**S/W멤버십 과제 완료문서**

**과 제 명 : DrawML**

* 소 속 : 신촌 소프트웨어 멤버십
* 작 성 자 : 이상봉(PL), 구한모,

김영찬, 권태국

* 작 성 일 : 2016. 10. 9 (일)

**S/W멤버십 과제 완료문서 초안**

|  |  |  |  |
| --- | --- | --- | --- |
| **과 제 명** | DrawML | | |
| **과제구분** | 창 의 과 제 | | |
| **과제기간** | 2016.07 ~ 2016.09 (3개월) | | |
| **지 역** | 신촌 멤버십 | **참여인원** | 4명 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **회원명** | **학 교** | **학 과** | **학년** | **연 락 처** | **E-mail** |
| 이상봉 | 인하대 | 전자공학과 | 4 | XXXX | gnob244@gmail.com |
| 구한모 | 항공대 | 소프트웨어학과 | 3 | XXXX | rngksah@naver.com |
| 김영찬 | 홍익대 | 컴퓨터공학과 | 4 | XXXX | ovekyc@gmail.com |
| 권태국 | 서강대 | 컴퓨터공학과 | 3 | XXXX | xornrbboy@naver.com |

|  |  |  |  |
| --- | --- | --- | --- |
| **구 분** | **내 용** | | |
| **개발목적**  **및 동기** | 현재 급속도로 발전 중인 Machine Learning을 이용 하여, ML의 지식이 있는 사람들이 편리하게 코드를 작성하지 않고 웹 컴포넌트를 통해서 ML을 구현 할 수 있는 프로그램을 개발   * 웹 컴포넌트 기반 ML 개발 | | |
| **개발환경**  **및 일정** | [개발 환경]   |  |  | | --- | --- | | 구 분 | 내 용 | | OS | Microsoft Windows 7/10, Ubuntu 16.04LTS, Mac OS X – El capitan | | Tool | PyCharm, Vim, Gcc, etc… | | Language | Python3, HTML5, Javascript, Bash script |   [개발 일정]   |  |  | | --- | --- | | 구 분 | 내 용 | | 7월 | 자료조사 및 전체 시스템 설계, ML 컴포넌트 화 및 XML 포멧 설계 | | 8월 | 프론트엔드, WAS 개발 시작, 분산 Task 시스템 프로토타이핑 | | 9월 | 분산 시스템 개발, Cloud File System 개발, 통합테스트 | | | |
| **창의성/**  **우수성** | 기존의 Microsoft Azure와 같은 웹 기반의 Machine Learning 서비스가 존재를 하지만, 전문가에 초점이 맞추어져 있어 초보자는 사용하기가 어려운 것에 반해, 본 프로젝트는 좀더 직관적인 컴포넌트의 사용과 오픈소스로 공개를 하여 특징을 살릴 예정이다. | | |
| **활용성/**  **사업성** | 추후 기업의 내부 ML인프라 구축 및 교육적 목적으로의 활용이 가능하다. | | |
| **지원부서** | S/W멤버십 | **기술지원 연구원** | XXXX |
| **전화번호** | XXXX | **E-Mail** | XXXX |

**목 차**

**- 내용 목차 -**

[1. 선정 배경 및 개발 목적 4](#_Toc462502998)

[가. Machine Learning의 수요 증가 4](#_Toc462502999)

[나. Machine Learning 코드의 불편함 4](#_Toc462503000)

[다. Cloud Machine Learning 5](#_Toc462503001)

[2. 개발 목표 7](#_Toc462503002)

[가. Web을 통해 ML 모델을 제작하고 Experiment를 수행할 수 있는 웹 어플리케이션을 만든다. 7](#_Toc462503003)

[나. 데이터 처리, ML 알고리즘 등을 분류 및 컴포넌트 화 하여 사용자에게 제공한다. 7](#_Toc462503004)

[다. 사용자가 Web을 통해 작성한 Graph형태의 모델 및 Experiment를 XML형태로 변환하여 저장하고, tensorflow 코드로 변환하여 실행한다. 7](#_Toc462503005)

[라. 동시에 여러 개의 Experiment를 실행할 수 있도록 여러 대의 컴퓨터를 활용하여 여러 개의 Experiment를 실행 할 수 있는 분산 처리 시스템을 구현한다. 7](#_Toc462503006)

[3. 제약 사항 8](#_Toc462503007)

[가. 미리 우리가 정한 핵심 ML 알고리즘들로 기능을 제한한다. 8](#_Toc462503008)

[나. 컴퓨팅 자원에 한계가 있으므로 동시에 너무 많은 ML이나 규모가 큰 ML은 처리할 수 없다. 8](#_Toc462503009)

[4. 개발 내용 9](#_Toc462503010)

[가. System Architecture 9](#_Toc462503011)

[나. Distributed Task System (DTS) 11](#_Toc462503012)

[다. Experiment Realization & Execution Flow 14](#_Toc462503013)

[라. ML 컴포넌트 화 15](#_Toc462503014)

[마. Web UI 26](#_Toc462503015)

[5. 향후 발전 방향 29](#_Toc462503016)

[가. ML 컴포넌트의 추가 & 자유도 향상 29](#_Toc462503017)

[나. 프로젝트의 오픈소스화 29](#_Toc462503018)

[다. Distributed Task System의 향상 29](#_Toc462503019)

[6. 개발 환경 30](#_Toc462503020)

[7. 항목별 가중치 31](#_Toc462503021)

[8. 개발 일정 32](#_Toc462503022)

**- 그림 목차 -**

[❚그림 1 TensorFlow 1](#_Toc455677116)

[❚그림 2 Neural networks의 예 2](#_Toc455677117)

# 선정 배경 및 개발 목적

## Machine Learning의 수요 증가

###### Machine Learning(이하 ML)의 수요는 이론적인 발전, 하드웨어의 발전, 및 최근 Alphago의 유명세를 통해서 ML을 학습 및 활용 하려는 수요가 급격하게 증가하였다. 이에 따라서 ML을 위한 여러가지 서비스 및 라이브러리 또한 급격한 증가를 하고 있다.

###### https://upload.wikimedia.org/wikipedia/en/7/74/TensorFlow.png

❚그림 1 TensorFlow

###### 대표적으로 잘 알려진 Google의 TensorFlow가 있고, 이외에도 Microsoft의 Azure 등 여러 가지 서비스 및 라이브러리가 있으며, Coursera와 같은 교육사이트에서도 ML학습을 위한 여러가지 강의를 제공해 주고 있으며, 이외에도 여러 커뮤니티에서 ML의 수요는 급격히 증가하는 추세이다.

## Machine Learning 코드의 불편함

###### ML이 [그림 1]의 TensorFlow와 같은 오픈소스 라이브러리를 사용하더라도 결국 이론에 입각한 모델을 우리들의 데이터로 학습 시키는 것은 코드를 작성하지 않고서 불가능한 일이다. Python과 TensorFlow를 이용하는 것이 다른 프로그래밍 언어를 이용하여서 작성하는 것보다는 상당히 직관적이라는 것은 부정할 수 없는 사실이나, 결국 복잡한 네트워크를 형성하게 된다면 자신이 작성한 코드라 할 지라도 한 눈에 전체 flow를 파악한 다는 것은 매우 어렵다는 것은 인정 할 수 밖에 없는 사실일 것이다.

###### http://neuralnetworksanddeeplearning.com/images/tikz12.png

❚그림 2 Neural networks의 예

###### [그림 2]는 어느 정도 수의 신경망을 그림으로 나타나져 있는 것이다. 실제 전문적인 용도로 사용을 하게 될 경우에는 이것보다도 훨씬 더 복잡한 신경망이 형성된다. 간단한 형태의 신경망을 코드로 작성을 할 경우에는 작성도 어렵지 않고, 실제 코드를 보고 다시 [그림 2]와 같은 형태로 역 변환 하는 것 또한 어렵지 않을 것이다. 하지만 실제 사용할만한 성능을 낼 수 있는 신경망의 경우 [그림 2]정도만 생각하여도 코드로 짜는 것 또한 어느 정도 걸리며, 이 코드를 보고 다시 신경망을 그림으로 나타내는 것은 더더욱 어려워 지게 될 것이다. 즉 신경망을 모델링 하는 과정에서는 위와 같은 그림으로 모델링이 되어 사람은 그림으로 이해를 하고 컴퓨터는 코드로 이해를 하게 되는데, 사용자가 직접 코드로 변환을 하지 않고 모델링 한 신경망 자체를 사용 할 수 있다면 훨씬 더 높은 효율을 통해 ML을 개발 할 수 있을 것이다.

## Cloud Machine Learning

###### Microsoft Azure, Google Cloud Machine Learning 등 많은 기업들이 ML을 위한 cloud service를 제공한다. Cloud 환경은 높은 computing power와 용이한 접근으로 널리 사용되고 있으며, 질 높은 서비스 또한 이용 이유 중 하나이다.

###### http://insightextractor.com/wp-content/uploads/2016/05/AzureML-with-name.pnghttps://3.bp.blogspot.com/-ySmp6NANwB4/VvLGkyVAQJI/AAAAAAAAA8c/iMeTQoMEM70YwLuUpqpYN100L85506N7Q/s320/image00.png

###### Google의 cloud machine learning은 학습된 모델 및 분석도구를 제공하여, 높은 접근성과 서비스의 질을 나타낸다. Azure의 가장 큰 강점은 visualized machine learning이다. 이 밖에도 많은 기업이 cloud환경에서 machine learning을 위한 서비스를 제공하지만, 값이 높은 유료이다. 또한, 대부분 사전 지식을 요구하는 서비스이기 때문에 ML을 필요로 하지만 분야 지식이 없는 사람은 이용하기 힘들다.

###### 이에 본 프로젝트는 tensorflow를 활용하여 cloud환경에서 visualized graph로 ML을 할 수 있도록 서비스를 제공 할 것이다.

# 개발 목표

## Web을 통해 ML 모델을 제작하고 Experiment를 수행할 수 있는 웹 어플리케이션을 만든다.

## 데이터 처리, ML 알고리즘 등을 분류 및 컴포넌트 화 하여 사용자에게 제공한다.

## 사용자가 Web을 통해 작성한 Graph형태의 모델 및 Experiment를 XML형태로 변환하여 저장하고, tensorflow 코드로 변환하여 실행한다.

## 동시에 여러 개의 Experiment를 실행할 수 있도록 여러 대의 컴퓨터를 활용하여 여러 개의 Experiment를 실행 할 수 있는 분산 처리 시스템을 구현한다.

# 제약 사항

## 미리 우리가 정한 핵심 ML 알고리즘들로 기능을 제한한다.

## 컴퓨팅 자원에 한계가 있으므로 동시에 너무 많은 ML이나 규모가 큰 ML은 처리할 수 없다.

# 개발 내용

## System Architecture

###### 

### WAS (Web Application Server)

WAS는 웹 어플리케이션에 필요한 주요 로직을 담당한다. User Manager는 사용자에 대한 로직을 담당하며 회원가입, 로그인, 로그아웃 등을 요청에 따라 처리하게 된다. Experiment Manager는 실험 페이지에 진입 시 사용자의 실험 데이터를 제공하고, 실험을 실행, 저장, 삭제하는 기능을 제공하게 된다. 비동기로 수행되어야 하는 작업들(코드 실행, 데이터 전처리 등)은 TaskRunner를 이용해 요청하게 된다. Data Manager는 Trained Model이나 Training Set, Experiment Code 같은 데이터를 관리하게 된다. 데이터들은 Mysql에 저장되며, WAS는 Python Flask로 구현하였다.

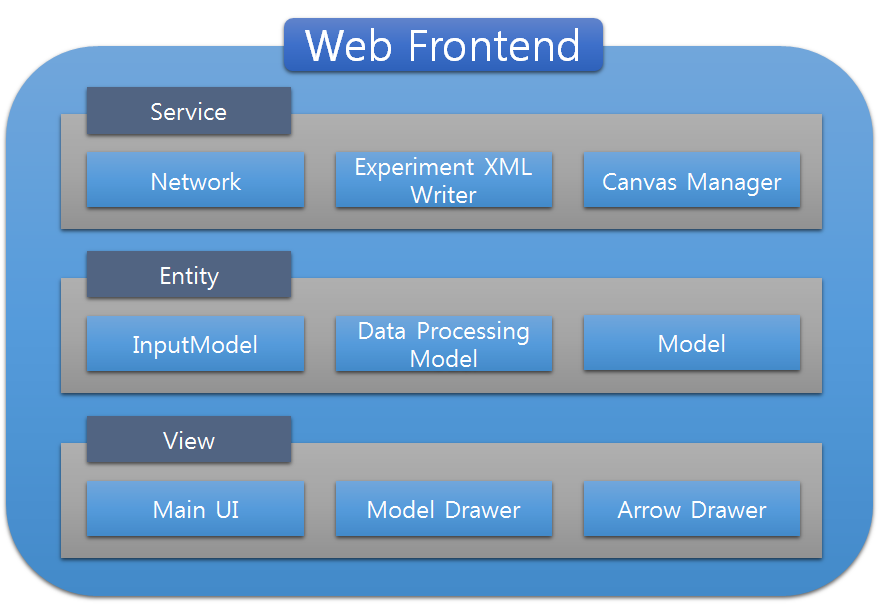
### Asynchronous Request Manager (ARM)

ARM은 서버 로직 중 비동기 작업을 관리하는 서비스이다. 사용자가 실험을 실행하는 경우 Task를 DTS의 Master로 전달하고 결과를 받아서 WAS에 돌려주는 일이나, 사용자가 업로드한 입력 파일을 Cloud File System에게 전송 후 완료에 대한 콜백 호출을 하는 등의 일을 하게 된다.

### Cloud File System (Cloud DFS)

Cloud File System은 실험에 필요한 입력 파일이나 실험 후 결과 파일 등의 데이터를 보관하고 관리한다. 그리고 사용자가 실험을 실행하는 경우 DTS의 Slave는 실험에 필요한 입력 파일을 DMS에 요청해서 가져오게 된다. 이렇듯 DrawML의 전체 시스템에서 파일 혹은 데이터를 공용으로 접근하고 관리해야할 경우 Cloud File System을 사용하게 된다.

### Web Frontend



Web Frontend는 클라이언트 사이드 로직을 나타낸다. 사용자가 UI에서 상호작용을 하면 Service를 사용해 서버와 통신하거나 해당 View의 Controller가 Model을 조작하는 하고 View가 다시 갱신되는 구조를 취한다. 핵심 로직인 실험 화면에서의 그래프 Visualization은 다음과 같이 진행된다.

먼저 사용자가 실험 페이지에 진입 시 Service의 Network module이 서버로부터 사용자의 Experiment XML을 가져오게 된다. 가져온 XML은 Converter에 의해 Graph Data 형태로 변환되어 Model로 보관하게 된다. 그리고 View에선 fabric.js를 기반으로 구현될 Experiment Renderer가 Experiment Data를 Visualization 하게 된다.

### Distributed Task System (DTS)

DTS는 Master, Slave, Worker 구조의 분산 Task 시스템이다. 여러 개의 Task들을 여러 대의 컴퓨터들을 이용해 효율적으로 실행하고 스케쥴링하는 것을 담당한다. 이에 대해서는 (다)에서 자세하게 다루도록 하겠다.

## WAS

###### 

###### WAS는 flask로 구성되었고, 전체적인 Architecture는 위와 같다. Request handler는 웹 과 관련된 처리를 하는 모듈이다. File handler는 upload된 파일을 관리하는 역할을 담당한다. TaskRunner는 python object code를 분산 시스템에 전달하고, 캐시를 업데이트하고, 결과를 전달 받는 역할을 한다. Code generator는 xml로부터 python object code를 생성하는 모듈이다. Redis-cache는 redis와 연동되어 캐시를 관리하는 모듈이고, SQLAlchemy는 Mysql과 연동되어 서비스 전반의 데이터를 관리하는 모듈이다.

### Request Handler

웹 서비스 전체의 요청과 응답을 담당하는 모듈이다. Blueprint를 이용하여 구성되었고, authentication의 경우 flask-login의 기능을 이용하였다. 각각의 모듈은 분리되어 directory로 구성되고, Blueprint는 이것들을 flask와 엮어 주는 다리역할을 수행한다. Permission required에 관한 부분은 flask-login을 사용하였으며, 웹 페이지의 대부분은 권한이 요구된다.

### File Handler

File Handler는 사용자가 업로드 하는 파일을 다루는 모듈이다. 서비스의 특성상 파일이 대용량 일 수 있기 때문에 file을 chunk단위로 받을 수 있도록 구성하였다. 사용자가 파일을 업로드 하면, was내 template file이 생성되고, 해당 파일은 Cloud DFS Connector를 통하여 storage에 저장된다. Cloud storage까지 chunk를 redirection하는 로직은 cloud DFS의 선정에 따라 달라질 수 있기 때문에 일반적으로 개방해 놓았다. 본 프로젝트의 cloud storage는 현재 chunk 전송기능을 제공하지 않기 때문에 파일 전송 후 cloud storage까지 업로드 하는 약간의 delay가 발생한다.

### Task Runner

Task Runner는 생성된 코드를 분산 시스템에 전송하고, 결과를 받고, 상태를 업데이트한다. 생성된 코드는 2개이상일 수 있다. 데이터 전처리와 학습 등으로 이루어 질 수 있기 때문에 이를 동기적으로 처리하기 위하여 Callback의 closure를 사용하였다. Callback pattern을 사용한 이유는 몇 가지가 있다. 첫번째로 분산 시스템에서 task가 비동기로 진행되어야 하기 때문에 task runner는 분산시스템에 요청을 한 후 바로 종료되어야 한다. 하지만 결과값을 받아야 하는 문제점이 있다. 이를 처리하기 위한 여러 방법이 있지만, 본 프로젝트에서는 직관적인 callback과 closure방식을 취하였다. Task Runner는 또한 task 의 결과에 따라 redis내 캐시의 업데이트를 요청한다.

### Code Generator

Code generator는 XML을 python code로 변환해 주는 모듈이다. Jinja2를 이용하며, 각각의 모델마다 template이 정의 되어 있다. XML의 parsing을 위하여 element tree를 사용하였으며, 재귀적으로 호출하며 xml의 구조를 template variable에 반영한다. 반영된 template variable은 마지막에 jinja2를 이용하여 렌더링 되고, 파이썬 코드가 생성된다. 모델의 layer의 경우 여러 레이어가 중첩될 수 있으므로 일반화 하기 위하여 재귀적 호출 방법을 적용하였다.

### Redis-cache

Redis-cache는 redis와의 통신을 담당하여 task의 상태를 저장하고 삭제하고 갱신하는 모듈이다. Task가 실행되거나 종료, 실패 취소 되었을 때 상태를 업데이트 하게되며, 상태의 종류에는 idle, success, fail, cancel 이렇게 4가지 종류가 있다.

### SQLAlchemy

본 프로젝트의 주 데이터들은 Mysql로 관리된다. SQLAlchemy는 myql의 orm으로 사용되었으며, 데이터의 접근 및 갱신, 생성, 삭제 등 모든 트랜젝션은 SQLAlchemy의 session을 통하여 이루어 진다. 본 프로젝트의 데이터 스키마는 다음과 같다.

1. User

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column | id | Date\_created | Date\_modified | User\_id | pw |
| Type | int | date | date | VARCHAR | VARCHAR |

1. Data

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column | id | Date\_created | Date\_modified | User\_id | name | path | type |
| Type | int | date | date | VARCHAR | VARCHAR | VARCHAR | VARCHAR |

1. Trained Model

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column | id | Date\_created | Date\_modified | User\_id | name | path | XML |
| Type | int | date | date | VARCHAR | VARCHAR | VARCHAR | BLOB |

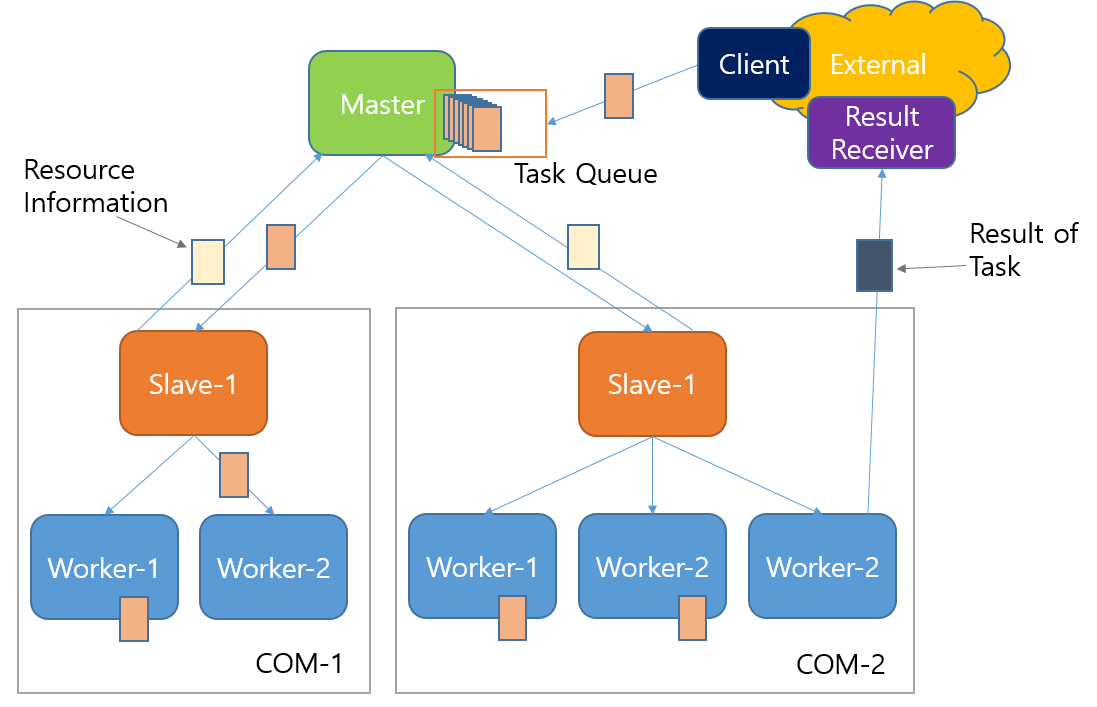
1. Experiment

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Column | id | Date\_created | Date\_modified | User\_id | name | drawing | XML |
| Type | int | date | date | VARCHAR | VARCHAR | BLOB | BLOB |

## Distributed Task System (DTS)

###### 우리 프로젝트에서는 여러 대의 컴퓨터들을 이용해 동시에 여러 개의 Task들을 처리해야 한다. 따라서 이러한 분산 기능을 담당하는 모듈을 최대한 독립적으로 설계하고 최소한의 인터페이스만 유지하여 다른 모듈이나 시스템에서 분산 Task 기능을 사용 할 수 있도록 하였다.

### Architecture



위는 우리가 설계한 분산 구조이다. 일단 개념적으로는 크게 Master와 Worker로 나뉜다. Master는 Task들을 관리하며 효율적으로 Worker들에게 Task를 할당하는 역할을 한다. Worker는 Master로부터 할당된 Task를 처리하는 역할은 한다. Master에는 Task Queue가 존재하여 Master로 요청된 Task들은 Task Queue에 쌓이게 된다. 그리고 Master들은 Task Queue들에 있는 Task들을 앞으로 기술할 알고리즘들에 따라 효율적으로 Worker에 분배한다. Worker는 그 자체가 하나의 컴퓨터를 의미할 수 도 있고 하나의 컴퓨터가 여러 개의 Worker로 구성 될 수도 있다. 이는 아래에서 자세하게 기술하도록 하겠다.

위는 우리가 설계한 분산 구조이다. 일단 분산 시스템 자체는 크게 Master, Slave, 그리고 Worker로 구성되어 있다. 그리고 분산 시스템이 외부 시스템에게 요구하는 interface로서 Client와 Result Receiver가 있다.

#### Client

Client는 분산 시스템에게 Task 를 등록하는 개체이다. 여기서 Task 처리는 비동기적으로 이루어진다. 즉, 분산 시스템에게 Task를 요청하고 그 결과는 추후에 Result Receiver를 통해 받는다.

#### Result Receiver

Result Receiver는 분산 시스템에 등록한 Task의 결과를 보고받는 개체이다. 결과는 Master, Slave, Worker중 누구에게 올 지 모른다. 만약 Master혹은 Slave에게서 온다면 그 것은 Task의 실패를 의미할 것이다. Worker에게서 온 결과라면 그것은 실패일 수도 있고 성공일 수 도 있다. 성공이라면 그 것은 처리 결과를 담고 있을 것이다.

#### Master

Master는 분산 시스템의 우두머리로서 전체 분산시스템에 1개만 존재한다. Master는 Client로부터 Task를 등록받고, 그 Task들을 여러 개의 Slave에 스케쥴링하는 역할을 한다. 그리고 스케쥴링에 필요한 리소스 정보를 Slave로부터 전달 받는다.

#### Slave

Slave는 분산 시스템에서 Computer 1대를 관리하는 역할을 한다. Slave는 Computer내의 리소스 자원 (cpu, ram, gpu등)을 실시간으로 모니터링하며 이 것을 실시간으로 Master에게 보고한다. 또한 Master로부터 Task들을 받아서 실행시키는 역할을 한다. 여기서 실행과정은 내부적으로 Worker를 생성한 뒤, Worker에게 실행을 위임하는 방식으로 이루어진다.

#### Worker

Worker는 분산 시스템에서 실질적으로 Task를 실행하는 역할을 한다. Master나 Slave는 영속적으로 존재하는 데에 비해 Worker의 생명주기는 Task와 함께한다. Slave에 의해서 생성되고, Task 처리가 끝나면 제거된다. Worker는 Task 처리가 완료되면, Result Receiver에 결과를 보고한다.

### System Stability

분산 시스템은 Network 망을 통해 서로 통신한다. 따라서 통신이 원활하지 않을 수 있다. 그리고 분산 시스템 중 특정 PC가 죽을 수도 있다. 이럴 경우에 대해서 대책을 마련해야만 한다. 이렇듯 우리는 시스템 안정성을 위해 여러 가지 방법을 사용하였다.

#### Master와 Slave간의 Heart Beat

Master와 Slave간의 연결 유지는 매우 중요하다. 우리는 Master와 Slave간의 연결 상태 확인을 위해 Heart Beat를 우리의 프로토콜 내에 추가하였다. Master와 Slave는 일정 주기(1초정도)마다 hear beat message를 주고 받는다. 만약 5회 연속 heart beat가 작동하지 않는다면, Master는 해당 Slave가 죽었다고 판단하고 시스템에서 제거한다. 이 때, Slave가 소유하고 있던 Task들을 전부 다시 Task Queue로 들어가 다른 Slave들에게 다시 스케쥴링된다.

#### Slave의 Worker Polling

Worker는 직접적으로 Task를 실행하고 처리하는 node이다. 따라서 만약에 Task가 이상하거나 뭔가 문제가 생겼을 경우에 Worker는 아무런 표식없이 죽을 수도 있다. 따라서 이러한 경우를 대비하기 위해 Slave는 주기적으로 Worker들이 살아있는지 Polling한다. 만약 죽었다면, Task가 실패한 것으로 간주한다.

#### Task Capability

시스템 전체의 처리능력에 비해서 지나치게 많은 task들이 요청된다면, 결국 task들을 다 처리하지 못하고 task queue에 task만 쌓일 수 있다. 따라서 시스템 전체에서 최대로 생성할 수 있는 worker 수의 2배가 넘는 task들이 task queue에 존재한다면, 더 이상 task 처리요청을 받지않고 거부하도록 한다.

#### Exception Handling

분산 Task 시스템 유닛들 간의 메시지들은 대부분 비동기적으로 전송되고 수신된다. 즉, 각 유닛들은 자기가 받은 메시지들이 어떠한 맥락에서 수신되어졌는지 판단해야하고, 그 메시지가 유효한 지를 검사해야한다. 만약 예외 상황의 경우 라면 적절한 처리를 해야한다. 특히, 네트워크 상에서 서로 병렬적으로 유닛들이 동작함으로 네트워크상에서의 latency때문에 생기는 동기화 문제들 또한 고려해야한다. 우리는 이러한 것들을 전부 고려하여 프로토콜을 설계하고 각종 예외에 대한 처리를 함으로서 분산 시스템의 안정성을 높였다.

### Task Scheduling Algorithm

우리는 주어진 Task들을 주어진 Resource들을 통해 효율적으로 실행 시킬 의무가 있다. 따라서 Task들의 정보와 Computer들의 정보를 통해 Task들을 여러 개의 Slave에 효율적으로 스케쥴링해야 한다. 그러나 문제는 Task들의 정보를 많이 알 수가 없다는 것이다. 따라서 불확실성 속에서는 오히려 복잡하고 추측에 근거한 스케쥴링 알고리즘은 성능을 떨어뜨릴 가능성이 있다고 판단하여 간단하고 명백한 스케쥴링 알고리즘을 사용하였다.

우리 분산 Task 시스템은 총 2가지의 Task를 지원함으로 각각에 대한 Scheduling Algorithm을 제공한다.

#### Scheduling Algorithm of Data Preprocessing Task

데이터 전처리의 경우 오직 cpu만을 사용한다. 따라서 아래의 우선 순위에 의해 적합한 slave의 cpu에 task가 할당된다.

1. Cpu core 개수가 더 많다.
2. a가 같다면, available cpu percentage가 더 크다.

#### Scheduling Algorithm of Tensorflow Task

아래의 우선 순위에 의해 적합한 slave의 cpu or gpu에 task가 할당된다. 여기서 gpu는 cuda가 가능한 gpu만을 말한다.

1. Gpu가 cpu보다 더 우선 순위이다.
2. Gpu라면, cuda compute capability가 더 높다.
3. Available한 Computation Resource가 오직 Cpu만 있다면, (가)의 알고리즘에 따라 스케쥴링한다.

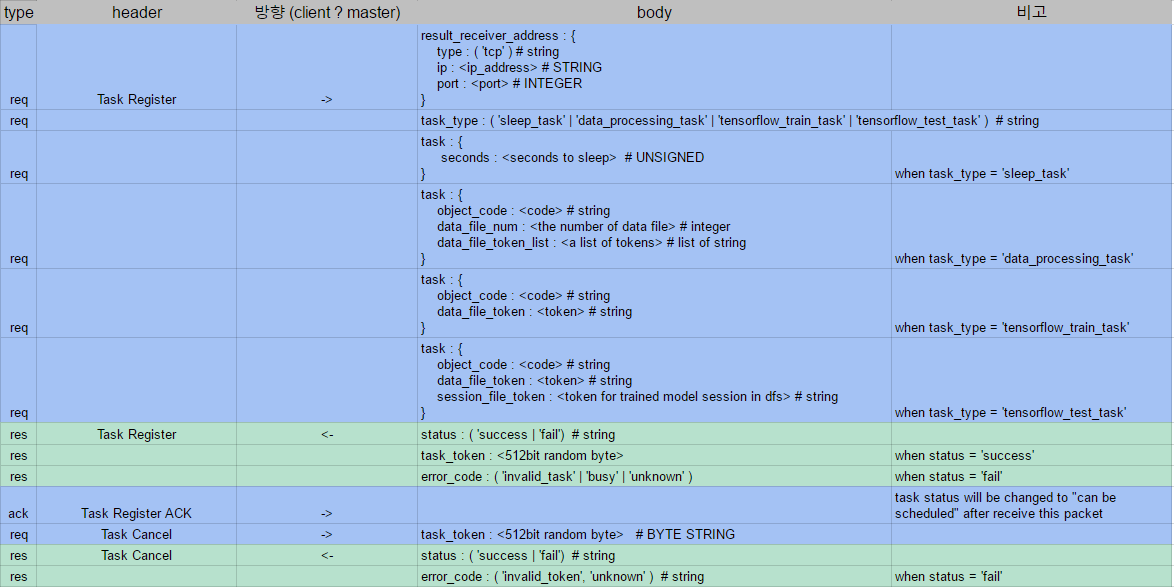
### Protocol

분산 Task 시스템의 구성요소인 Client, Result Receiver, Master, Slave, Worker간의 통신을 위한 프로토콜을 직접 설계하고 구현하였다. 우리가 설계한 프로토콜은 tcp/ip layer 위에서 동작한다.

#### Message Serialization

우리는 네트워크 bandwidth를 최소화하고 메시지 송수신 IO시간을 줄이기위해서 메시지를 직렬화하여 주고 받는다. 우리는 message serialization을 위해서 Google의 ProtoBuf를 사용하였다.

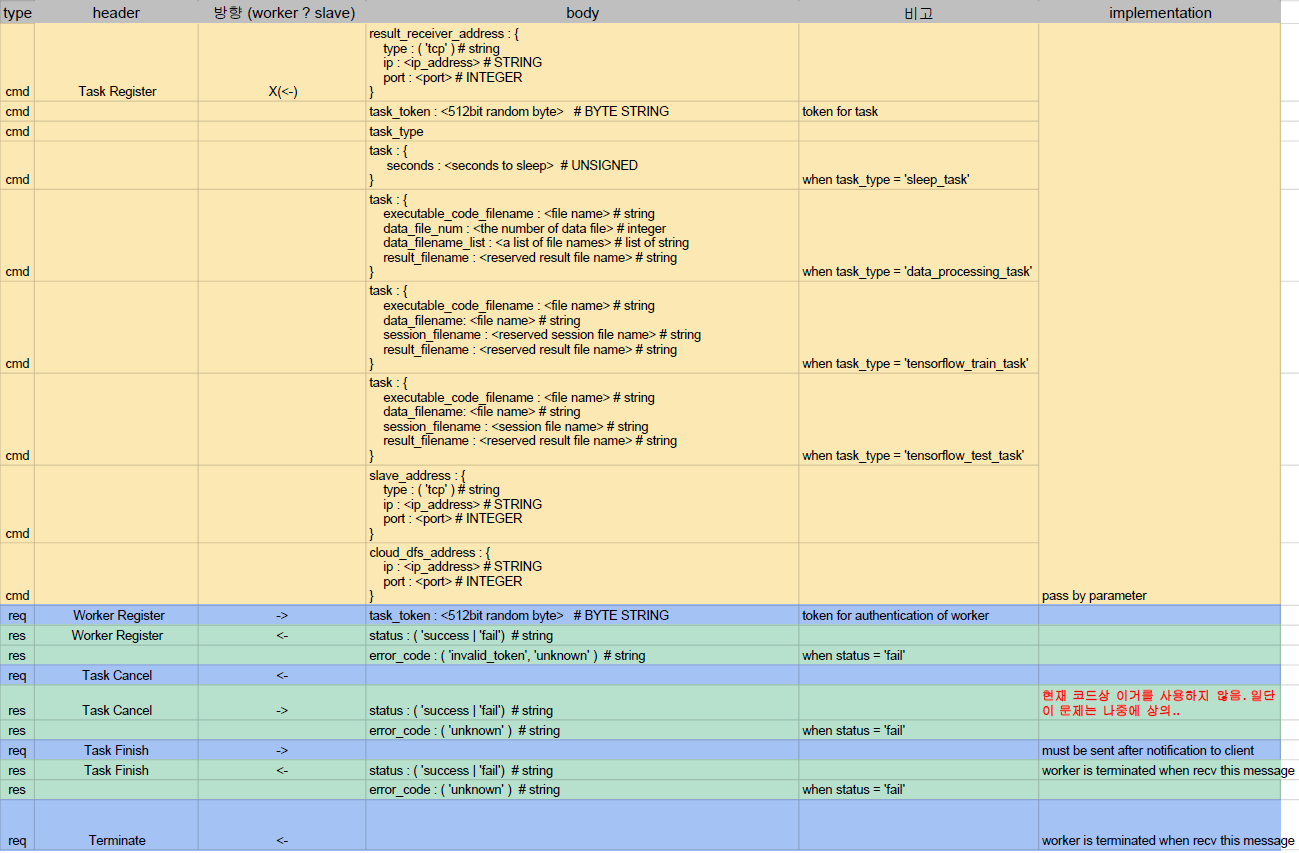
#### Client <-> Master



#### Master <-> Slave



#### Slave <-> Worker



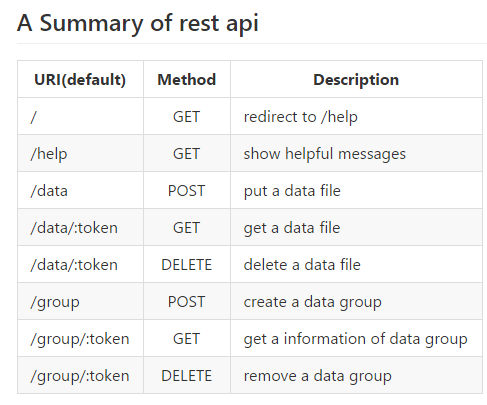
#### Any <-> Result Receiver



## Cloud File System

###### Cloud File System은 DrawML에서 각종 데이터파일들을 관리하는 시스템이다. 외부에서 rest api를 통해 파일들 업로드 & 다운로드 할 수 있다.

### Rest Api



자세한 API문서는 <https://github.com/DrawML/cloud-dfs/wiki/Rest-API> 를 참고하기 바란다.

### 외부에서 손쉽게 사용할 수 있도록 python library 지원

사용자가 일일이 Cloud File System에 rest-api를 보내는 것은 불편하고 성가신 일이다. 따라서 우리는 Cloud File System에 관련된 동작들을 추상화하여 python library를 만들어 제공한다.

## Experiment Realization & Execution Flow

###### 

###### 위는 실제로 사용자가 Web UI를 통해서 작성한 Graph UI형태의 Experiment가 어떤 흐름으로 변환되고 실행되는 지 나타낸 그림이다.

###### Graph UI

###### 처음에 사용자는 Web UI를 통해서 편리하게 Graph 꼴의 Experiment를 만든다.

###### XML format

###### Graph UI 형태의 Experiment는 서버로 전송이 가능한 XML format으로 변환된다. 이 XML format의 experiment는 서버에서 실제로 DB에 저장된다.

###### Tensorflow Object Code

###### b)에서의 XML format의 Experiment가 실제로 실행이 필요한 경우 이는 Tensorflow Object Code로 변환된다. 이는 실제로 바로 실행가능한 Code는 아니고, 여러가지 Code Block들에 tag정보들이 붙은 형태이다.

###### Executable Tensorflow Code

###### c)에서 생성된 Tensorflow Object Code는 현재 시스템의 정보와 Worker들의 상태에 따라 실제적으로 실행가능한 code로 변환된다. 이 code는 distributed processing service의 master에 의해 특정 worker에게 전달되고 실행된다.

## ML 컴포넌트 화

### Linear Regression

###### Linear Regression은 선형적인 값을 학습을 시켜서 예측을 하는 알고리즘이다. 예를 들어 몇 시간을 공부를 하였을 때 몇 점의 시험점수가 나오는지, 또는 2종류 이상의 데이터 값을 입력 받아서 선형적인 하나의 값을 예측하는데 사용된다.

###### 실제 Linear Regression을 사용을 하기 위해서는 간단한 초기값을 설정 하게 된다. 이후 주로 Gradient Descent 알고리즘을 이용하여 Learning rate를 조정을 하여서 training set에 적합한 rate를 고르게 된다. 즉 일반적인 경우에는 training data가 들어오면 초기값을 설정하고 Gradient Descent의 Learning rate를 조정하는 것이 전부이다.

###### Linear regression은 위에서 설명한 것과 같이 사용하는 입장에서 W의 초기값과 Learning rate만 조절을 하면 누구나 사용할 수 있기 때문에 컴포넌트화 하는 과정에서 설정해 주는 값은 2가지 초기값 설정과 Learning rate은 조정 가능 하다.

### Polynomial Regression

Polynomial Regression은 Linear Regression과 대부분 유사하지만, 실제 원하는 모델이 직선이 아닌 경우에 주로 사용하게 된다. 즉 2차 이상의 항이 들어간다.

Linear Regression과 대부분 유사한 구조를 가져서 형태 또한 비슷하지만 1가지가 달라지게 되는데 들어오는 Training data에 대한 W를 직접 선택을 해야 된다는 점이다. 예를 들면 X1,X2,X3가 들어왔다고 하면, Polynomial regression의 Hypothesis는 가 될 수도 있고, 처럼 만들 수도 있다. 즉 사용자가 직접 각 X마다 값을 지정해 주어야 된다.

Polynomial Regression의 경우 다차항에 따른 x값, 즉 model 입장에서의 input값의 영향 외엔 Linear Regression과 모든 로직이 같다. Polynomial regression의 다차항을 parsing하여 training을 할 수도 있지만, 위의 경우 직접 입력한 식에 대하여 값을 맞추는 것보다는 curve fitting을 수행하는 것이 보다 일반적이고, 유용하다고 판단되었기 때문에, 이번 과제에서는 보류 하였다.

Curve fitting의 경우 scikit-learn 의 optimal weight gradient를 이용하는 것을 고려 해 보았다.

### Logistic Classification

Logistic Classification은 linear한 값이 아닌, discrete 한 값을 예측 할 경우에 사용한다. 예를 들면 종양의 크기를 입력 받아서 악성종양인지, 일반종양인지 예측을 한다거나, 키, 몸무게, 등 신체 사이즈를 입력을 받아서 남성인지 여성인지 구분을 할 때 사용하면 유용하다.

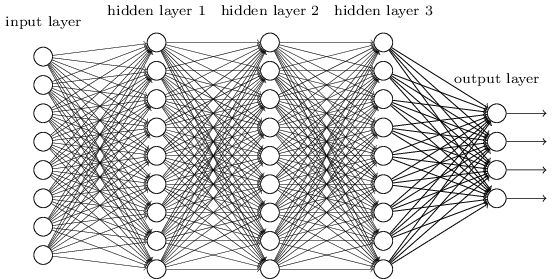
Logistic Classification부터 이후의 model들은 classification으로 사용되는 것들이다. 앞서의 것들은 prediction을 위한 model인 반면에, classification model은 output을 vector로 표현 한다. Logistic regression의 경우 model의 특성 상 2가지 class 로 분류하는 binary classification으로 동작한다.

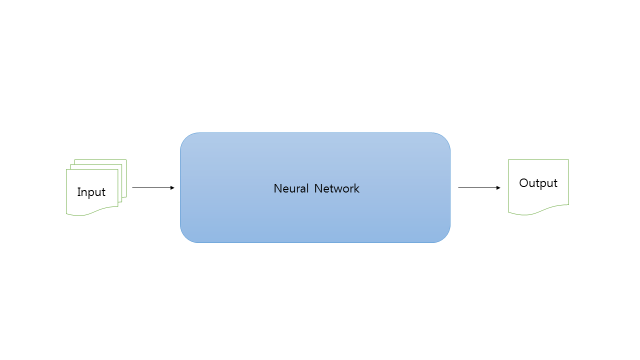
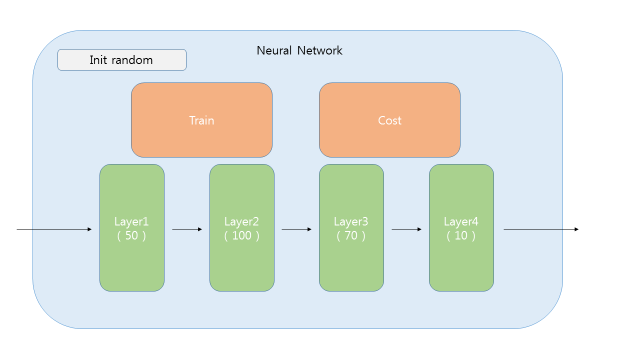
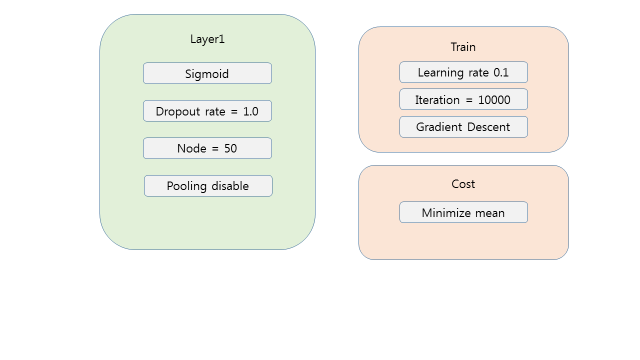
### Softmax Classification (Multinomial classification)

Softmax Classification은 2가지로 분류 할 수 있었던, Logistic Classification을 활용하여 2가지 이상의 것으로 분류를 해 내는 ML 알고리즘이다. 예를 들면 한우의 각 부위의 크기, 무게, 키를 입력 받아 한우의 등급이 A, B, C 등급으로 나누어 질 때 어떠한 등급으로 분류가 될 것인지 예측을 하는 경우를 예로 들 수 있다. 이 외에도 Softmax Classification은 다른 ML의 복잡한 알고리즘의 근간이 되는 아주 중요한 알고리즘 이다.

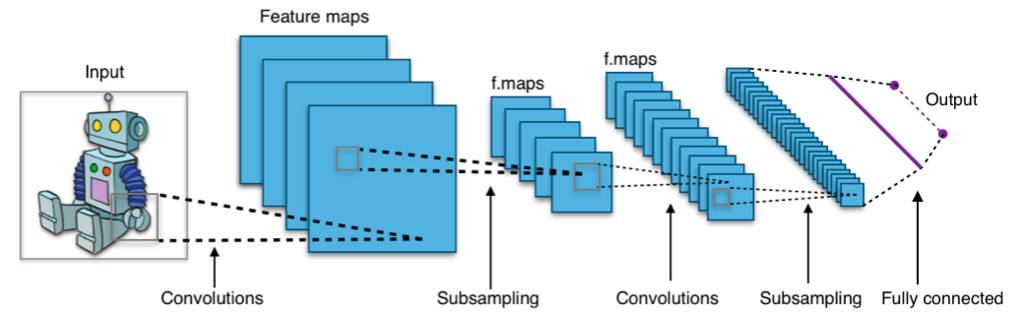
Softmax classification의 logistic regression과 가장 큰 차이점은 여러가지의 class이다. 본 프로젝트에서는 이 여러 클래스를 다루기 위하여 one-hot-vector를 사용한다. 따라서 사용자의 input data shape은 feature의 개수와 class의 수를 반영한 값이 되어야 한다.

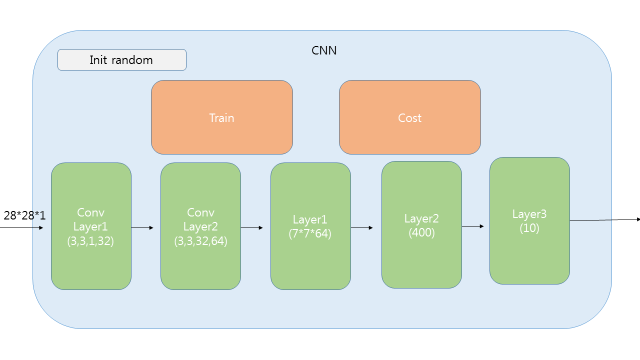
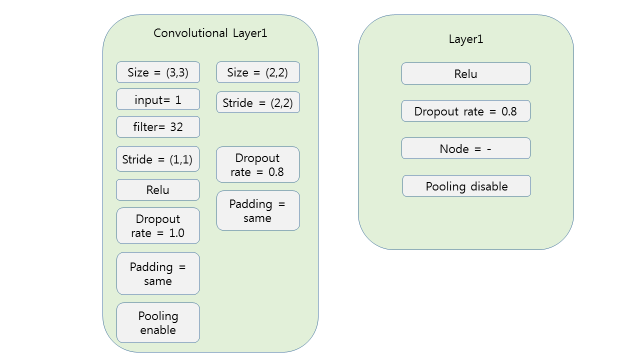
### Neural Network (NN)



Neural Network은 여러 개의 layer를 이용하여 학습하는 것이 특징이다. 최근 Deep learning이라고 불리는 방법 또한 neural network의 연장선이므로 두 개념을 통합하여 칭하였다.  
  
  
NN은 기계학습 방법론에 layer를 1개이상 놓을 수 있는 모듈이다. 기본적으로 3개의 layer가 있으며 필요에 따라 layer를 추가하거나 삭제할 수 있도록 구성했다. 각 layer는 요약 정보로 layer내 node의 수가 보여지며, layer간 weight의 초기화 방법은 설정할 수 있도록 제공된다.  
  
layer의 상세정보는 layer에 적용되는 수식계산과, dropout rate 그리고 node 수 등이 있고 그림과 같다. Train과 cost의 경우 필요에 따라 상세 설정을 바꿀 수 있도록 제공한다.

### Convolutional Neural Network (CNN)



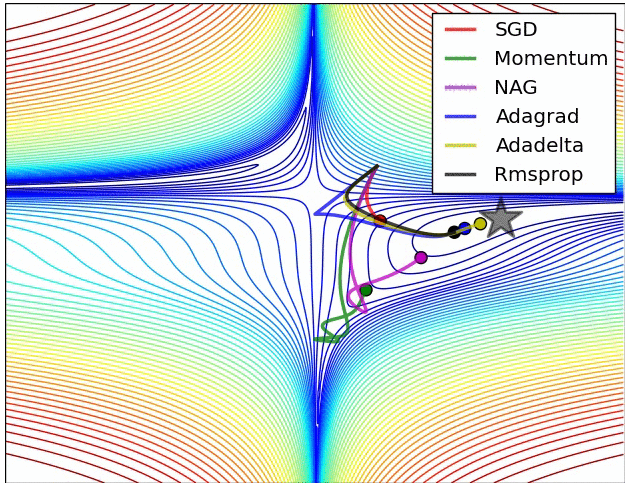
CNN은 일부를 feature로 여러 개의 filter에 적용하는 방법이다. CNN의 가장 큰 특징은 convolutional latyer와 pooling이다. 레이어가 굉장히 많기 때문에 이 두 가지를 하나의 layer로 묶어서 추상화 하였다.  
   
CNN은 convolution layer와 일반 layer로 구성된다. Convolution layer는 요약정보로 size와 input, output의 수가 주어진다. Input의 경우 이전 layer의 output과 연관되어 결정된다. Convolution layer는 merge를 위해 일반 layer에서 합쳐지게 되고, 최종적으로 예측된 값을 내보낸다.  
  
 Convolution layer의 상세정보는 그림과 같다. Size와 input, filter(convolution)을 설정 할 수 있도록 하며, 이는 이후의 convolution layer에게 영향을 미치게 된다. 기본적으로 적용되는 수식과 dropout rate, padding 그리고 pooling을 설정할 수 있도록 하며, pooling이 설정되었을 경우 옆에 pooling layer의 정보가 표현된다.

### Activation & Initializer & Optimizer

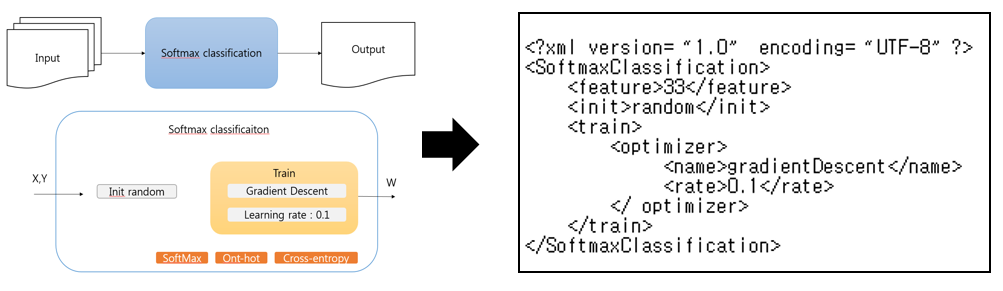
Activation으로는 sigmoid, tanh, softmax, relu 등을 제공할 예정 이었으나, 현재는 relu와 sigmoid만 제공한다. 또한 RBM, xavier등의 초기화 방법은 추후 제공 예정이다. Layer별로 dropout rate를 정할 수 있도록 구성하여, 필요 시 조정 할 수 있도록 하였다.

모든 방법론들은 feature간의 범위를 고려하여 regularization이 필요하면 적용된다. 또한 최적화를 위하여 꼭 필요한 learning rate와 optimizer를 선택할 수 있도록 구성된다. Learning rate는 default값이 정해진 상태에서 입력 값으로 대체할 수 있도록 구성 되고 optimizer의 경우 tensorflow에서 지원하고 있는 7가지를 제공 된다. 다음은 tensorflow에서 지원하고 있는 7가지 optimizer와 각각의 성능을 보여주는 그림이다.

* + 1. Stochastic Gradient Descent
    2. Momentum algorithm
    3. Adagrad algorithm
    4. Adadelta algorithm
    5. Adam algorithm
    6. FTRL algorithm
    7. RMSProp algorithm



### 컴포넌트 화 된 모델의 XML화



XML은 데이터를 구조적인 텍스트 데이터 포맷으로 체계화를 하여 데이터 구조가 모호하지 않도록 설계를 할 수 있다. ML 각 Model들의 컴포넌트화를 진행을 하였고 각 컴포넌트에는 변할 수 있는 각 속성이 지정이 되어 있다. 그리고 컴포넌트화 된 ML 알고리즘 들의 속성들을 그대로 XML로 적용한다. 다음은 본 프로젝트에서 사용 되었던 모델 별 xml들이다.

#### Linear regression



#### Logistic regression



#### Softmax regression



#### Neural network



#### Convolutional neural network





#### Data pre-processing



위의 xml들은 각각 type에 맞는 Python code로 변환된다. Python code로의 변환은 Jinja2를 이용한다. Jinja2는 HTML template engine으로 배포되어 있지만, 본 프로젝트에서는 python code를 생성하도록 만들었다. 모든 model에서 사용하는 것들은 공통모듈로 빠져있고, 각각의 model마다 특성을 살려서 해당 template code를 작성하였다. Template code는 1차적으로 xml에서 변환되며, 이때는 device configuration이 되어 있지 않은 상태이다. 이 코드는 다시 distribution system에 들어가게 되고, 이 중간에서 device configuration이 설정된다. 설정된 python code는 master를 거쳐 worker에서 실행된다. 실행된 결과를 받기 위해, WAS 내에서 closure 방식의 Callback을 사용한다. 이 callback은 돌아온 결과에 대하여, 다른 task를 실행하거나, status를 업데이트하거나, 작업을 완료하고, 데이터를 저장한다.

## Web UI

###### 

###### 

###### 

###### 전체적인 web UI는 기본적인 HTML, CSS, JavaScript를 토대로 진행을 했으며, 실제 중앙부분의 canvas의 경우 Fabric.JS를 이용하여 구현을 하였다. 기본적을 틀을 보면, 좌측에서 모델 및 데이터를 선택 할 수 있고, 이 모델 또는 데이터를 중앙 canvas에 끌어놓게 되면, Fabric.JS를 이용해 만든 UI 컴포넌트가 만들어 지게된다. 이 UI 컴포넌트를 클릭, 이동, 두 모델의 연결을 통해서 자신이 원하는 Experiment를 제작 할 수 있다.

###### 가장 주요한 중앙 canvas의 구조는 내부적으로 2가지로 구별이 되어 있다. 우선 Regression 관련된 머신러닝의 모델들은 일괄적인 UI로 표현해도 무리가 없기 때문에, 하나의 모델로 구조화를 시켜서 제작을 하였다. 가장 위에 모델의 종류가 나오고, initializer와, Optimizer의 경우는 머신러닝의 모델에서 주요한 파라미터이기 때문에 UI에 표현이 될수 있게 하였다. 모델을 클릭을 하게 되면, 오른쪽 Option패널에서, Initializer, Optimizer, Regularization training\_epoch를 설정을 할 수 있으며, Delete를 통해서 현재 내가 선택한 모델을 제거 할 수 있다.

###### 

###### 

###### Regression 모델들의 경우에는 속성값들이 정적인 반면에 Neural Network나, Convolutional NN의 경우에는 그 컴포넌트의 모양이 가변적일 수 밖에 없다. 따라서 Regression 컴포넌트와는 다르게 가변적으로 Layer를 추가를 하여서 컴포넌트의 모양이 바뀔 수 있도록 하였다. 전반적인 동작 방식은 위의 Regression모델과 거의 일치하지만, Layer를 추가, 삭제를 진행을 하면서 컴포넌트의 모양이 변하는 과정이 Regression모델과 가장 큰 차이를 보이는 부분이다. 또한 NN과 CNN의 경우 추가된 Layer에 따른 Option 패널의 종류들이 바뀌어 지게 된다.

###### 하단패널의 경우에는 Save, Send의 버튼이 있고 최우측에 현재 모델의 상태가 표시되어 있다. Save 버튼을 누를 경우 Canvas에 그린 Experiment가 우리가 미리 지정한 XML형태로 변환이 되어지고 이것을 WAS로 전송을 하게 된다. 이 XML를 차후에 웹페이지를 reload하거나 Was 서버에서 distribution 서버로 보내서 tensorflow task를 진행하는 데 이용되어진다. Send 버튼을 누르면, 저장되어진 XML을 토대로 서버에서 나의 experiment를 학습시켜주며, 이 때의 상태를 최우측 status에 표시를 하게 해준다.

# 향후 발전 방향

## ML 컴포넌트의 추가 & 자유도 향상

###### 현재에는 한정된 몇 개의 컴포넌트만 지원하지만, 향후에는 더 많은 컴포넌트들을 지원할 수 있을 것이다. 그리고 사용자가 자유롭게 machine learning을 설계할 수 있는 체계를 갖출 것이다.

## 프로젝트의 오픈소스화

###### 프로젝트를 오픈소스로 공개하고, 개발자들이 자신의 컴퓨터들의 환경에 맞춰서 우리 시스템을 설치하고 운용할 수 있도록, 다양한 스크립트들과 툴들을 제공할 수 있을 것이다. 그리고 오픈소스화를 위해 라이센스 고려, 문서화 등의 작업이 추가로 수행 될 수 있을 것이다.

## Distributed Task System의 향상

###### 현재에는 분산 Task 시스템이 오직 스케쥴링만을 지원하지만, 한 개의 Task를 여러 개의 Computation Node를 통해 수행하는 분산처리 기능을 추가할 수 있을 것이다.

## Cloud File System의 분산시스템화

###### 현재에는 Cloud File System이 오직 1개의 컴퓨터로만 이루어져있다. 클라우드 분산 파일시스템을 구축하면 좀 더 많은 데이터를 저장하고 동시에 여러 개의 IO를 진행할 수 있을 것이다.

# 개발 환경

##### **OS** : Microsoft Windows 7/10, Ubuntu 16.04LTS, Mac OS X – El capitan

##### **Tool**: PyCharm, Vim, Gcc, etc…

##### **Language**: Python3, HTML5, Javascript, Bash script

# 항목별 가중치

# 개발 일정

##### 