



---

## 8회차 교육

2022

# I. Autoencoder

---

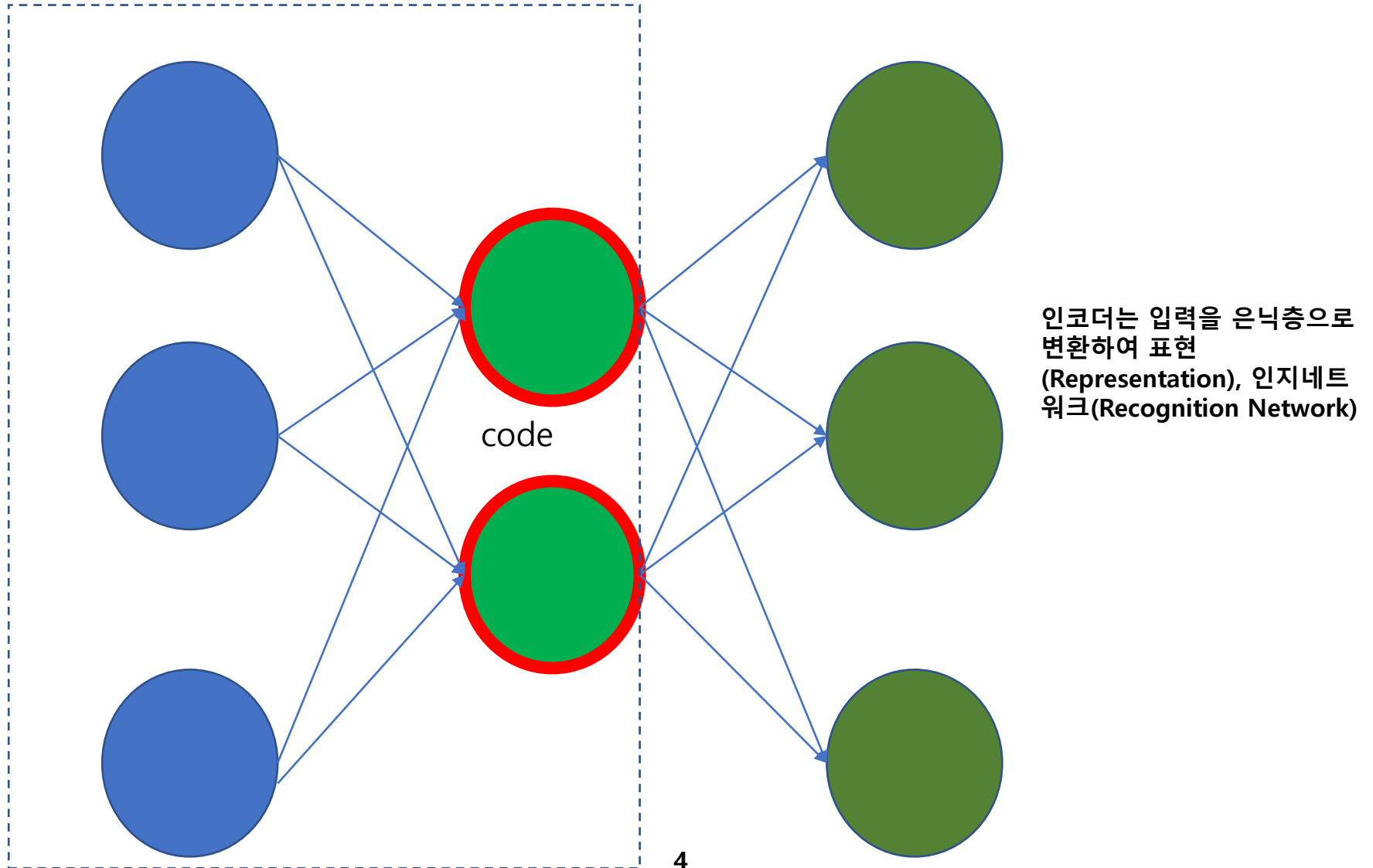
1. 오토인코더(Autoencoder): 입력과 출력이 동일하며, 입력을 출력으로 나타내는 비지도학습을 위한 인공신경망의 한 종류
2. 인코더(Encoder)와 디코더(Decoder): 인코더는 입력을 은닉층으로 변환하여 표현(Representation), 디코더는 은닉층에 표현된 값을 출력으로 변환
3. Stacked Autoencoder: 여러 은닉층을 갖는 Autoencoder이며, 여러 은닉층을 통해 입력값을 더 복잡한 인공신경망으로 복원함(Reconstruction)

## 오토인코더란?

- 입력과 출력이 동일
- 입력을 출력으로 나타내는 비지도학습을 위한 인공신경망의 한 종류
- 인공신경망이 입력을 재구성(Reconstruction)
- 입력과 재구성된 출력의 차이를 줄이도록 학습

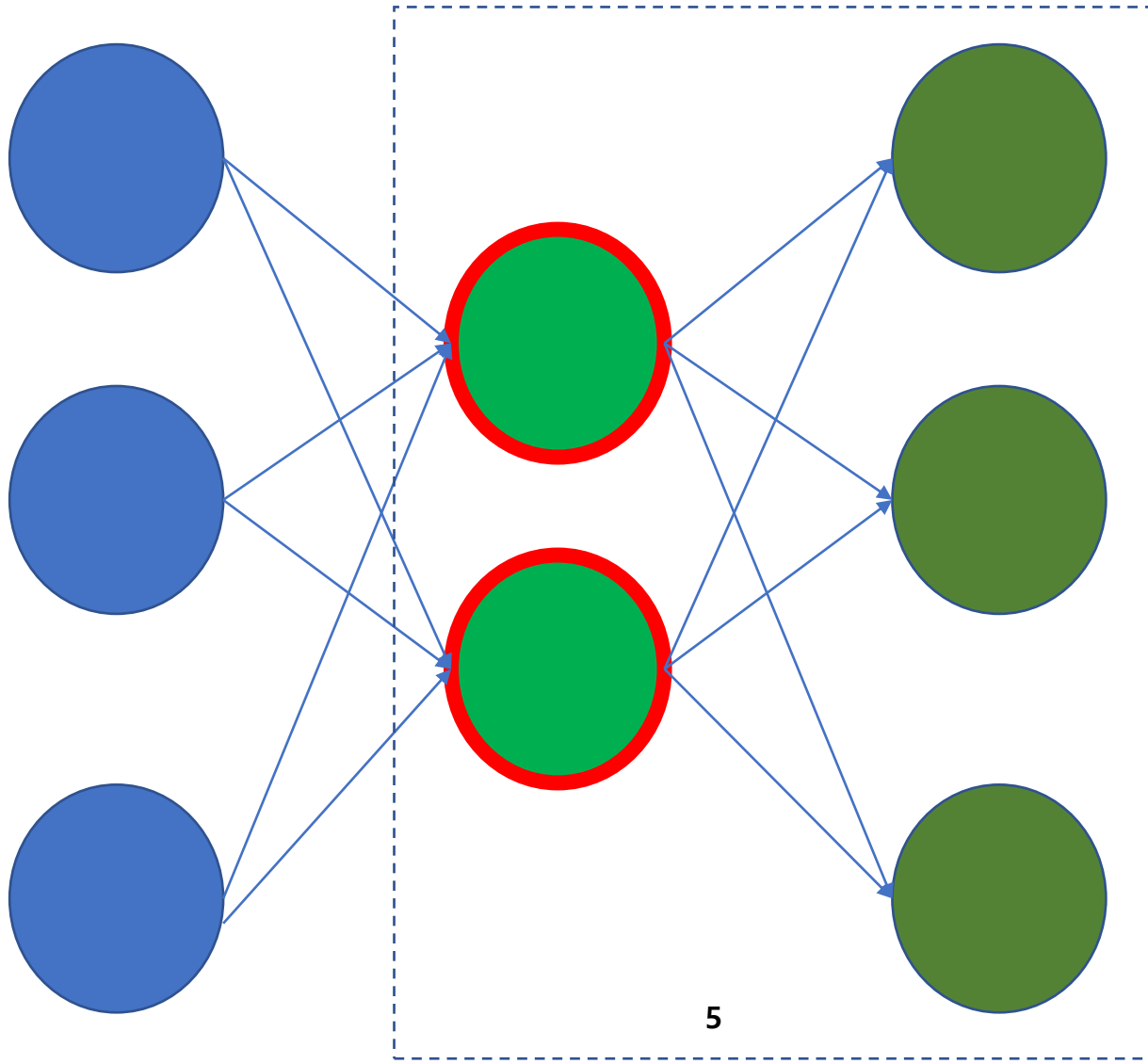
# I. Autoencoder

## 오토인코더의 구성: Encoder+Decoder



# I. Autoencoder

## 오토인코더의 구성: Encoder+Decoder

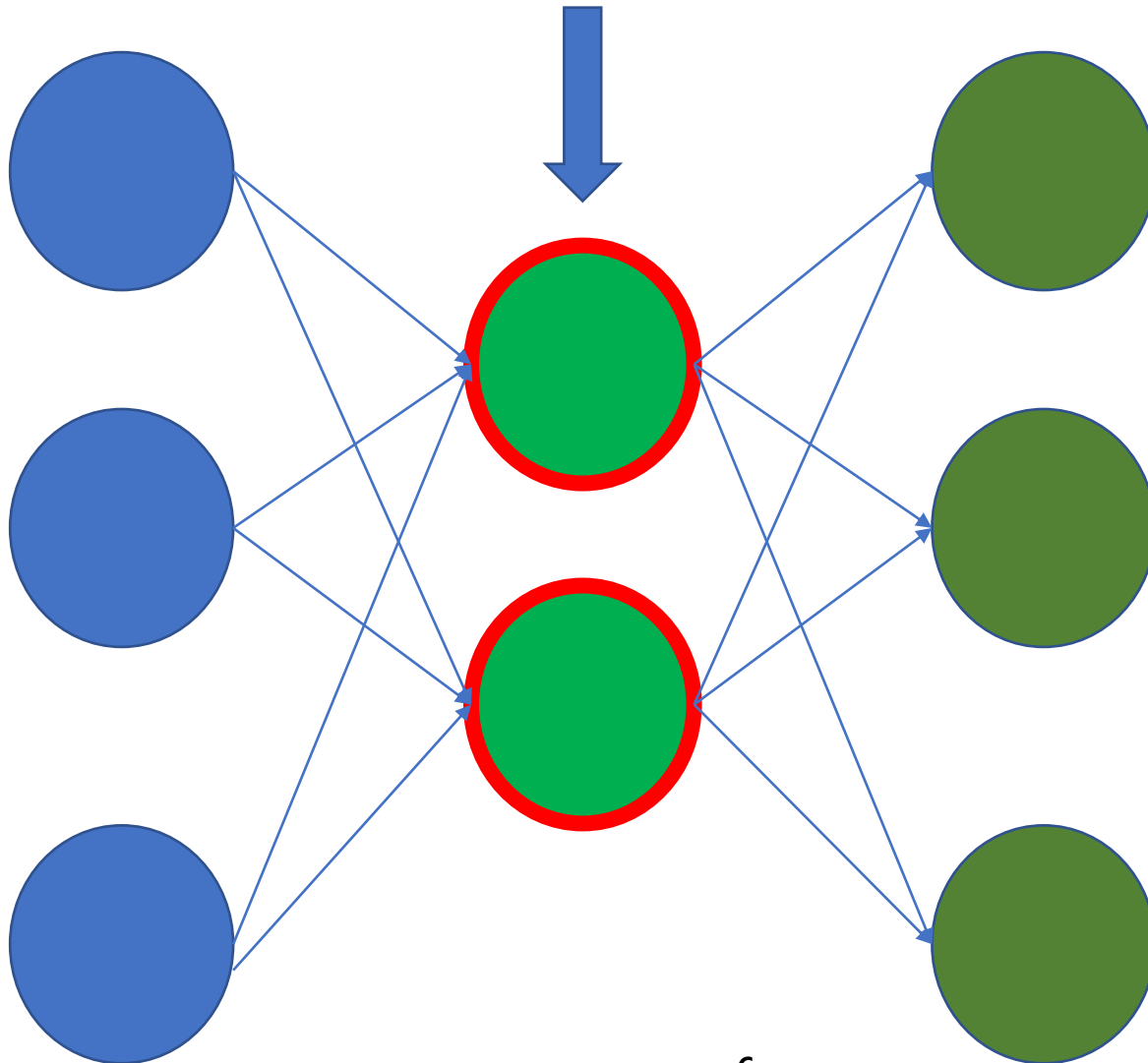


디코더는 은닉층에 표현된 값을 출력으로 변환, 생성 네트워크(Generative Network)

# I. Autoencoder

## 오토인코더의 의미

데이터의 압축(Undercomplete Autoencoder)

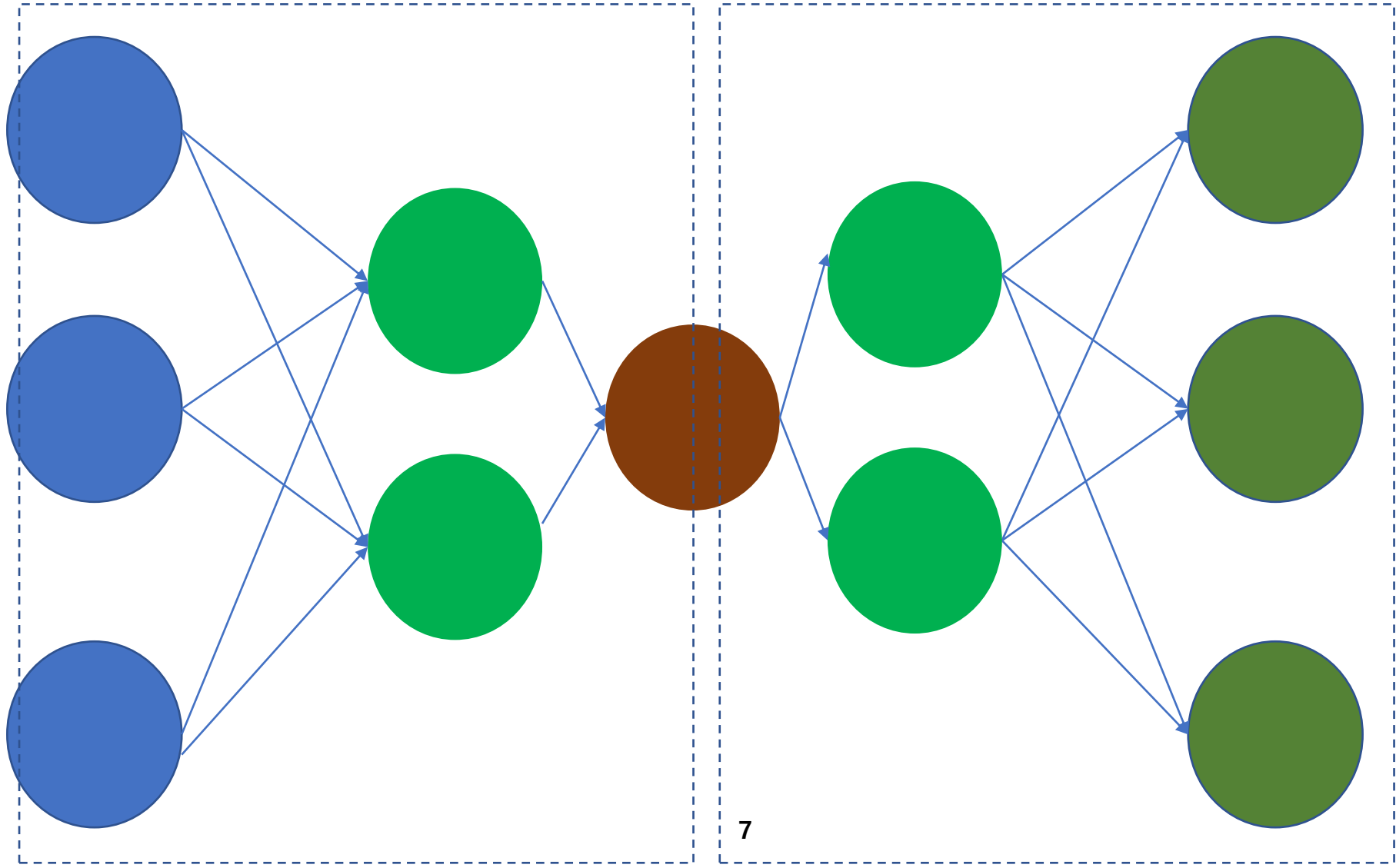


데이터 압축  
주요 Feature의 발견

# I. Autoencoder

## Stacked Autoencoder

- 입력에서 code까지 여러 은닉층을 구성하고
- Code에서 출력까지 대칭이 되도록 여러 은닉층을 구성



# I. Autoencoder

---

## Deep Belief Network(심층신뢰망)

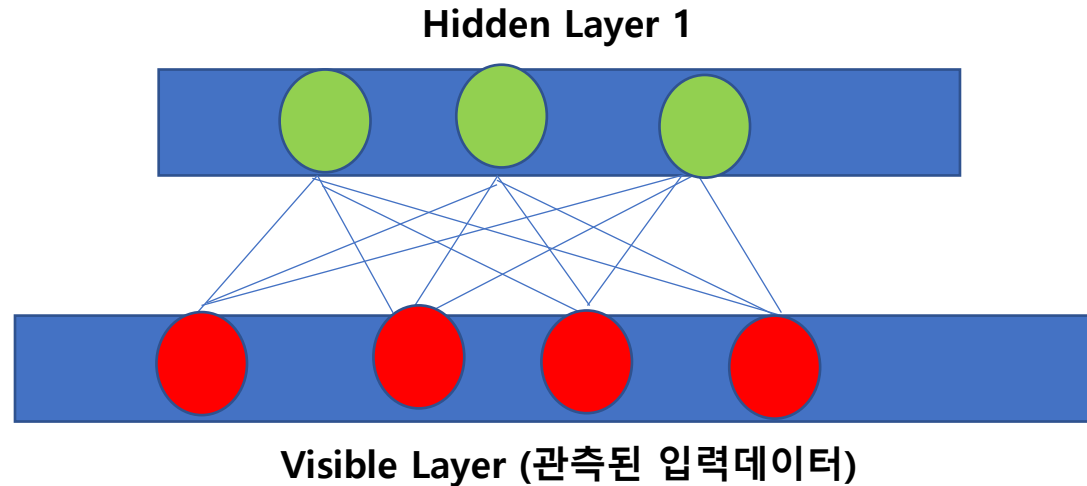
: 비지도 학습-입력데이터로만 학습, 학습 데이터가 충분하지 않는 경우 활용,  
RBM-사전 훈련된 RBM을 쌓아서 구성

- 은닉층과 입력층으로 구성된 RBM을 여러 층으로 쌓은 인공신경망
- RBM: Restricted Boltzmann Machine(제한된 볼츠만 머신)
- Feedforward 신경망에서의 Vanishing Gradient의 해결을 위해 고안



# I. Autoencoder

## 제한된 볼츠만 머신(RBM)



- 위와 같은 노드의 연결된 형태를 RBM이라고 함
- 다른 인공지능망의 학습에 활용
- 입력층(Visible Layer라고도 부름)과 은닉층 1개씩으로 구성
- RBM이 여러 개 쌓여서 DBN을 구성

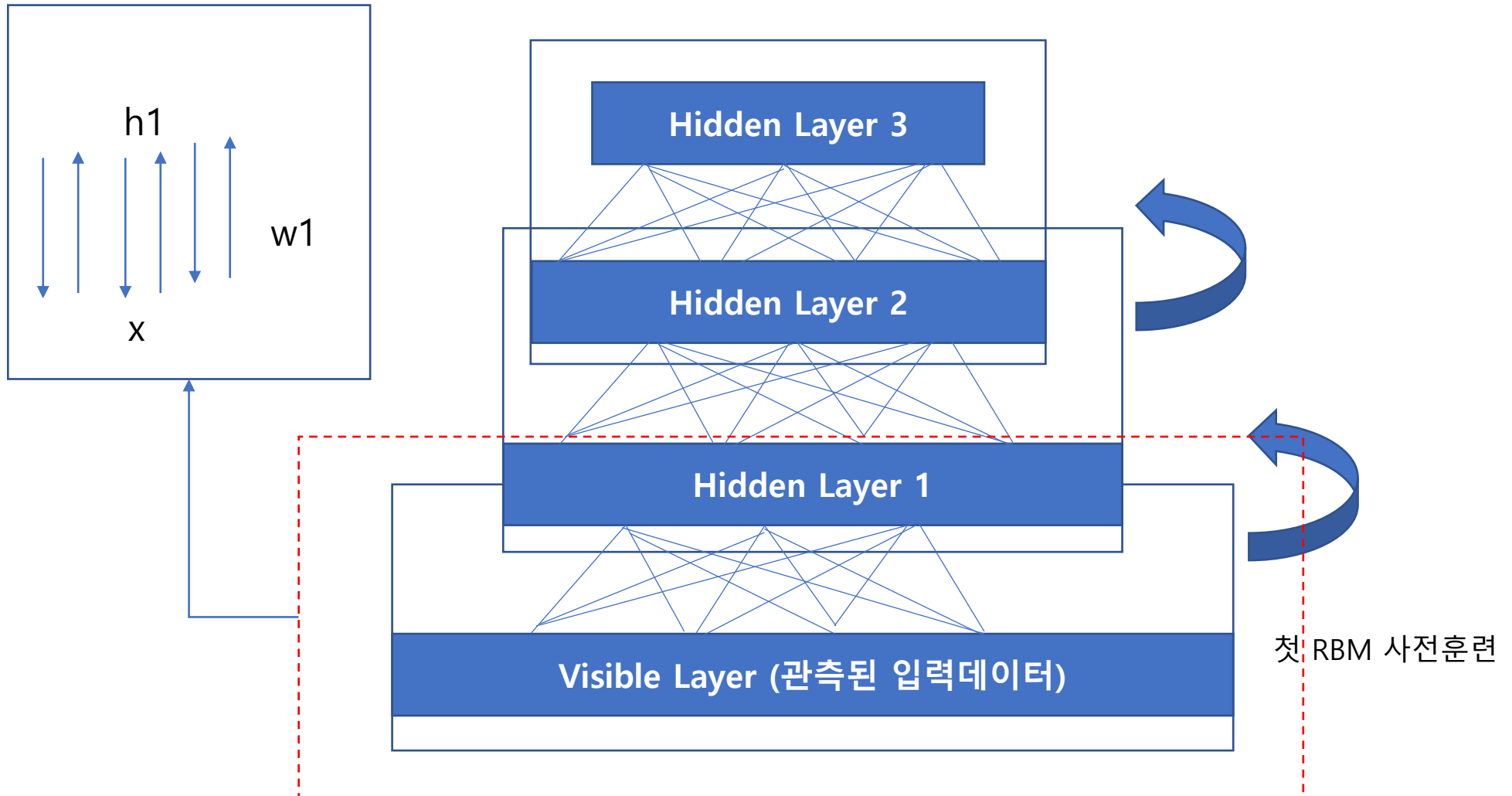
왜 “제한된”?

- 볼츠만 머신은 같은 층 내의 연결을 갖음
- 같은 층 내의 연결은 없기 때문!

# I. Autoencoder

## DBN의 구성1

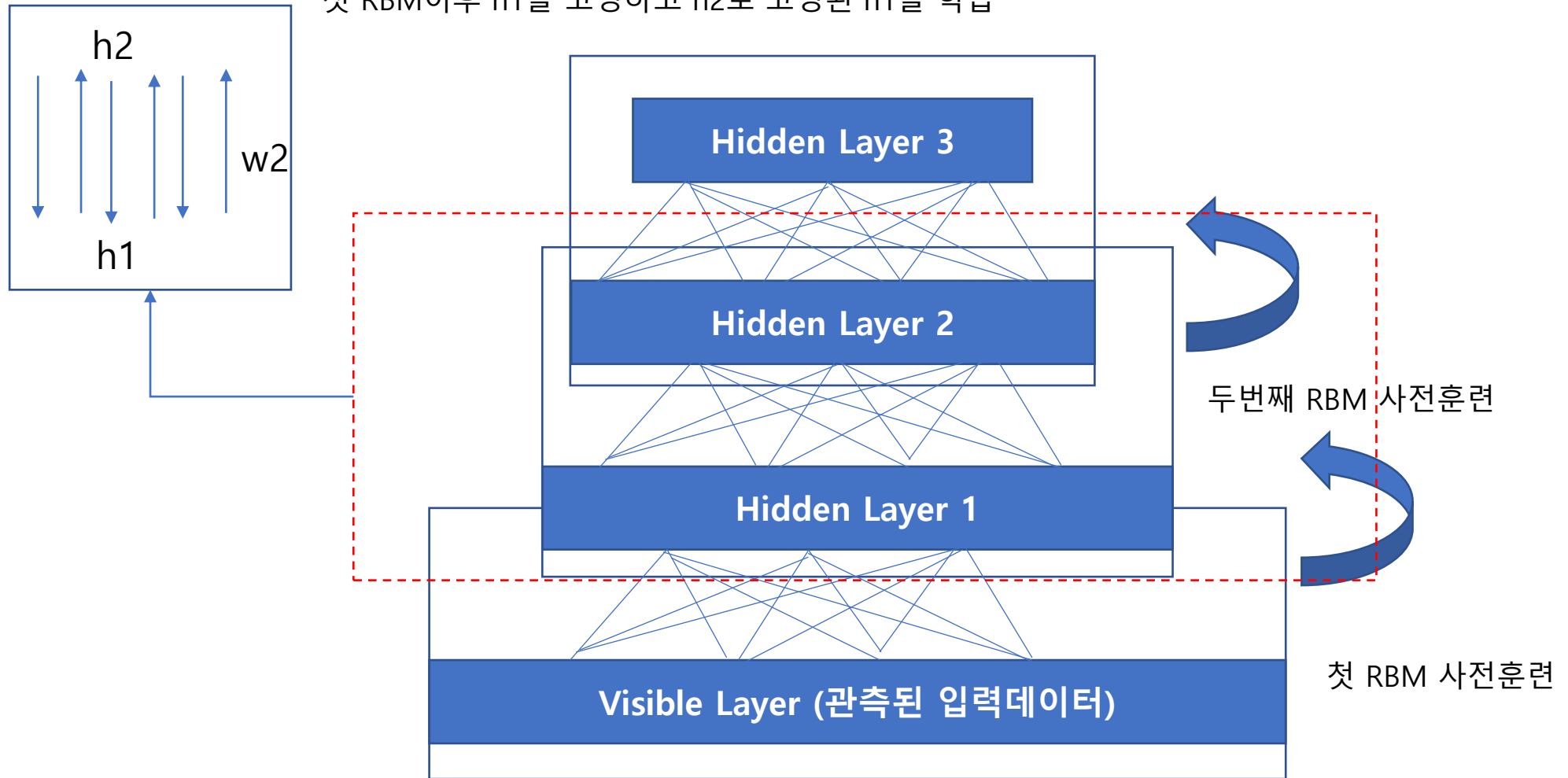
최초 랜덤한  $h1$ 으로  $x$ 를 학습



# I. Autoencoder

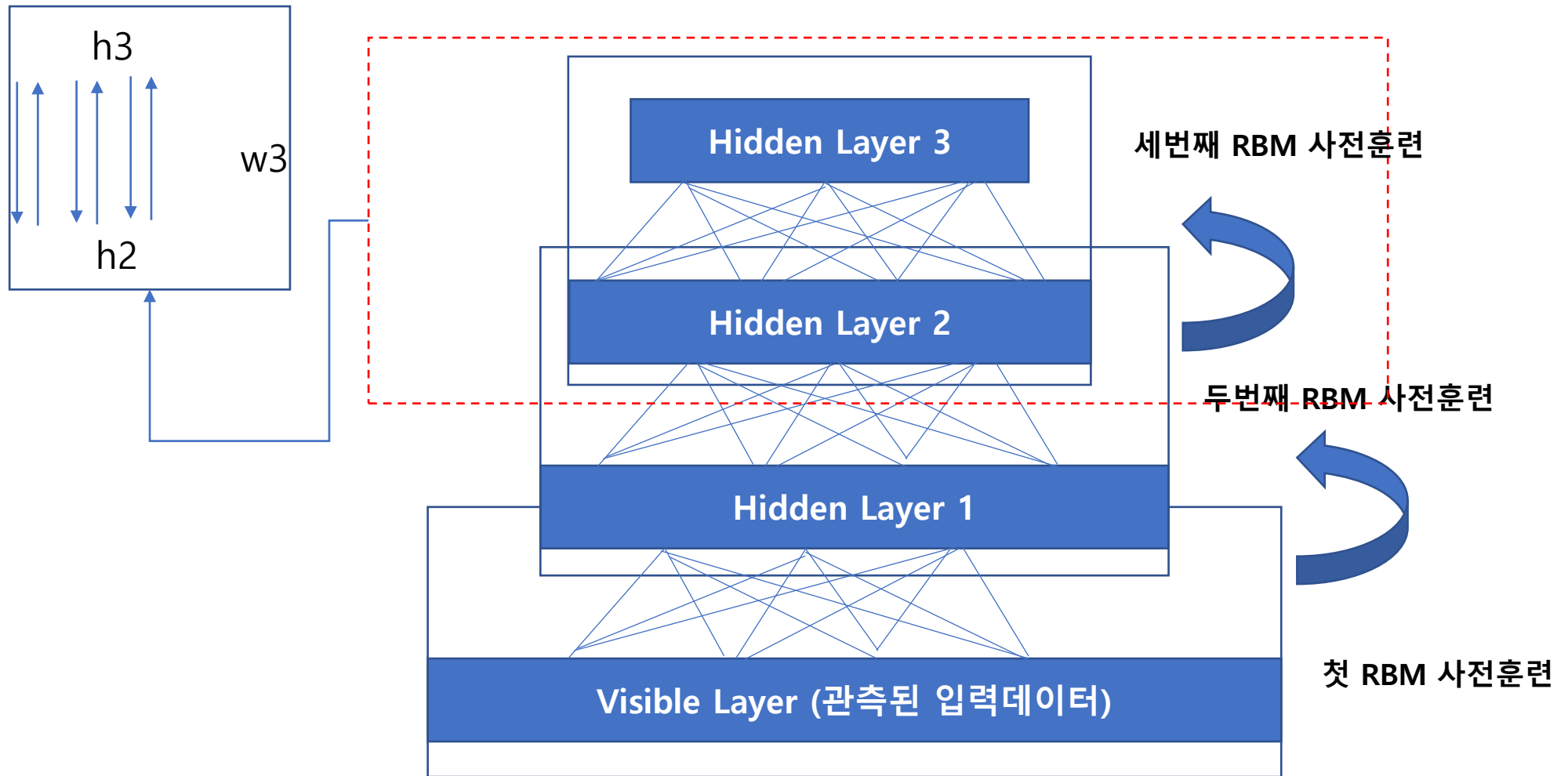
## DBN의 구성2

첫 RBM이후 h1을 고정하고 h2로 고정된 h1을 학습



# I. Autoencoder

## DBN의 구성3



### DBN의 활용

- 차원 감소
- 분류
- 협업필터링
- Feature Learning 등에 활용!

## II. 딥러닝과 자연어처리

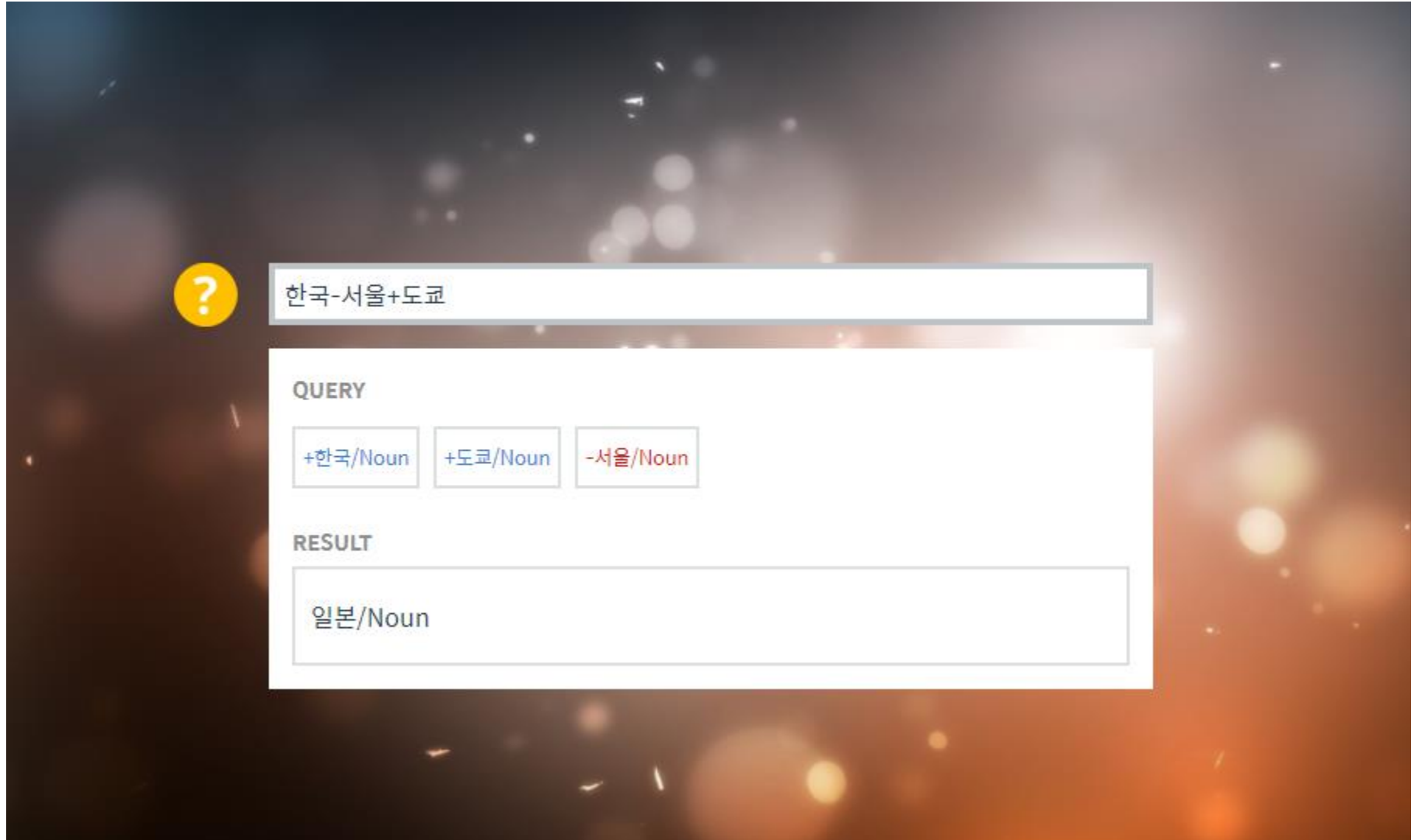
---

1. 임베딩(Embedding)과 워드2벡터: Embedding은 자연어처리에서 단어를 수치화하여 표현하며 다양한 방식이 있고, 워드2벡터는 단어를 벡터로 바꾸는 대표적인 임베딩 모형
2. 워드2벡터의 원리: 단어를 벡터로 변환하는 알고리즘으로 인공신경망 언어모형을 따르면서도 학습 속도와 성능을 개선하며, CBOW와 Skip Gram등의 기법이 있음
3. Word2Vec에서 Doc2Vec으로: 단어 임베딩 모형인 Word2Vec에 이어 발표된 Doc2Vec 모형은 Paragraph2Vec이라고도 지칭되며 문장/단락/문서에 대한 임베딩을 할 수 있음
4. Doc2Vec 원리: 다음 단어를 예측하며 로그 확률 평균을 최대화하고 문장/단락/문서의 임베딩 표현을 수행함. PV(Paragraph Vector)-DM(Distributed Memory)과 PV(Paragraph Vector)-DBOW(Distributed Bag of Words)의 두 가지 방식이 있음

## II. 딥러닝과 자연어처리

휴먼 + 알코올 = 강아지

휴먼 - 야근 = 만족



<https://word2vec.kr/search/>

## II. 딥러닝과 자연어처리

---

### 임베딩?

- 자연어처리에서 많이 활용
- 사람이 사용하는 자연어의 단어를 수치화하여 벡터로 표현한 결과가 그 과정
- 다양한 방법이 사용

### 워드2벡터(Word2Vec)

- 워드2벡터는 단어를 벡터로 바꾸는 대표적인 임베딩 모형
- 단어를 벡터로 변환하는 알고리즘으로 인공신경망 언어모형을 따르면서  
학습 속도와 성능을 개선
- 단어 유사도 계산/ 단어와 특정 쿼리의 유사도 계산 / 문장 분류



## II. 딥러닝과 자연어처리

### 워드2벡터(Word2Vec)의 대표적인 알고리즘: CBOW

Continuous Bag of Words

주변 단어(Context Word)로 중심의 단어(Center Word)를 예측

This cat jumps onto the chair

입력

출력

This cat

onto the chair

jumps

Context word

Center word

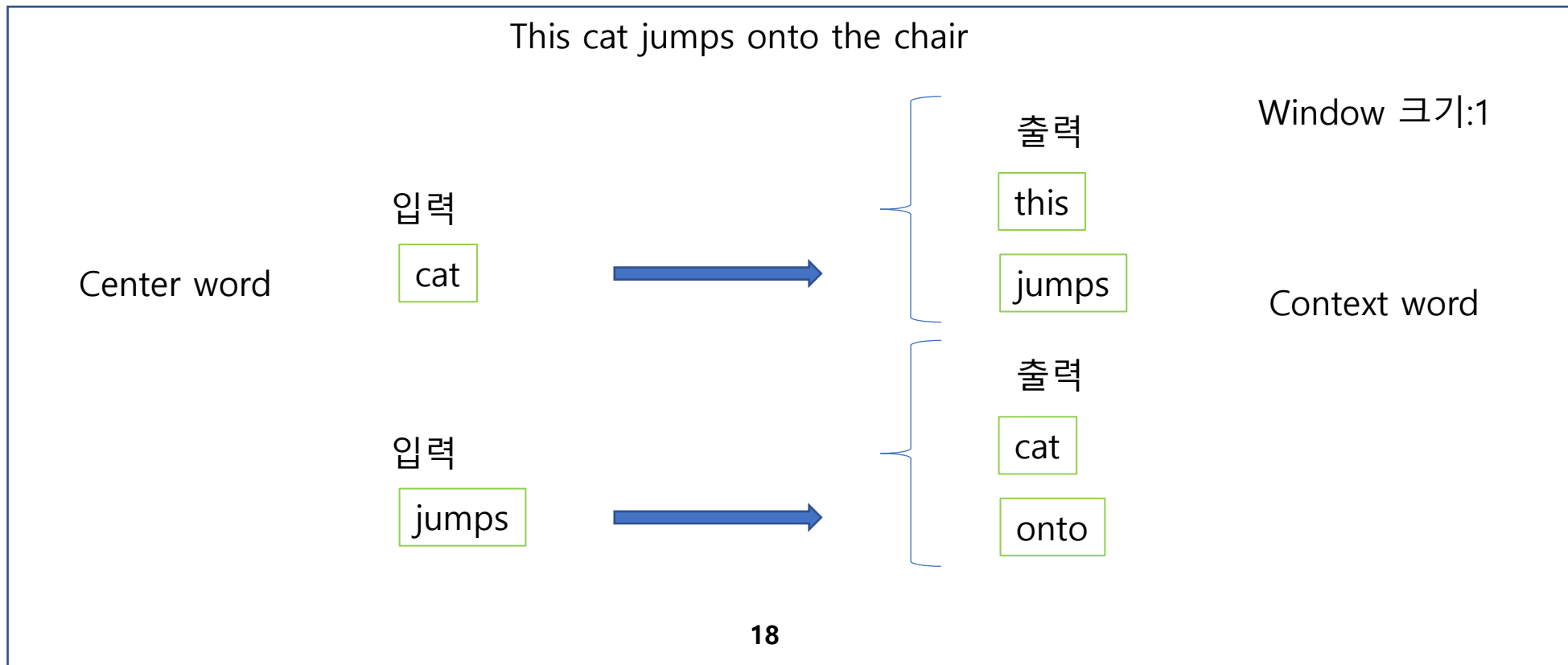
- Window: Center word 기준으로 앞 뒤 몇 개의 단어를 참고하는지
- Sliding Window: 중심 단어를 바꿔가며 Window 적용해서 학습데이터 생성

## II. 딥러닝과 자연어처리

### 워드2벡터(Word2Vec)의 대표적인 알고리즘: Skip Gram

#### *Skip Gram*

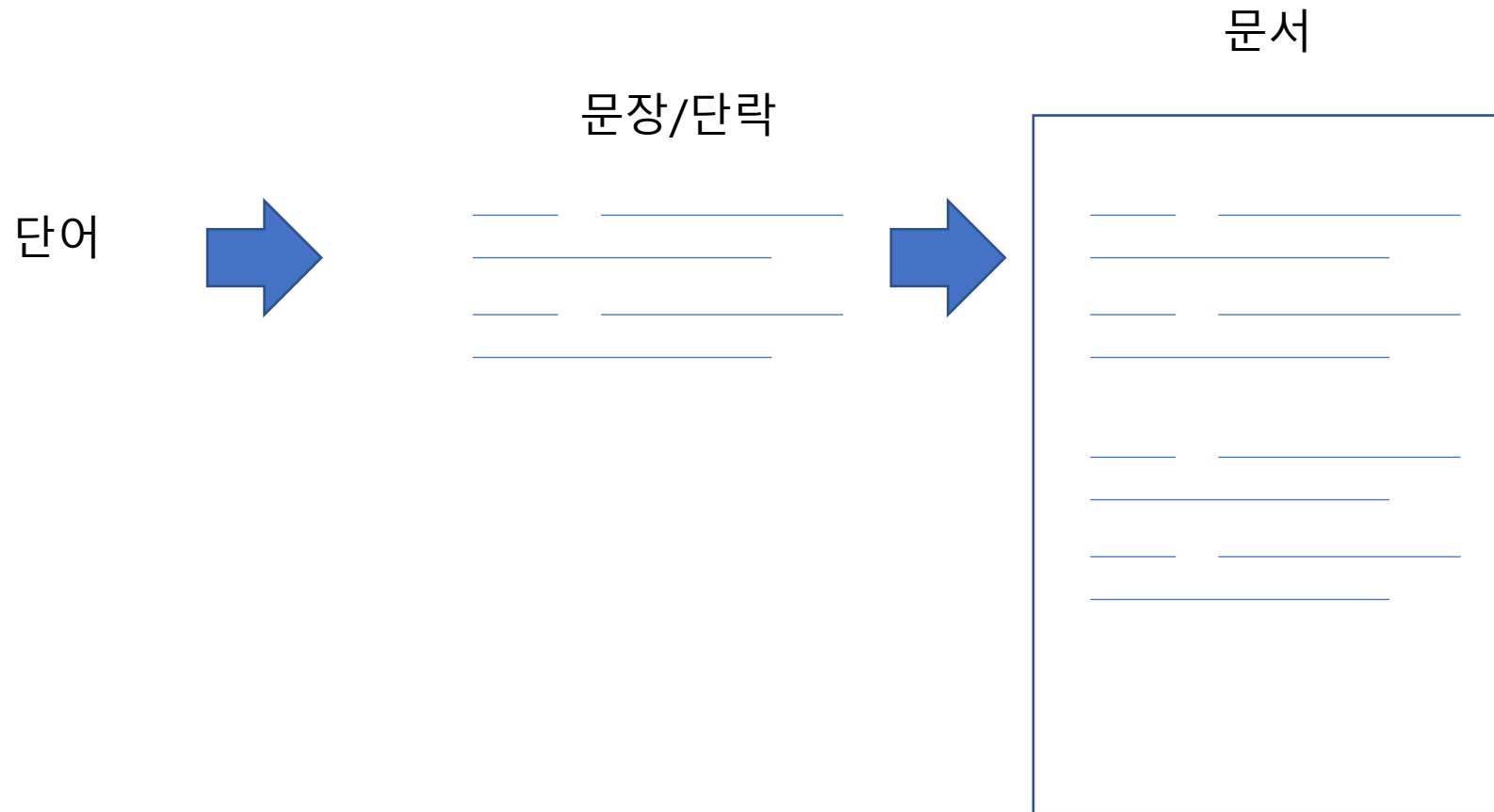
*중심 단어(Center Word)로 주변 단어(Context Word)를 예측*



## II. 딥러닝과 자연어처리

---

### Word2Vec에서 Doc2Vec으로



## II. 딥러닝과 자연어처리

---

### Doc2Vec이란?

- *Word2Vec에 이어 2014년 구글에서 개발한 모형*
- *다음 단어를 예측하며 로그 확률 평균을 최대화하고 문장/단락/문서의 임베딩 표현을 수행함.*

## II. 딥러닝과 자연어처리

---

### PV(Paragraph Vector)-DM(Distributed Memory), 성능 우수!

#### Paragraph ID: para\_1

Sentence: The dog sleep on the sofa

Y: Target 단어

X: Target 이전의 k개 단어+Paragraph ID

D2v는 단락에서 단어를 예측하며, 로그확률평균을 최대화하는 학습

예: k=2

X	Y
[para_1, the, dog]	sleep
[para_1, dog, sleep]	on
[para_1, sleep, on]	the
[para_1, on, the]	sofa

## II. 딥러닝과 자연어처리

---

### PV(Paragraph Vector)- DBOW(Distributed Bag of Words)

**Paragraph ID: para\_1**

Sentence: The dog sleep on the sofa

Paragraph ID가 입력, 문장의 단어들이 Target

X	Y
[para_1]	the
[para_1]	dog
[para_1]	sleep
[para_1]	on
[para_1]	the
[para_1]	sofa

## II. 딥러닝과 자연어처리

---

### Doc2Vec의 활용

- Q&A에 활용
- 기계번역에 활용
- 문장 주제 찾기 및 분류에 활용
- 사람과의 대화에 활용
- 다양한 파생모형!

## II. 딥러닝과 자연어처리

### Transformer

자연어 처리를 위한 발전된 모형들  
이며, Transformer는 LSTM없이  
Attention으로 Encoder와 Decoder  
를 구성한 것

2017년 구글의 논문,  
"Attention is all you need"

#### Attention Is All You Need

<b>Ashish Vaswani*</b> Google Brain avaswani@google.com	<b>Noam Shazeer*</b> Google Brain noam@google.com	<b>Niki Parmar*</b> Google Research nikip@google.com	<b>Jakob Uszkoreit*</b> Google Research usz@google.com
<b>Llion Jones*</b> Google Research llion@google.com	<b>Aidan N. Gomez* †</b> University of Toronto aidan@cs.toronto.edu	<b>Łukasz Kaiser*</b> Google Brain lukaszkaier@google.com	
<b>Illia Polosukhin* ‡</b> illia.polosukhin@gmail.com			

#### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

#### 1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

\*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and



## II. 딥러닝과 자연어처리

---

### BERT(Bidirectional Encoder Representations from Transformers)

BERT는 문맥의 앞과 뒤를 고려할 수 있는 양방향 Transformer

#### BERT

Pre-training of Deep Bidirectional Transformers for Language Understanding

Transformer 구조를 적극 활용

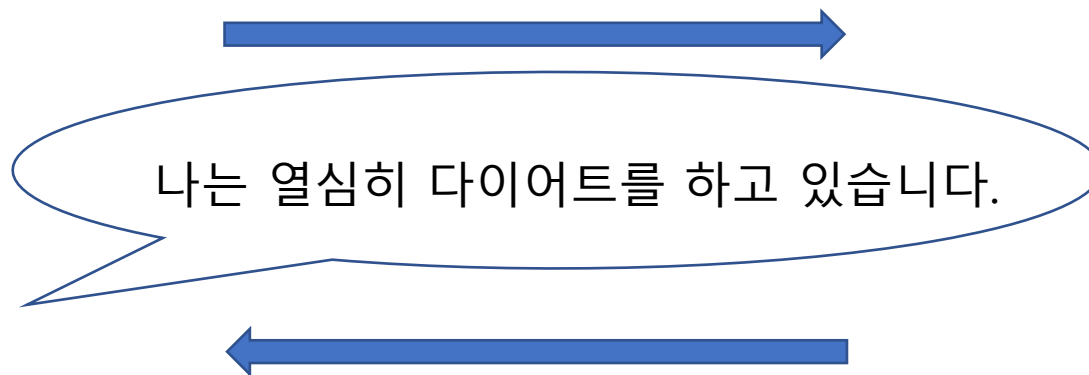
구글에서 개발한 NLP 사전 훈련 임베딩 모형

앞뒤 문맥정보를 활용!

## II. 딥러닝과 자연어처리

### BERT의 특징

- 양방향성!
- 문맥의 파악은 왼쪽에서 오른쪽으로 문장을 이해하는 것이 단방향성이며,
- 양방향성은 문장을 왼쪽에서 오른쪽으로 이해하고, 다시 오른쪽에서 왼쪽으로 이해하는 것을 의미



**문장의 특정 단어 이해는 앞뒤의 맥락(단어)을 보는 것이 중요!**

## II. 딥러닝과 자연어처리

---

### BERT의 방법

- Transformer 구조 중 Encoder만 사용
  - Self-attention layer를 여러 개 사용
  - 임베딩-Token / Sentence / position embedding을 활용
  - 언어모델링-Maked Language Model / Next Sentence Prediction
- 
- *Language Representation!*
  - *Pre-training이 가능:처음부터 다시 만들 필요가 없으며, Pre-training된 언어모형을 특정 task에 적용하고, fine tuning을 통해 사용함*
  - *ELMO, XLNet, GPT 등이 만들어짐!*

## II. 딥러닝과 자연어처리

---

### BERT의 활용

- Q&A에 활용
- 기계번역에 활용
- 문장 주제 찾기 및 분류에 활용
- 사람과의 대화에 활용
- 다양한 파생모형!

### III. GAN

---

1. GAN(Generative Adversarial Network): “경찰과 위조지폐범”의 예시로 알려진, 적대적 생성 네트워크. “가짜” 데이터를 생성하는 인공지능망 모형
2. GAN 특징: 인공지능망을 기반으로 실제 데이터 분포를 학습하는 생성자(Generator)와 생성자가 만든 데이터와 실제 데이터를 구분하는 판별자(Discriminator) 두 모형이 적대적으로 학습
3. GAN 활용: GAN을 통해 이미지/글/컨텐츠 등을 생성할 수 있음, 이미지 생성과 스타일 변환, 얼굴 이미지 합성 등에 많이 활용

### III. GAN

---

#### GAN이란

- GAN(Generative Adversarial Network)
- “가짜” 데이터를 생성하는 인공신경망 모형
- “경찰과 위조지폐범”의 예시
- 인공신경망 기반의 적대적 생성 모형

*Ian Goodfellow, 2014*

#### GAN특징

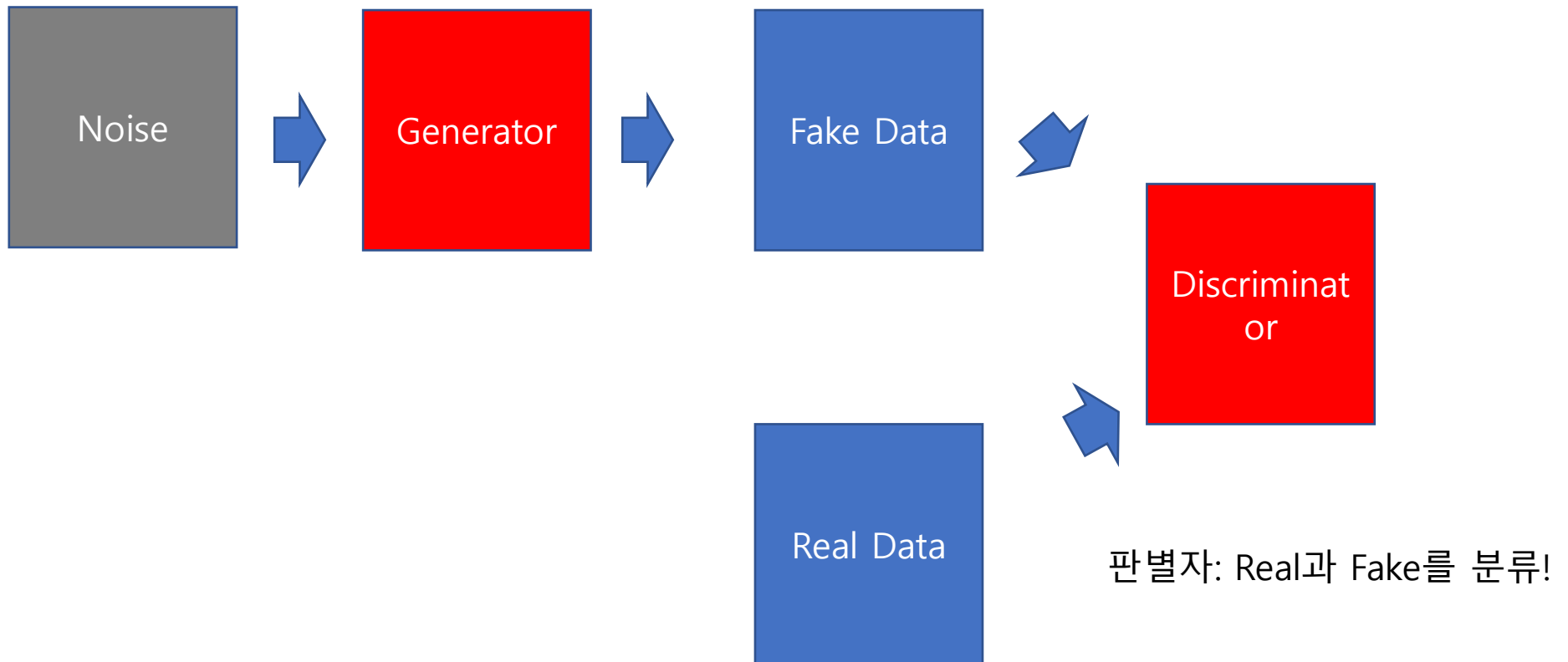
- 생성자와 판별자가 적대적으로 경쟁하며 학습
- 생성자(Generator): 실제 데이터 분포를 학습
- 판별자(Discriminator): 생성자가 만든 데이터(fake data)와 실제 데이터(real data)를 구분하는 모형

### III. GAN

---

#### GAN특징

생성자: D가 틀리도록 Fake를 생성!



## GAN활용: 이미지/음성/영상 등, Deep Fake

**다양한 GAN 모형들!**

- DCGAN(Deep Convolutional GAN)
- InfoGAN
- WassersteinGAN
- BEGAN
- starGAN
- StyleGAN
- .....



#### GAN 한계점

- 쉽지 않은 학습
  - Mode Collapse: 학습이 잘 진행되지 않는 현상
- 생성자와 판별자가 같이 발전해야 함
- 이미지/음성 등의 성공사례 VS 텍스트 생성 시 개선의 여지

## IV. 웹 시각화

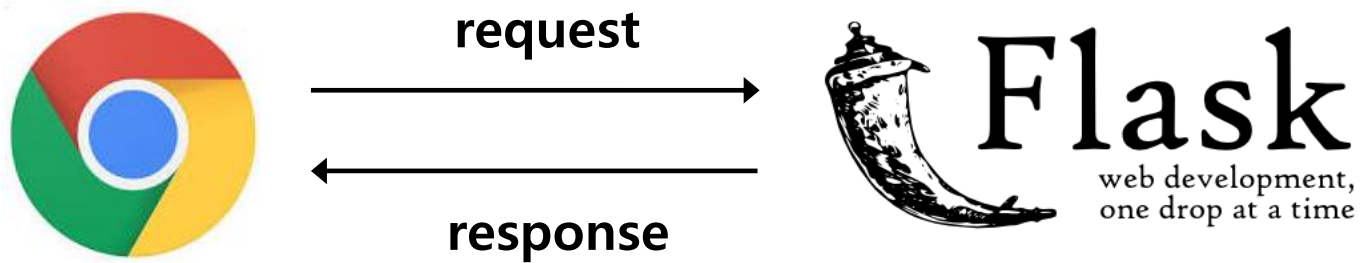
---

1. FLASK 소개
2. Python Anywhere 서비스
3. Dash 소개

## IV. 웹 시각화

---

- 웹 어플리케이션을 만드는 Framework



- flask 모듈에서 Flask 클래스 import
- Flask 클래스에 대한 인스턴스 생성

```
from flask import Flask  
app = Flask(__name__)
```

## IV. 웹 시각화

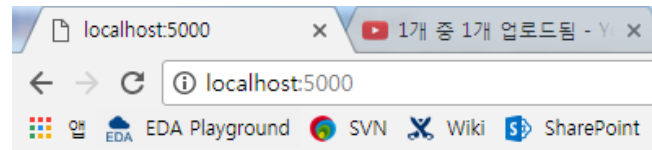
- 라우팅 : 데이터 접근 경로 지정

```
@app.route('/')  
def index( ):  
    return "<h1> Hello </h1>"
```

← 데이터 접근 경로



<http://127.0.0.1:5000/> 로 접근



Hello

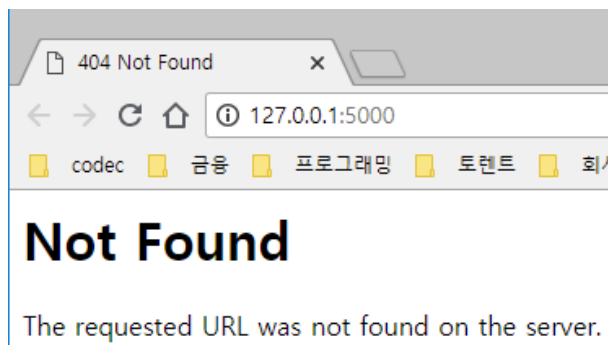
## IV. 웹 시각화

---

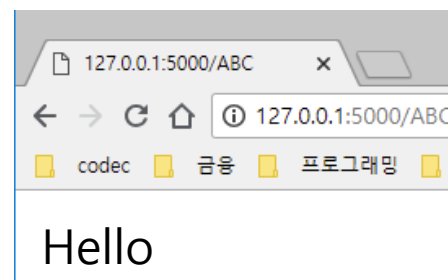
- 라우팅2

```
@app.route('/ABC')  
def index():  
    return "<h1> Hello! </h1>"
```

<http://127.0.0.1:5000/>



<http://127.0.0.1:5000/ABC>



## IV. 웹 시각화

---

- 서버 시작
  - 플라스크의 웹 서버 실행

```
app.run(debug=True)
```



```
from flask import Flask
app = Flask(__name__)

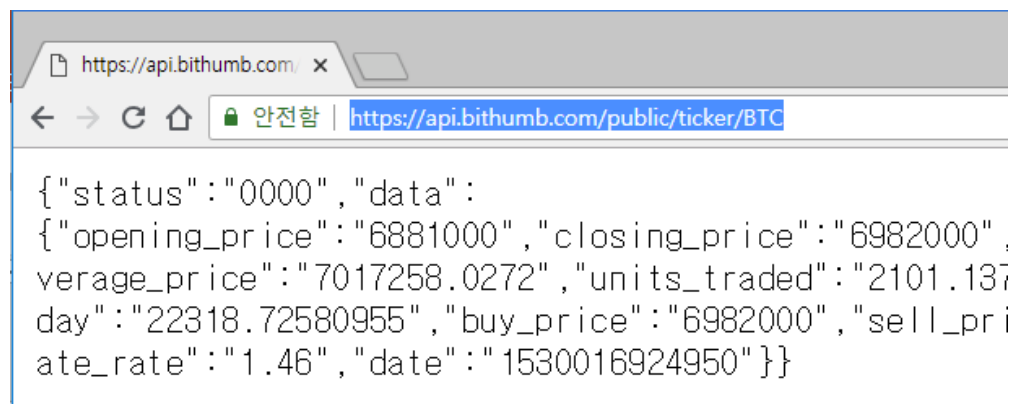
@app.route('/')
def index():
    return "<h1> Hello </h1>"

app.run(debug=True)
```

HTML 코드

### Dictionary 반환

- 예: <https://api.bithumb.com/public/ticker/BTC>



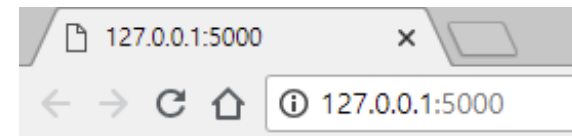
## IV. 웹 시각화

---

```
from flask import Flask, jsonify
app = Flask(__name__)

@app.route('/')
def index():
    data = {
        "icecream": 500,
        "snack": 1000,
    }
    return jsonify(data)

app.run(debug=True)
```



```
{
  "icecream": 500,
  "snack": 1000
}
```



### 동적 Routing

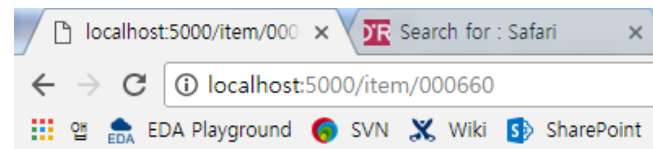
```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return "<h1> Hello </h1>"

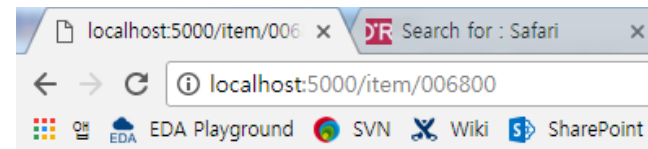
@app.route('/item/<code>')
def item(code):
    return "<h1> %s </h1>" % code

if __name__ == "__main__":
    app.run(debug=True)
```

변수



000660



006800

## IV. 웹 시각화

### Scraping + Flask

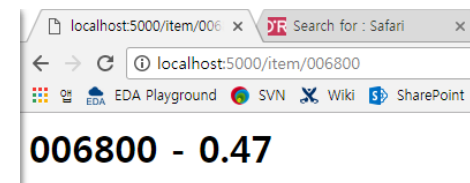
```
import requests
from bs4 import BeautifulSoup
```

```
def get_dividend_yield_ratio(code):
    url = http://finance.naver.com/item/main.nhn?code= + code
    html = requests.get(url).text
```

```
    soup = BeautifulSoup(html, "html5lib")
    p = soup.select("#_dvr")
    return p[0].text
```



```
@app.route('/item/<code>')
def item(code):
    dividend_yield_ratio = get_dividend_yield_ratio(code)
    return "<h1> %s - %s </h1>" % (code, dividend_yield_ratio)
```



## IV. 웹 시각화

- Pandas+Scraping+Flask

```
@app.route('/table/<code>')
def index(code):
    url = "http://finance.naver.com/item/sise_day.nhn?code={}&page=1".format(code)
    df = pd.read_html(url)
    return df[0].to_html()
```

일별 시세						
날짜	종가	전일비	시가	고가	저가	거래량
2018.06.26	84,300	▼ 500	82,500	84,800	82,400	5,378,361
2018.06.25	84,800	▼ 4,700	88,800	88,800	84,600	4,420,542
2018.06.22	89,500	▲ 1,000	87,300	89,500	86,700	2,569,978
2018.06.21	88,500	▲ 700	88,900	89,300	87,500	3,274,612
2018.06.20	87,800	▲ 3,700	85,800	88,100	85,000	4,183,210
2018.06.19	84,100	▲ 100	84,100	86,100	83,700	3,824,878
2018.06.18	84,000	▼ 3,000	86,800	86,900	82,300	4,723,914
2018.06.15	87,000	▲ 700	87,400	87,800	86,100	2,945,006
2018.06.14	86,300	▼ 2,100	88,300	88,300	86,300	4,646,948
2018.06.12	88,400	▼ 800	89,900	90,100	88,200	1,876,012

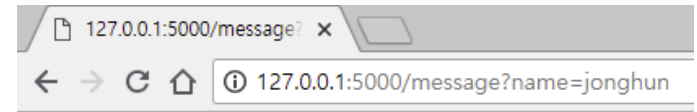
	0	1	2	3	4	5	6
0	날짜	종가	전일비	시가	고가	저가	거래량
1	2018.06.26	84300	500	82500	84800	82400	5378361
2	2018.06.25	84800	4700	88800	88800	84600	4420542
3	2018.06.22	89500	1000	87300	89500	86700	2569978
4	2018.06.21	88500	700	88900	89300	87500	3274612
5	2018.06.20	87800	3700	85800	88100	85000	4183210
6	2018.06.19	84100	100	84100	86100	83700	3824878
7	2018.06.18	84000	3000	86800	86900	82300	4723914
8	2018.06.15	87000	700	87400	87800	86100	2945006
9	2018.06.14	86300	2100	88300	88300	86300	4646948
10	2018.06.12	88400	800	89900	90100	88200	1876012

## IV. 웹 시각화

### "GET" 데이터 처리

- [http://127.0.0.1:5000/message? name=hello](http://127.0.0.1:5000/message?name=hello)

```
from flask import Flask, request
@app.route('/message')
def message():
    user = request.args.get('name')
    return "<h1> Hi, %s </h1>" % user
```



Hello

- methods를 생략하면 기본적으로 GET이 선택

```
@app.route('/message', methods=['GET'])
def message():
    user = request.args.get('name')
    return "<h1> Hi, %s </h1>" % user
```

- **Post Methods : 프로그램을 통해 데이터 보내기**
  - `requests.post(url, param)`

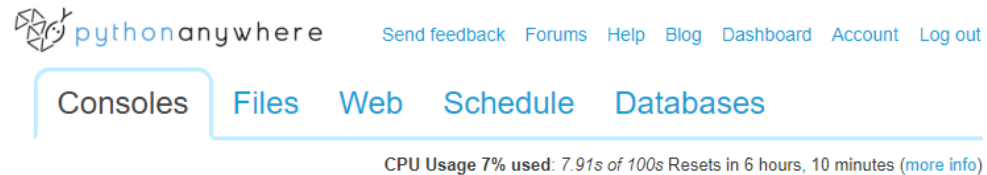


## IV. 웹 시각화

---

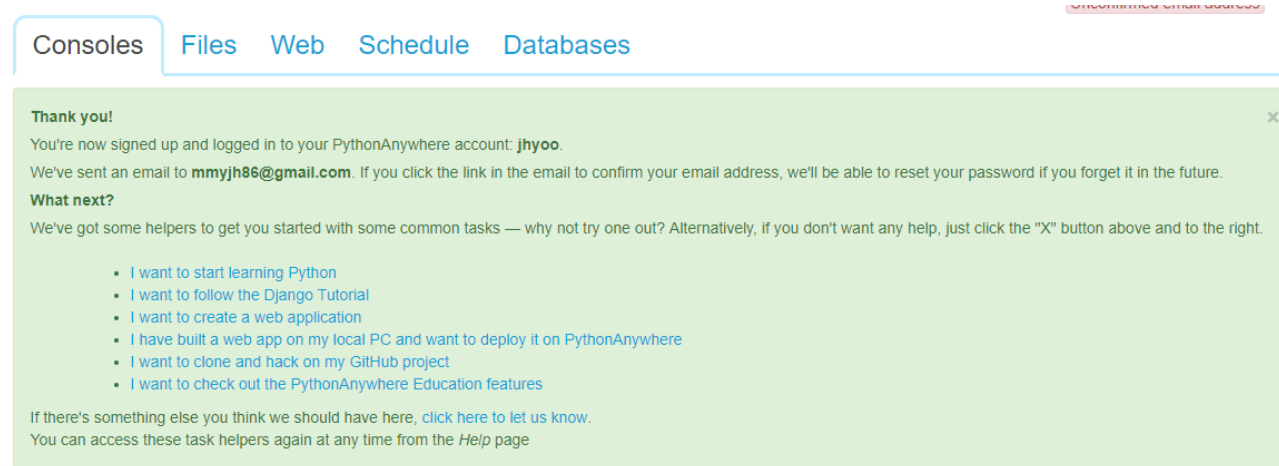
### 개인 웹 서버

- <https://www.pythonanywhere.com/>
  - 소규모 애플리케이션 위한 웹 서비스 제공
    - Online에서 파이썬 실행
    - 서브-도메인 서비스 제공
    - 무료



## IV. 웹 시각화

- 계정 생성 등

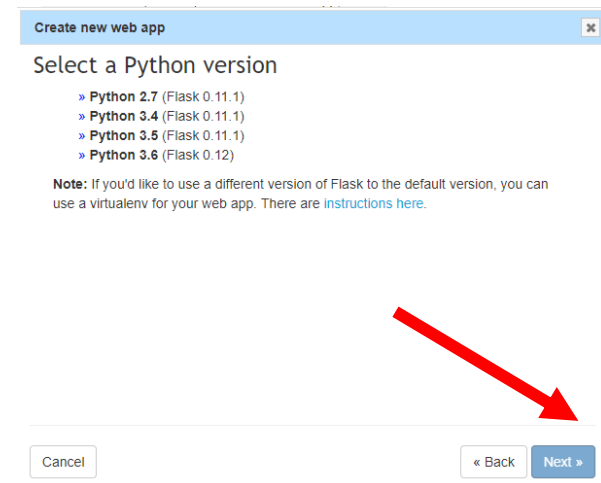
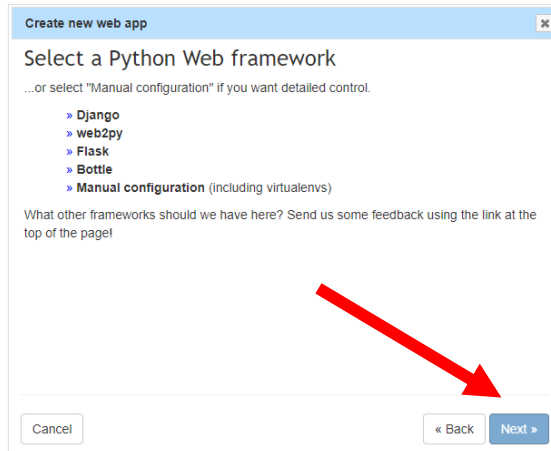
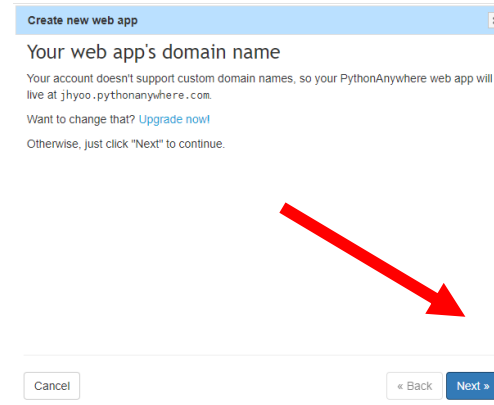
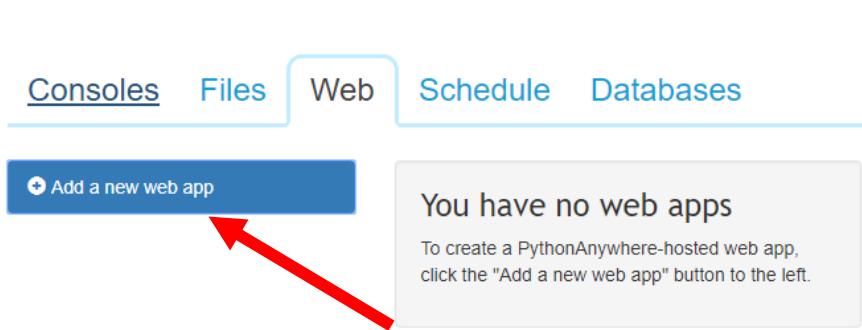


### Python Anywhere+flask Web App

- 웹 앱 (Web App) 이란?
  - 온라인으로 접속 가능한 애플리케이션
  - 프로그램의 실행/설치없이 웹브라우저에서 결과를 확인
  - 웹 서버에서 자동으로 동작

## IV. 웹 시각화

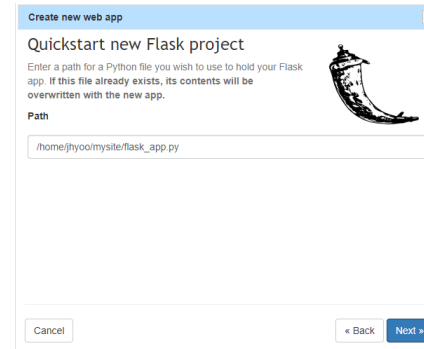
### Python anywhere+Web App



## IV. 웹 시각화

### Web App 만들기

- 파이썬 Flask 실행 파일 경로
  - 기본 경로 사용



Create new web app

Quickstart new Flask project

Enter a path for a Python file you wish to use to hold your Flask app. If this file already exists, its contents will be overwritten with the new app.

Path

/home/jhyoo/mysite/flask\_app.py

Cancel Back Next

All done! Your web app is now set up. Details below.

jhyoo.pythonanywhere.com

+ Add a new web app

Configuration f 생성한 ID. [pythonanywhere.com](https://pythonanywhere.com)

Reload:

Reload jhyoo.pythonanywhere.com

Best before date:

Free sites have a limited lifespan, but you can renew that here up to a maximum of three months from today's date. You can always extend it later too! We'll send you an email a week before it expires. [See here for more details.](#)

This site will be disabled on **Thursday 02 November 2017**

Run until 3 months from today

[Paying users'](#) sites do not have expiry dates.



## IV. 웹 시각화

- 코드 수정: Files 탭 -> mysite Directories -> flask\_app.py

The screenshot displays the PythonAnywhere web interface. At the top, there are three tabs: 'Consoles', 'Files', and 'Web'. The 'Files' tab is selected. Below the tabs, the breadcrumb path is '/ home / jhyoo'. Under the 'Directories' section, there is a list of directories: '.local/', '.virtualenvs/', and 'mysite/'. A red arrow points from the 'Files' tab to the 'Directories' section, and another red arrow points from the 'mysite/' directory to the 'Files' section. In the 'Files' section, there is a text input field 'Enter new file name, eg hello.py' and a list of files. The file 'flask\_app.py' is listed with a download icon, a share icon, a delete icon, and a timestamp '2017-08-06 01:59' and size '205 bytes'. A red arrow points to the 'flask\_app.py' file. Below the 'Files' section, there is a large green arrow pointing down to the code editor. The code editor shows the PythonAnywhere logo and the breadcrumb path '/ home / 본인 ID / mysite / flask\_app.py'. The code content is as follows:

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello_world():
7     return 'Hello from Flask!'
```

At the bottom of the code editor, there are several buttons: 'Send feedback', 'Forums', 'Help', 'Blog', 'Dashboard', 'Account', and 'Log out'. A red arrow points to the 'Dashboard' button. Below these buttons, there is a 'Keyboard shortcuts' section with a dropdown menu set to 'Normal', a 'Share' button, a 'Save' button (highlighted in green), a 'Save as...' button, a '>>> Run' button, and a refresh icon. A red arrow points to the 'Save' button, and another red arrow points to the refresh icon.

## IV. 웹 시각화

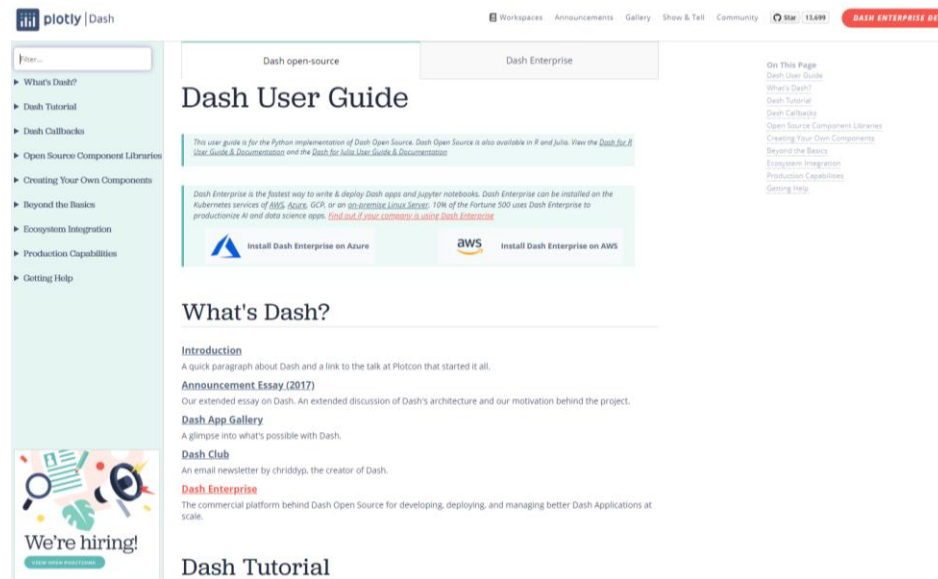
### Dash

파이썬에서 웹 애플리케이션 구현을 위한 프레임워크

Flask, Plotly.js, React.js 기반으로 개발

데이터 시각화를 위한 라이브러리

인터랙티브 웹 애플리케이션을 지원



<https://dash-gallery.plotly.host/dash-oil-and-gas>

## IV. 웹 시각화

---

### 기본 사용법

App 생성: Dash 함수

App Layout 구성: 생성된 App의 layout을 지정

App 실행: `run_server` 함수로 실행하며, Localhost로 확인

Localhost는 네트워크에서 현재의 컴퓨터 의미

## IV. 웹 시각화

---

### 1: App 생성

```
import dash
import dash_core_components as dcc
import dash_html_components as html

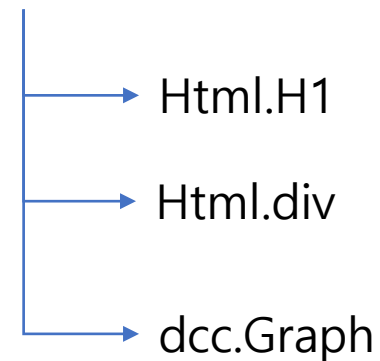
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
app
```

## IV. 웹 시각화

### 2: app의 layout구성

```
app.layout = html.Div(children=[  
    html.H1(children='Hello'),  
    html.Div(children=''  
        Dash Test''),  
    dcc.Graph(  
        id='example',  
        figure={  
            'data':[  
                {'x':[1,2,3], 'y':[3,2,1], 'type':'bar','name':'AAA'},  
                {'x':[1,2,3], 'y':[1,2,3], 'type':'bar','name':'BBB'},  
            ],  
            'layout':{  
                'title':'Dash'  
            }  
        })  
])
```

Html.div



## IV. 웹 시각화

---

### 3: 실행하기

```
if __name__ == '__main__':  
    app.run_server(debug=True)
```

```
if __name__ == '__main__':  
    app.run_server(debug=True, use_reloader=False)
```

## IV. 웹 시각화

---

- 전체 코드

```
import dash
import dash_core_components as dcc
import dash_html_components as html

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div(children=[
    html.H1(children='Hello'),
    html.Div(children='''
        Dash Test'''),
    dcc.Graph(
        id='example',
        figure={
            'data': [
                {'x': [1,2,3], 'y': [3,2,1], 'type': 'bar', 'name': 'AAA'},
                {'x': [1,2,3], 'y': [1,2,3], 'type': 'bar', 'name': 'BBB'}
            ],
            'layout': {
                'title': 'Dash'
            }
        })
])

if __name__ == '__main__':
    app.run_server(debug=True, use_reloader=False)
```

## IV. 웹 시각화

### • 전체 코드2

```
import dash
import dash_core_components as dcc
import dash_html_components as html

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div(children=[
    html.H1(children='Baby Robo Advisor'),
    html.Div(children='''
        Test'''),
    dcc.Graph(
        id='example',
        figure={
            'data':[
                {'x':df.index[0:5], 'y':df.values[0:5,3], 'type':'bar','name':'Samsung'},
            ],
            'layout':{
                'title':'Stock Prices'
            }
        })
])

if __name__ == '__main__':
    app.run_server(debug=True, use_reloader=False)
```



## IV. 웹 시각화

- Colab에서의 Dash!

- Shell 에서의 설치 및 실행!

```
%%sh
```

```
pip install -q dash  
pip install -q dash_core_components  
pip install -q dash_html_components  
pip install -q dash_table
```

```
%%sh
```

```
# get ngrok
```

```
curl -O https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip  
unzip ngrok-stable-linux-amd64.zip
```

```
%%writefile dash_app.py
```

```
Dash 실행코드!
```

```
# launch ngrok
```

```
get_ipython().system_raw('./ngrok http 8050 &')
```

```
%%sh
```

```
# get url with ngrok
```

```
curl -s http://localhost:4040/api/tunnels | python3 -  
c "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

```
!python dash_app.py
```

## IV. 웹 시각화

---

### Dash+작업 스케줄러

*스케줄러? 미리 정의 된 시간 또는 지정된 시간 간격에  
지정된 컴퓨터 프로그램 또는 스크립트의 실행을 예약하는 기능!*

#### 배치 방식으로 파이썬 실행

실행할 코드를 포함한 batch1.py 파일 생성

CMD로 위의 파일 실행

위 내용을 작업 스케줄러에 등록

작업스케줄러로 웹 브라우저 실행

## IV. 웹 시각화

---

### 예: 주가 시계열 분석

```
from pykrx import stock
from statsmodels.tsa.arima_model import ARIMA

ticker1="005930"
df = stock.get_market_ohlcw_by_date("20200122", "20210121", ticker1)
df.columns = ['open','high', 'low', 'close','volume']

model = ARIMA(df.close.values, order=(1,1,1))
model_fit = model.fit()
print(model_fit.summary())

#1시점 앞을 예측
forecast_data = model_fit.forecast(steps=1)
```

## IV. 웹 시각화

---

### Dash 시각화

```
import dash
import dash_core_components as dcc
import dash_html_components as html

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

result = 'Down'
if df.tail(1).close.values < forecast_data[0]:
    result='Up'

app.layout = html.Div(children=[
    html.H1(children='Roboadvisor'),
    html.Div(children='''My Stock Price'''),
    dcc.Graph(
        id='example',
        figure={
            'data':[
                {'x':['D-2','D-1','D'], 'y':df.tail(3).close.values, 'type':'bar','name':ticker1},
            ],
            'layout':{
                'title':ticker1
            }
        },
    ),
    html.H1(children='''Prediction'''),
    html.H2(children=forecast_data[0]),
    html.H2(children=result)
])

if __name__ == '__main__':
    app.run_server(debug=True, use_reloader=False)
```

---

**QnA**