



**산포도 그래프**

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
x = np.random.rand(100)  
y = np.random.rand(100)  
  
plt.scatter(x, y)  
plt.title("example")  
plt.xlabel("X axis")  
plt.ylabel("Y axis")  
plt.show()

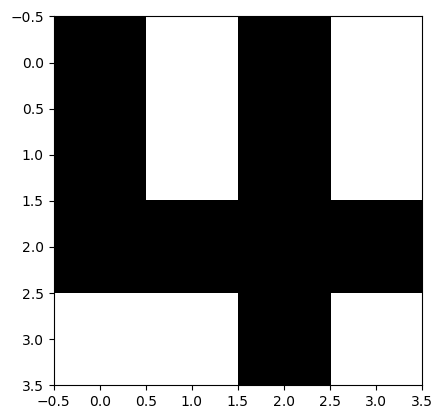
실행결과



**imshow() 함수**

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
a = [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0]  
np1 = np.array(a)  
plt.imshow(np1.reshape(4, 4), cmap='Greys', interpolation='nearest')  
plt.show()

실행결과



imshow() 함수는 배열의 데이터를 이미지로 렌더링하는 함수입니다.

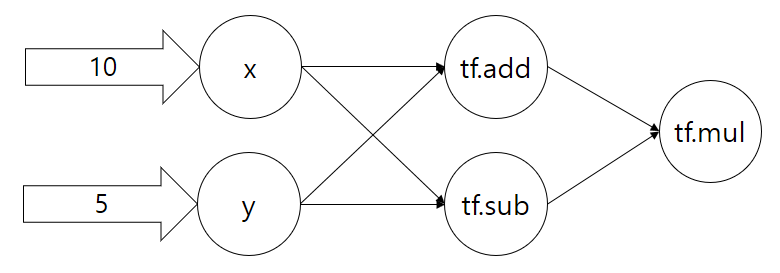
결론은

**9. 텐서플로우 Basic**

지금부터 텐서플로우에 대해서 알아보자



텐서플로우는 모든 프로그램이 계산 그래프 (computational graph)를 통해 구성되며, 이 계산 그래프가 CPU, GPU 등의 연산장치에서 작동된다.



계산 그래프란 프로그램의 계산 구조를 정의하는 방향성 그래프 (directional graph)를 말한다.

계산 그래프는 노드 (node)와 엣지 (edge)로 구성된다. 노드는 계산 그래프에서 값들의 계산하는 등의 작업을 나타낸다.

텐서플로우에서 노드는 오퍼레이션(OP)라고 부른다. 그리고 실제 값을 전달하는 것이 엣지이다. 엣지는 화살표로 표현된다.

텐서플로우의 계산 그래프는 엣지를 따라 텐서(tensor)라는 형태로 값이 다른 오퍼레이션으로 이동한다고 말한다.

그래서 언어 이름이 텐서(tensor)의 흐름(flow)이다.

텐서는 데이터를 표현하는 기본 단위로 다차원(multi-dimensional) 행렬을 의미한다.

**Session**

텐서는 Spark의 RDD(Resilient Distributed Dataset)과 개념이 비슷하다.

공통점은 정의되었을 때 실행되는게 아니라 실제 연산을 수행할 때 실행한다.

**import** tensorflow **as** tf  
  
x = tf.constant(8)  
y = tf.Variable(x + 17)  
print(y)  
  
init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
print(sess.run(y))

실행결과

<tf.Variable 'Variable:0' shape=() dtype=int32\_ref>  
25

tf.global\_variables\_initializer()는 세션 속의 tf.Variable 형태로 저장한 변수를 초기화 해주는 기능을 수행한다.

텐서플로우는 세션을 기준으로 성능을 향상시키기 위해 정의와 실행을 분리하였다.

파이썬은 다른 프로그램 언어에 비해 실행속도가 느린 단점이 있다. 이를 극복하기 위해 외부에서 연산을 수행하는 numpy와 같은 라이브러리를 사용하지만 외부에서 연산한 값이 파이썬으로 전환될 때 오버헤드가 발생하는 문제가 생깁니다.

텐서플로우는 이런 오버헤드를 피하기 위해 세션 이전에는 파이썬에서 계산 그래프를 그리고 세션을 통해 계산 그래프를 CPU와 GPU가 처리할 수 있는 다른 언어로 변환하여 연산하는 방법으로 속도 문제를 극복한 것이다.

**텐서플로우 자료형**

|  |  |  |
| --- | --- | --- |
| 텐서플로우 자료형 | 파이썬 자료형 | 설명 |
| DT\_FLOAT | tf.float32 | 32 비트 실수형 |
| DT\_DOUBLE | tf.float64 | 64 비트 실수형 |
| DT\_INT8 | tf.int8 | 8 비트 정수형 |
| DT\_INT16 | tf.int16 | 16 비트 정수형 |
| DT\_INT32 | tf.int32 | 32 비트 정수형 |
| DT\_INT64 | tf.int64 | 64 비트 정수형 |
| DT\_UINT8 | tf.uint8 | 8 비트 부호 없는 정수형 |
| DT\_UINT16 | tf.uint16 | 16 비트 부호 없는 정수형 |
| DT\_STRING | tf.string | 가변 길이 문자열 배열 |
| DT\_BOOL | tf.bool | boolean |
| DT\_COMPLEX64 | tf.complex64 | 32 비트 실수형과 복소수의 조합 |
| DT\_COMPLEX128 | tf.complex128 | 64 비트 실수형과 복소수의 조합 |
| DT\_QINT8 | tf.qint8 | 양자화된 오퍼레이션 안에서 사용되는 8비트 부호 있는 정수형 |
| DT\_QINT32 | tf.qint32 | 양자화된 오퍼레이션 안에서 사용되는 32비트 부호 있는 정수형 |
| DTQUINT8 | tf.quint8 | 양자화된 오퍼레이션 안에서 사용되는 8비트 부호 없는 정수형 |

**상수**

상수는 변경할 수 없는 고정된 값을 말하며 tf.constant()를 통해 생성한다.

**import** tensorflow **as** tf  
  
a = tf.constant([1, 2, 3, 4, 5, 6, 7, 8, 9])  
b = tf.constant([[[1, 1], [2, 2]], [[3, 3], [4, 4]]])  
  
init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
print(sess.run(tf.reshape(a, [3, 3])))  
print(sess.run(tf.reshape(b, [2, 4])))  
print(sess.run(tf.reshape(b, [-1])))

실행결과

[[1 2 3]  
 [4 5 6]  
 [7 8 9]]  
[[1 1 2 2]  
 [3 3 4 4]]  
[1 1 2 2 3 3 4 4]

**행렬의 곱셉**

**import** tensorflow **as** tf  
  
a = tf.constant([[1., 2.]])  
b = tf.constant([[3.], [4.]])  
  
product = tf.matmul(a, b)  
init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
result = sess.run(product)  
print(result)  
print(result.shape)  
sess.close()

실행결과

[[ 11.]]  
(1, 1)

**변수**

변수는 tf.Variable()을 사용해서 만든다.

만약 세션 속에서 변수의 값을 변경하고 싶다면 tf.assign() 함수를 이용한다.

**import** tensorflow **as** tf  
  
state = tf.Variable(0)  
one = tf.constant(1)  
  
new\_value = tf.add(state, one)  
update = tf.assign(state, new\_value)  
  
init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
print(sess.run(state))  
**for** \_ **in** range(5):  
 sess.run(update)  
 print(sess.run(state))  
sess.close()

실행결과

0  
1  
2  
3  
4  
5

**Placeholder**

텐서플로우의 또 다른 데이터 저장형이다. Placeholder는 초기값을 입력하지 않고 정의한다는 특징이 있다.

세션 속에서 함수처럼 원하는 값을 넣고 실행할 수 있다는 장점을 가지고 있다.

tf.placeholder()는 데이터타입과 feed를 통해 데이터 모양을 정의한다.

**import** tensorflow **as** tf  
  
x = tf.placeholder(tf.float32, [2])  
  
init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
print(sess.run(x, feed\_dict={x: [1, 3]}))  
sess.close()

실행결과

[ 1.  3.]

**import** tensorflow **as** tf  
  
x = tf.placeholder(tf.float32)  
y = tf.placeholder(tf.float32)  
  
output = tf.multiply(x, y)  
  
init = tf.global\_variables\_initializer()  
sess = tf.Session()  
sess.run(init)  
print(sess.run(output, feed\_dict={x: [10.], y: [20.]}))  
sess.close()

실행결과

[ 200.]

**텐서플로우 기본 함수**

|  |  |
| --- | --- |
| 함수 | 설명 |
| tf.add(x, y) | x값과 y값의 덧셈을 반환한다.  tf.add(2000, 17) |
| tf.subtract(x, y) | x값고 y값의 뺄셈을 반환한다.  tf.subtract(20, 17) |
| tf.multiply(x, y) | x값과 y값의 곱셈 값을 반환한다. |
| tf.divide(x, y) | x값과 y값의 나눗셈의 몫을 반환한다. |
| tf.abs(x) | x의 절대값을 반환한다. |
| tf.squre(x) | x의 제곱을 반환한다. |
| tf.less(x, y) | x의 값이 y의 값보다 크면 true를, 적다면 false를 반환한다. |
| tf.sqrt(x) | x의 제곱근을 반환한다. |
| tf.assign(x, y) | y값을 x값에 대입하여 x값을 업데이트 한다. |
| tf.transpose(a, perm=none) | perm에 따라 a 변수들의 차원을 변경한다.  a = [[1,2],[3,4]]  tf.transpose(a, perm=[1,0]) #[[1,3],[2,4]] |
| tf.random\_uniform(shape, minval, maxval, dtype, seed) | 정규분포를 따르는 난수를 생성한다.  tf.random\_uniform([1],0,1) |
| tf.maximum(x, y) | x, y중 최대값을 리턴한다. |
| tf.minimum(x, y) | x, y중 최소값을 리턴한다. |
| tf.reduce\_mean(x) | x의 평균값을 구한다.  x = tf.Variable([[1, 2, 3], [4, 5, 6]])  y = tf.reduce\_mean(x) |
| tf.reduce\_sum(x) | x의 합을 구한다.  x = tf.Variable([[1, 2, 3], [4, 5, 6]])  y = tf.reduce\_sum(x) |

이상으로

# 10. 지도학습

이제 부터 본격적으로 텐서플로우를 이용한 머신러닝 학습을 시작한다.

지도학습(Supervised Learning)이란 **정답이 알려진 학습데이터**로 부터 어떤 일관된 규칙을 찾아가는 것을 말한다.

지도학습은 세부적으로 회귀(Regression)와 분류(Classification)가 있다.

회귀는 우리가 첫번째 모형으로 이미 알고 있다. 하지만 텐서플로우로 어떻게 처리하면 되는지 자세히 알아보자.

Linear Regression Model

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** tensorflow **as** tf  
  
num\_points = 200  
vectors\_set = []  
**for** i **in** range(num\_points):  
 x = np.random.normal(5, 5) + 15  
 y = x \* 1000 + (np.random.normal(0, 3)) \* 1000  
 vectors\_set.append([x, y])  
  
x\_data = [v[0] **for** v **in** vectors\_set]  
y\_data = [v[1] **for** v **in** vectors\_set]  
  
plt.plot(x\_data, y\_data, label="ro")  
plt.ylim([0, 40000])  
plt.xlim([0, 35])  
plt.xlabel('x')  
plt.ylabel('y')  
plt.legend()  
plt.show()  
  
  
W = tf.Variable(tf.random\_uniform([1], -1.0, 1.0))  
b = tf.Variable(tf.zeros([1]))  
y = W \* x\_data + b  
  
loss = tf.reduce\_mean(tf.square(y - y\_data))  
optimizer = tf.train.GradientDescentOptimizer(0.0015)  
train = optimizer.minimize(loss)  
  
init = tf.initialize\_all\_variables()  
sess = tf.Session()  
sess.run(init)  
  
**for** step **in** range(10):  
 sess.run(train)  
 print(step, sess.run(W), sess.run(b))  
 print(step, sess.run(loss))  
  
 plt.plot(x\_data, y\_data, label='ro')  
 plt.plot(x\_data, sess.run(W) \* x\_data + sess.run(b))  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend()  
 plt.show()

실행결과

0 [ 1292.52966309] [ 60.23651505]  
0 4.75884e+07  
1 [ 906.6595459] [ 42.19635773]  
1 1.27082e+07  
2 [ 1021.94250488] [ 47.54182434]  
2 9.595e+06  
3 [ 987.50317383] [ 45.90071487]  
3 9.31713e+06  
4 [ 997.79418945] [ 46.3468895]  
4 9.29233e+06  
5 [ 994.72167969] [ 46.16947937]  
5 9.29011e+06  
6 [ 995.6416626] [ 46.17838287]  
6 9.28991e+06  
7 [ 995.36895752] [ 46.13163376]  
7 9.2899e+06  
8 [ 995.45245361] [ 46.10151291]  
8 9.28989e+06  
9 [ 995.42956543] [ 46.06643295]  
9 9.28989e+06

이상으로

**11. 비지도학습**

| **작성자** | **sangmin** |
| --- | --- |
| 작성일 | 2017.06.11 00:01:00 |
| 수정자 | **sangmin** |
| 수정일 | 2017.06.11 00:01:00 |
| 조회수 | 4 |
| 추천수 | 0 |
| 글상태 | 기본 |

# 12. 인공 신경망

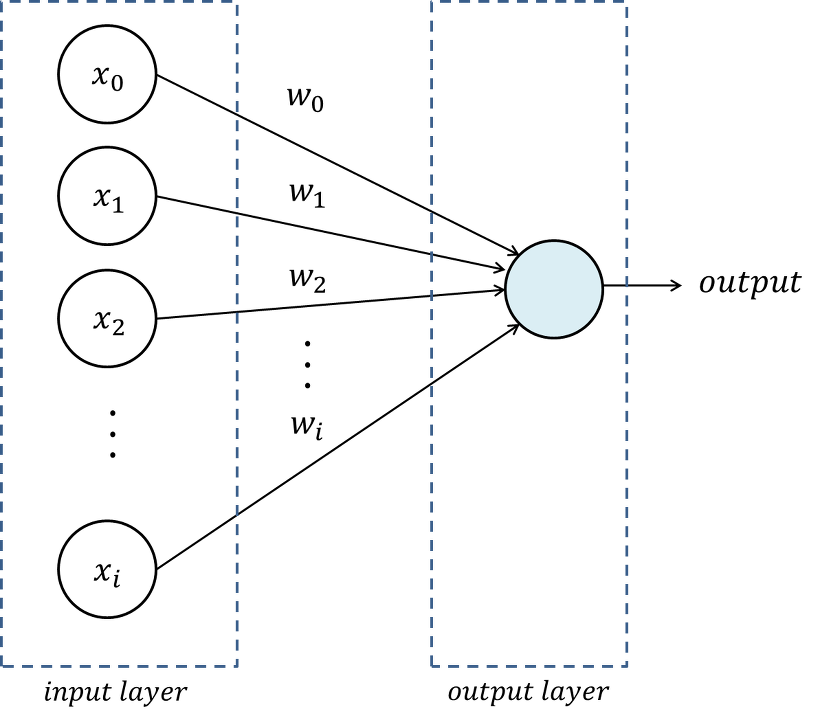
인공 신경망 (Artificial Neural Network: ANN)은 인간의 뇌 구조를 착안하여 만든 학습 기법이다.

수백억 개의 신경 세포인 뉴런들과 수 만개이 시냅스들로 서로 연결된 인간의 뇌 구조를 흉내 낸 구조이지만 많은 분야에서 만족할 만한 성능을 내고 있다.

인공 신경망의 역사에 대해서는 생략하고

가장 간단한 신경망인 퍼셉트론(Perceptron)부터 살펴보자.

**퍼셉트론: 가장 간단한 신경망**



위의 그림과 같이 입력값에 가중치를 곱한 값을 활성화 함수(activation function)을 통과시켜 최종 결과값을 생성하는 것이 신경망의 구조이다.

output = f(NET - T), NET = SUM(Xi \* Wi) 로 표현할 수 있다.