



(Summer School) 딥러닝 기반 Neuroimaging 데이터 분석

# 텐서플로우(TensorFlow) 기초 및 실습

\*본 실습은 [TensorFlow v1.9](#) 기준으로 진행됩니다.

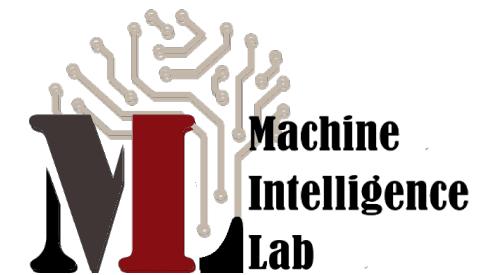


## 조교: 윤지석

강사: 석홍일

wltjr1007@korea.ac.kr

<http://www.ku-milab.org>

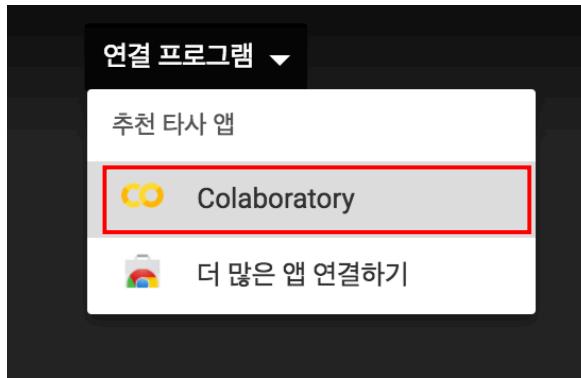


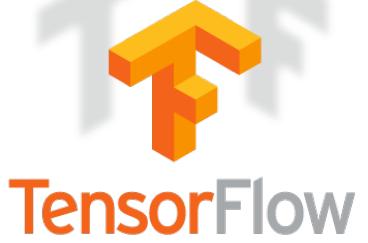
고려대학교 뇌공학과  
기계지능연구실

2018년 8월 18일

# Google Colaboratory 실습 환경 준비

1. drive.google.com 로그인
2. <https://goo.gl/M1bqJJ> 접속
3. Tutorial 1-1 더블 클릭 (연결 프로그램 – Colaboratory 클릭)
4. 왼쪽 상단 “실습 모드에서 열기” 클릭
5. 셀 실행 (각 셀의 재생 버튼 또는 단축키 사용)



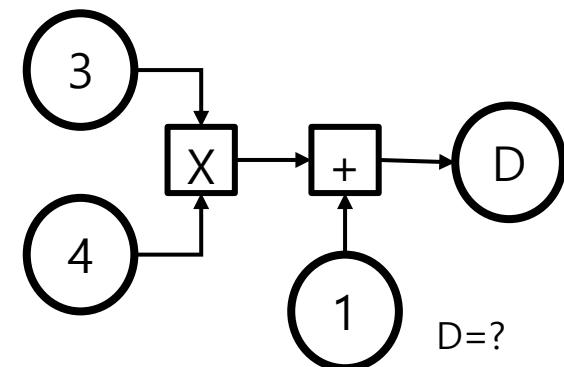
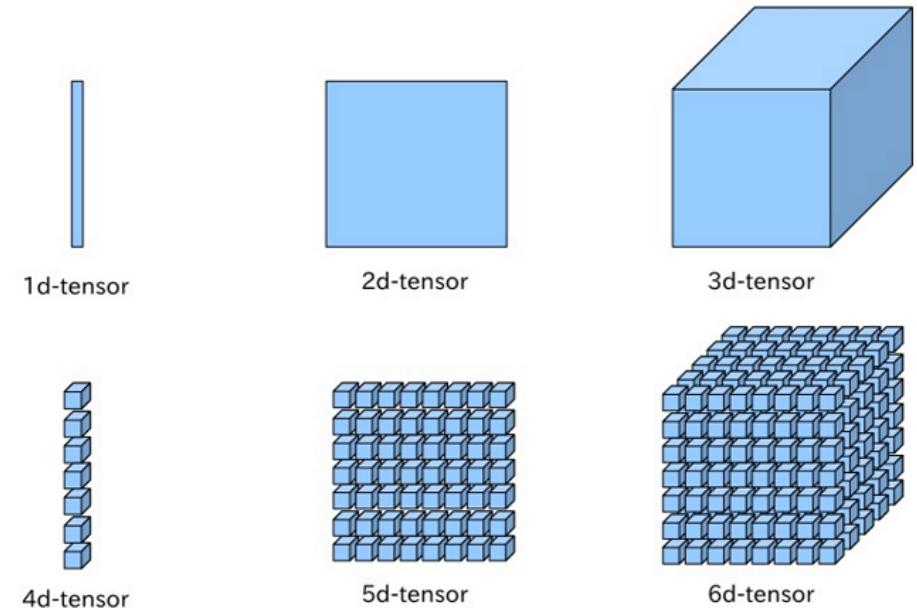


## 계산식을

**Tensor:** 텐서로 형성된 그래프로

**Flow:** 흘려 보내

풀어주는 딥러닝 라이브러리



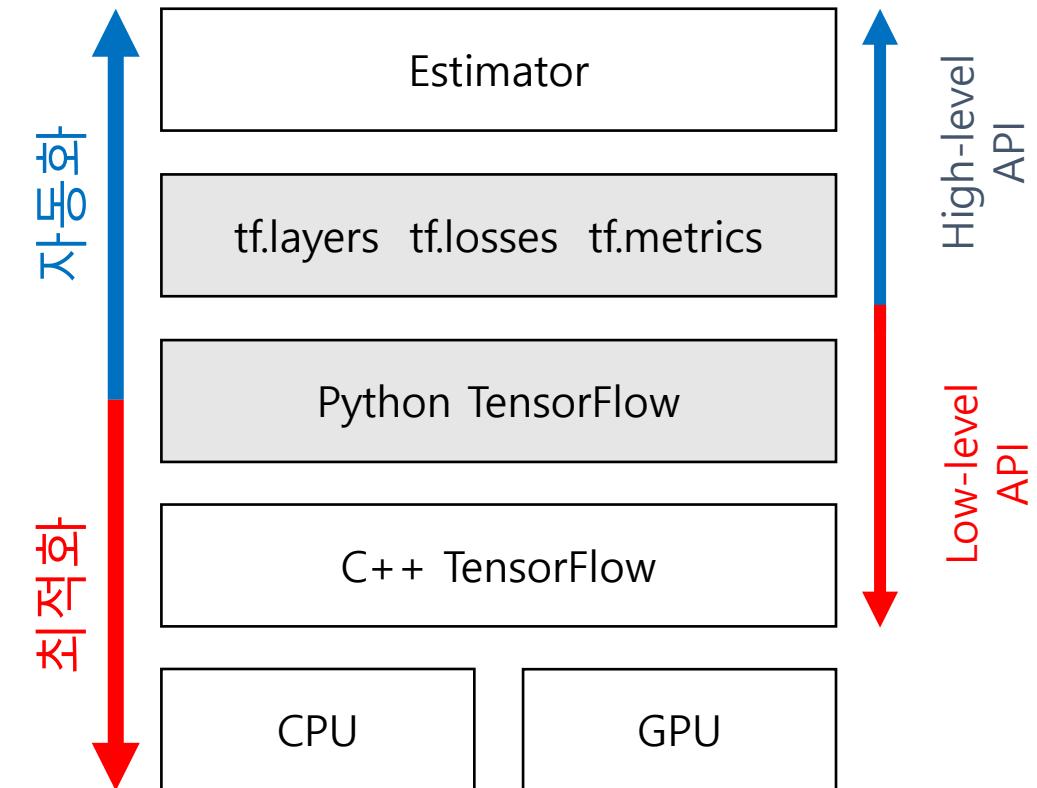
# TensorFlow 구성

**API** (Application Programming Interface)  
CPU 및 GPU에서 사용될 미리 정의된 명령어

TensorFlow API: `tf.add(A, B)`



CPU / GPU:  $A + B$



# TensorFlow의 프로그래밍 구조

## Graph

계산식 구축

예: ReLu , Logit Layer

## Session

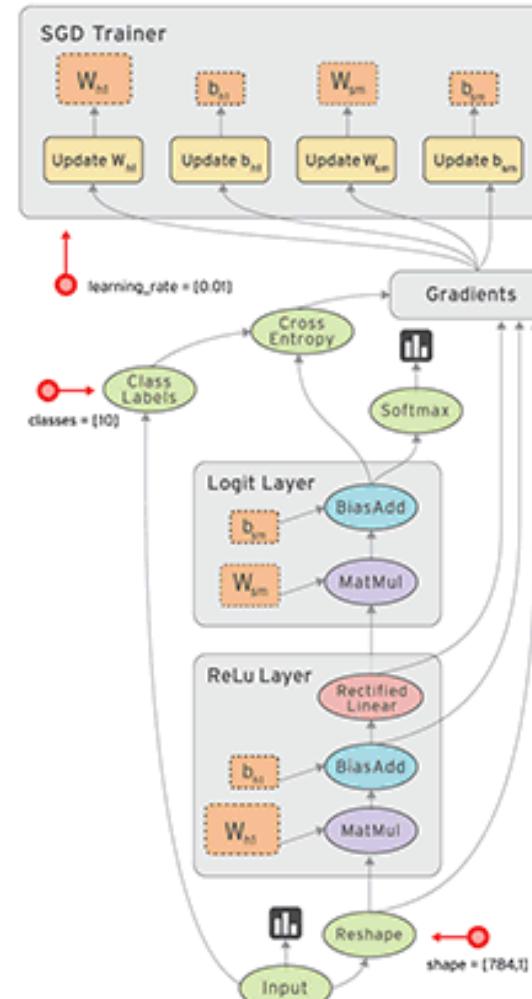
계산 실행

예: Tensor의 입력 및 출력

## Trainer

가중치 갱신

예: SGD (Stochastic Gradient Descent)



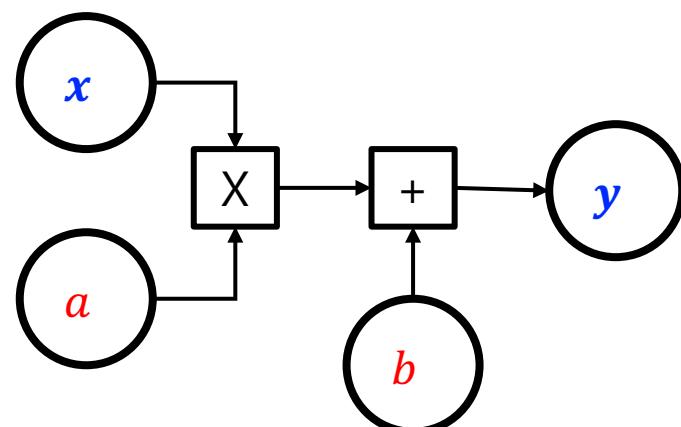
# Tensor and Variable

TensorFlow에서 가장 하위 자료구조

Tensor는 크게 **가변형** 과 **불변형**으로 나뉜다  
(가변형 Tensor은 **Variable**이라고 불린다)

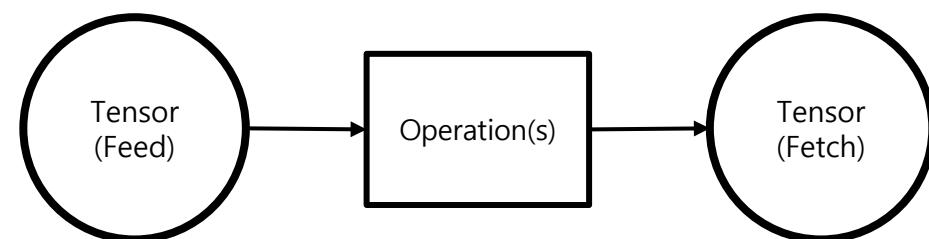
예: 선형 회기 (Linear Regression)

$$y = ax + b$$



# Session / 입출력

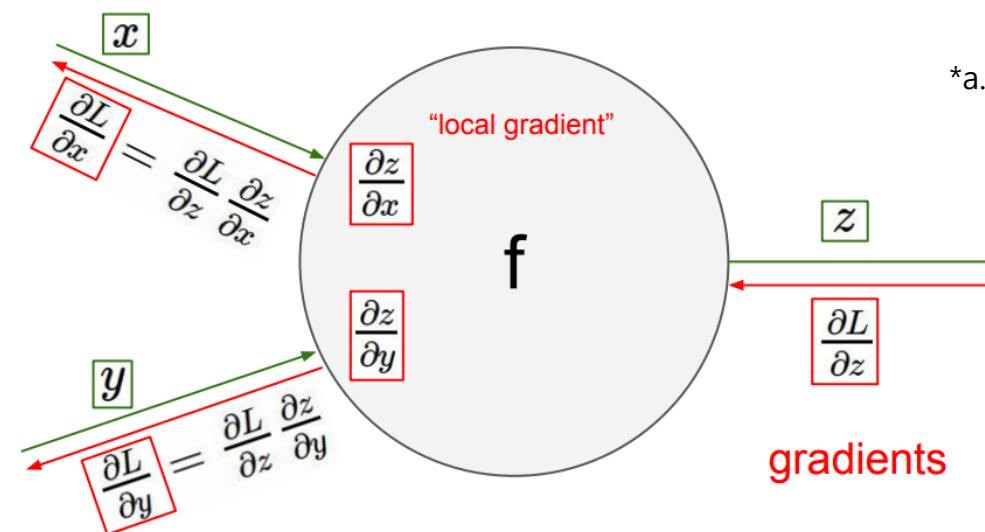
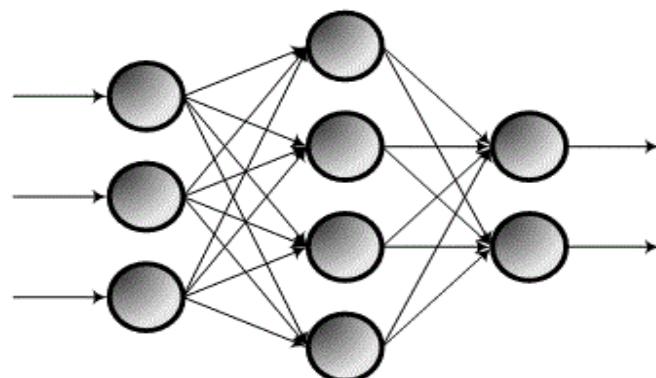
- Session – Tensor에 대한 계산 실행
  - 즉, operation을 실행
  - Feed tensor를 입력으로 받고, fetch tensor를 출력
    - 예: Result = Session.run(fetch, feed)
- Feed Tensor – tf.placeholder
  - 입력을 받을 수 있는 tensor 생성 방법



# Trainer

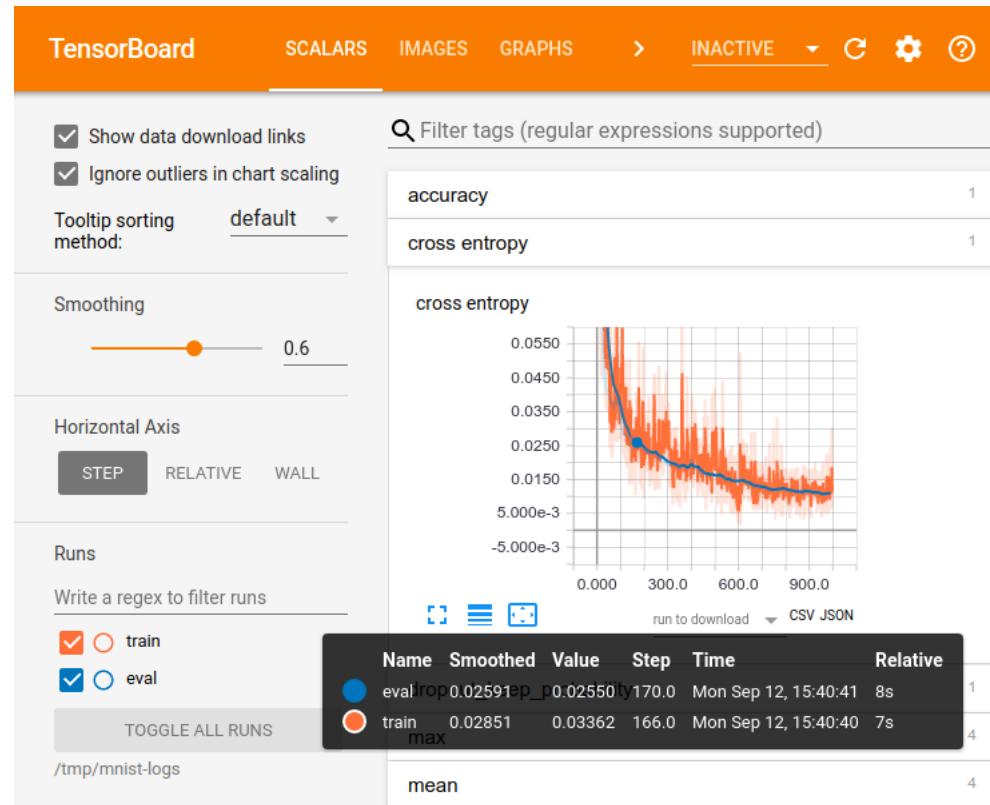
## 미분 및 Backpropagation API

**tf.train.Optimizer(learning\_rate).minimize(loss\*)**



# Tensorboard

## 딥러닝 학습 시 유용한 시각화 툴



# 요약

## Graph

그래프 구축 –  $y = ax + b$

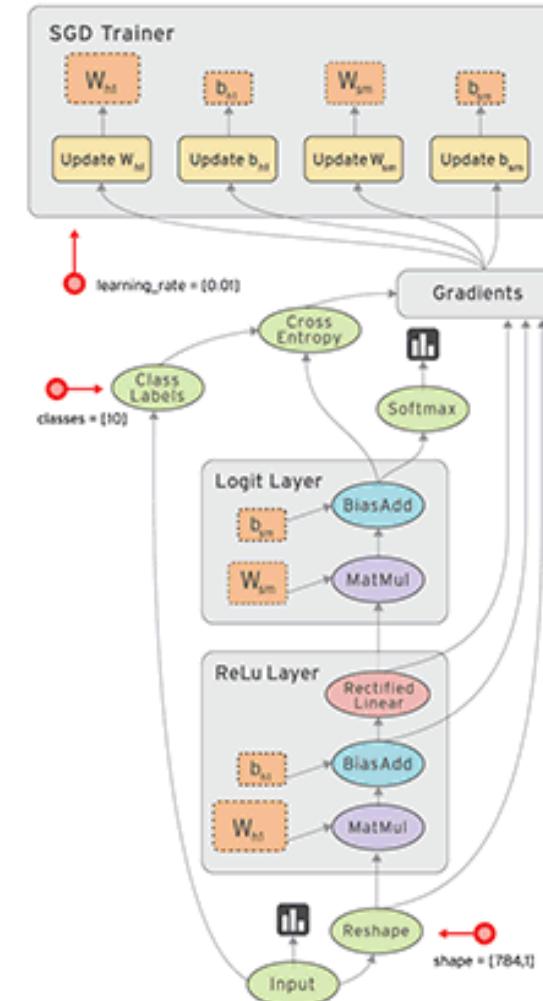
## Session

입력 – Feeding

출력 및 실행 – sess.run(Fetching)

## Trainer

Optimizer – 미분, 갱신



# Thank You

## Q & A

wltjr1007@korea.ac.kr  
<https://ku-milab.org>



(Summer School) 딥러닝 기반 Neuroimaging 데이터 분석

# 딥 뉴럴 네트워크 기반 Resting-State fMRI 분석 및 실습

\*본 실습은 [TensorFlow v1.9](#) 기준으로 진행됩니다.

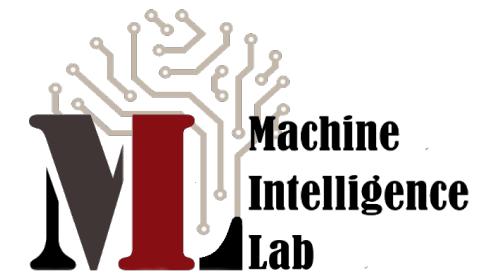


## 조교: 강은송

강사: 석홍일

eunsong1210@korea.ac.kr

<http://www.ku-milab.org>



고려대학교 뇌공학과  
기계지능연구실

2018년 8월 18일

# 개요

**목표** Matrix factorization 관점에서의 Deep neural network 기반 Resting-state fMRI 분석

## Data

ADNI2

Resting-state fMRI

<http://adni.loni.usc.edu/>



**61** subjects (MCI 31/ CN 30)  
(Mild Cognitive Impairment/ Cognitive Normal)

**130** time scans

**116** ROIs

Computation 문제로  
voxel을 AAL116 atlas 이용하여  
ROIs 단위로 변경

## Matrix Factorization

Independent  
Component Analysis

(ICA)

vs.

Deep Neural Networks

(DNN)

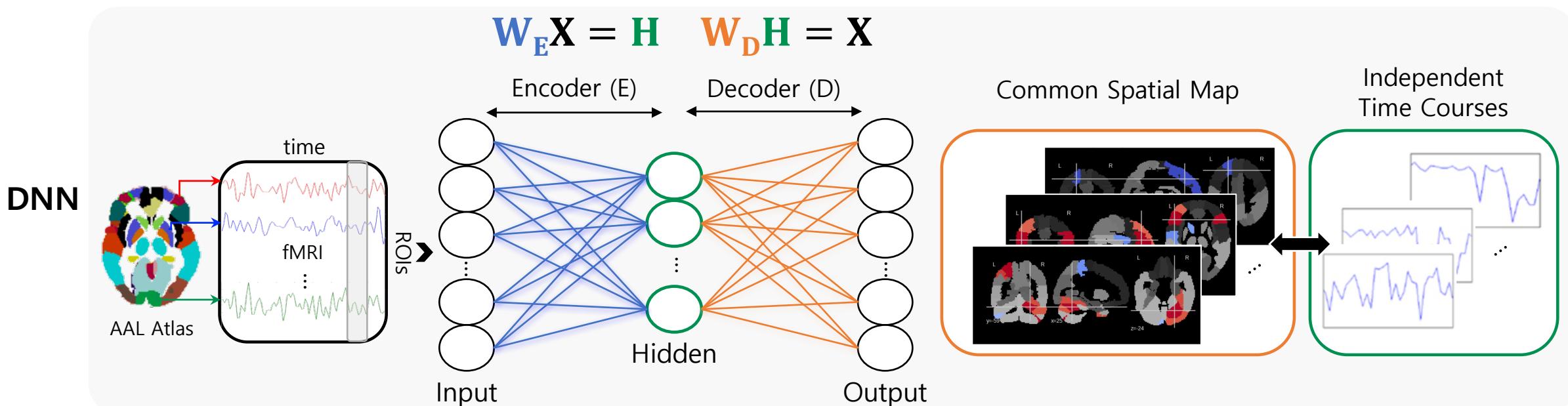
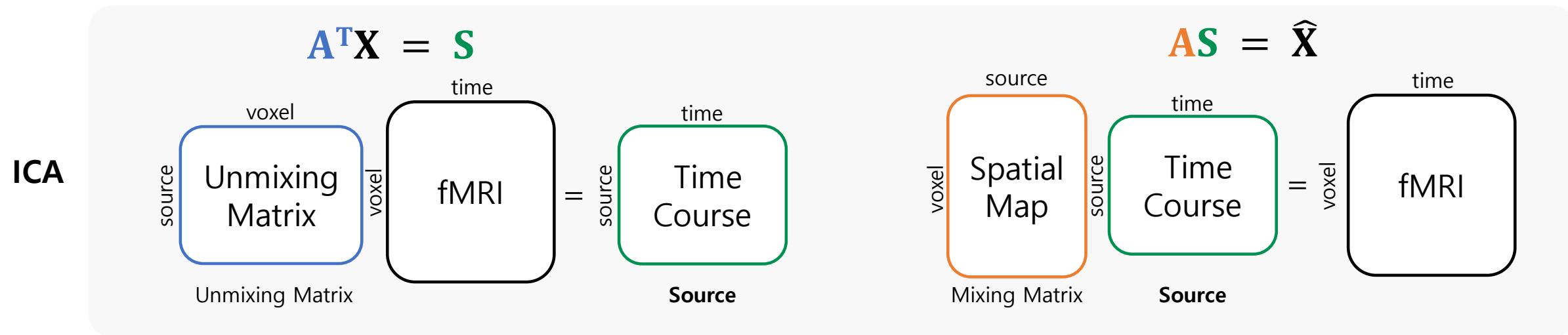
## Deep Neural Network

Auto-encoder

Stacked Auto-encoder

Sparse Auto-encoder

# Matrix Factorization



# Auto-encoder & Stacked Auto-encoder

Encoder:  $\mathbf{h} = f_E(\mathbf{W}_E \mathbf{x} + \mathbf{b}_E)$

Decoder:  $\hat{\mathbf{x}} = f_D(\mathbf{W}_D \mathbf{h} + \mathbf{b}_D) \approx \mathbf{x}$

Loss:  $J(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^N \| \mathbf{x} - \hat{\mathbf{x}} \|_2^2$

$\mathbf{x} \in \mathbb{R}^{D_I}$ ,  $\mathbf{h} \in \mathbb{R}^{D_H}$

$D_I$ : # of ROIs,

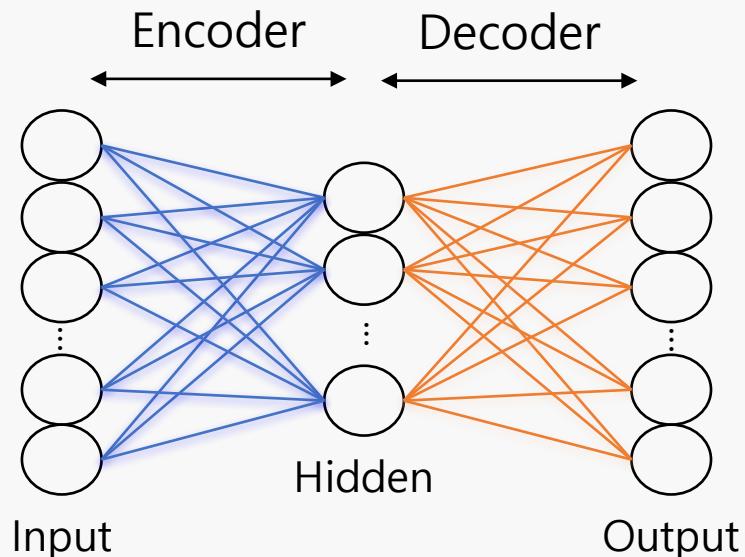
$D_H$ : # of hidden layer units

$\mathbf{W}_E \in \mathbb{R}^{D_I \times D_H}$ ,  $\mathbf{W}_D \in \mathbb{R}^{D_H \times D_I}$

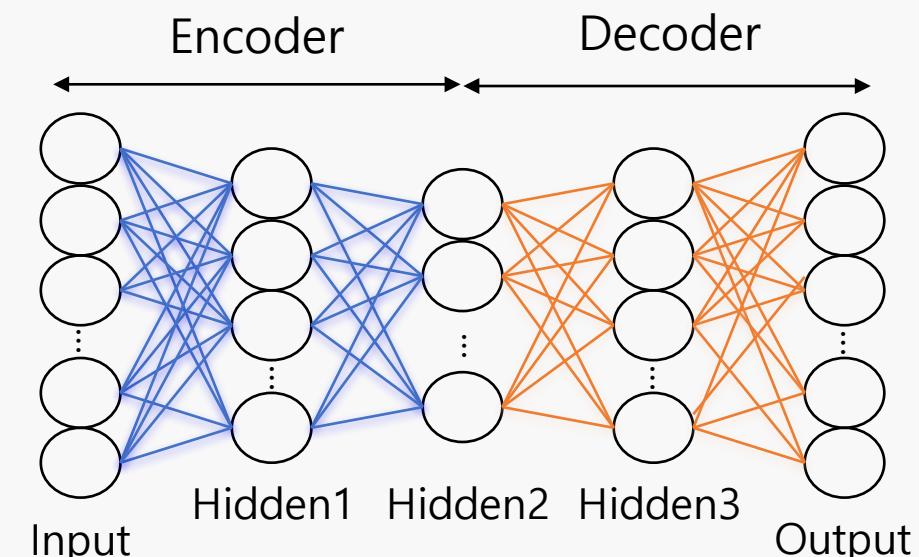
$N$ : # of data samples

$f_E$ : non-linear activation function

$f_D$ : linear activation function



Auto-encoder



Stacked auto-encoder

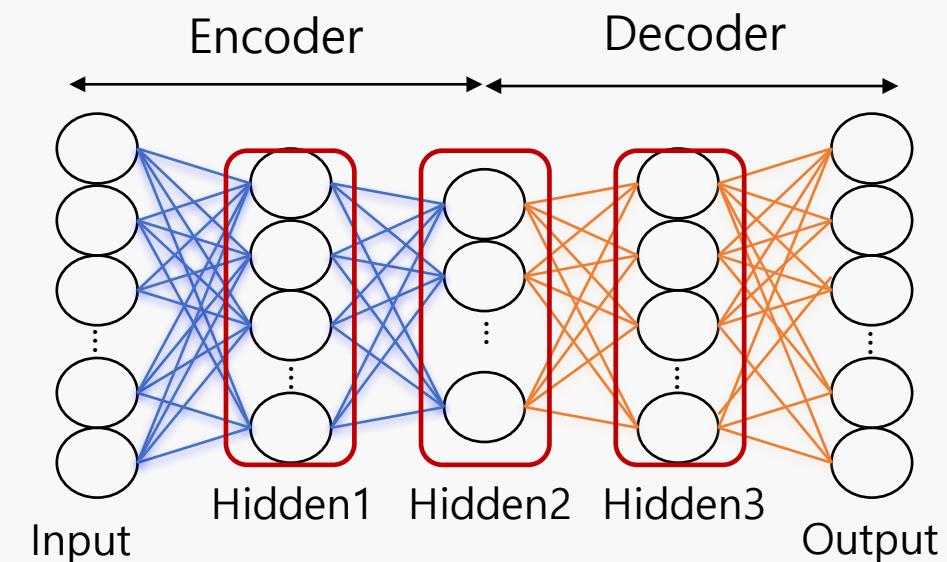
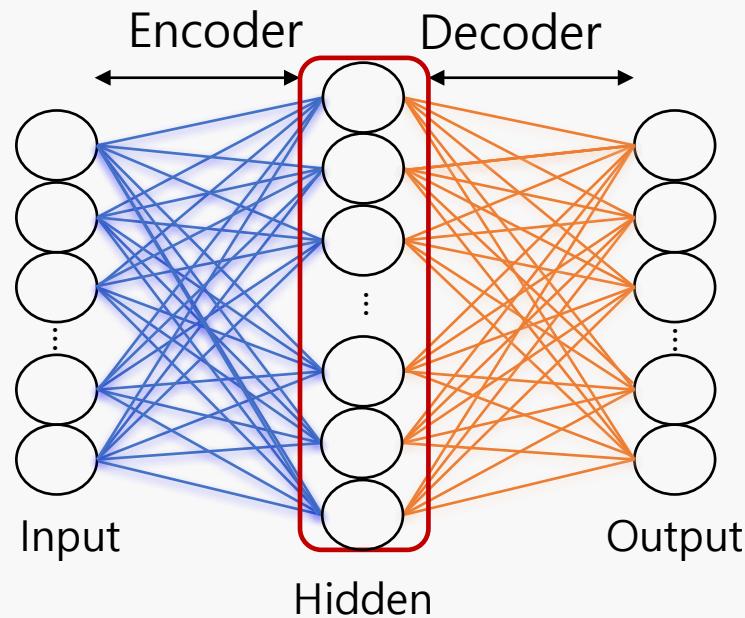
# Sparse Auto-encoder

Constraint:  $\hat{\rho}_j = \rho$

Loss:  $J(\mathbf{W}, \mathbf{b}) + \beta \sum_{j=1}^{D_H} KL(\rho || \hat{\rho}_j)$

Mean Squared Error (MSE) + Sparsity

$N$ : # of data samples  
 $D_H$ : # of hidden layer units  
 $\hat{\rho}_j = \frac{1}{N} \sum_{n=1}^N \mathbf{h}_j$



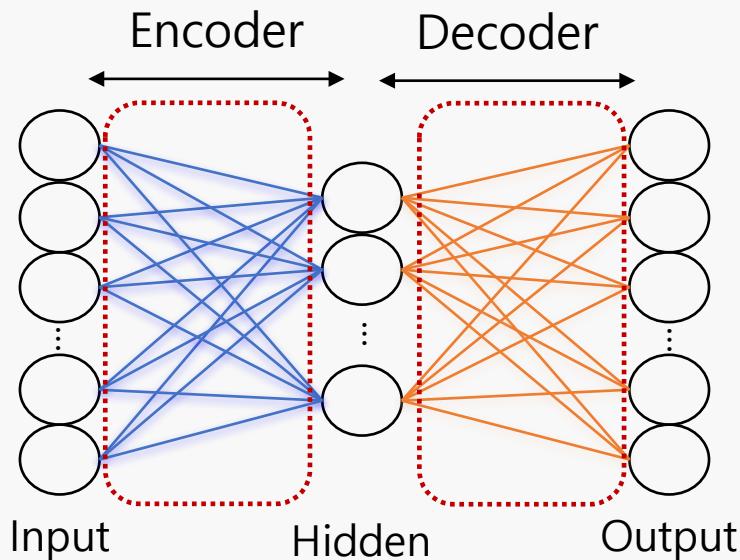
# L2 Regularization

$$\text{Loss: } J(\mathbf{W}, \mathbf{b}) + \lambda \|\mathbf{W}\|_2^2$$

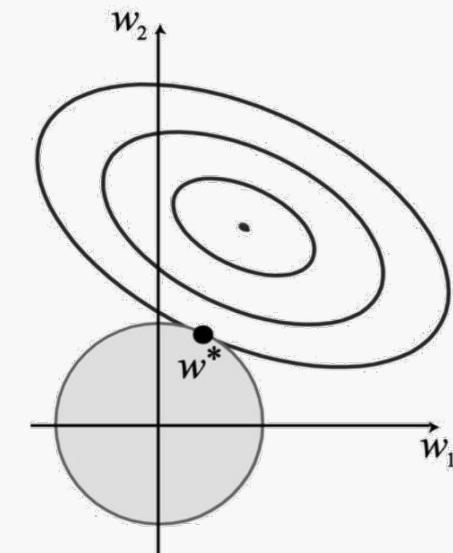
Mean Squared Error (MSE) + L2 regularization

$\lambda$ : L2 regularization parameter

$\mathbf{W}$ : Encoder or decoder weight



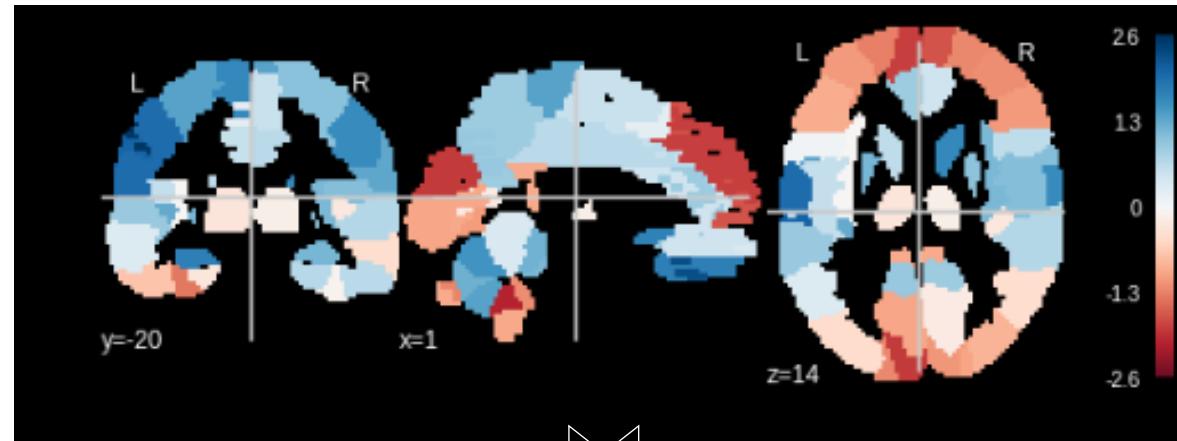
Auto-encoder



L2 regularization

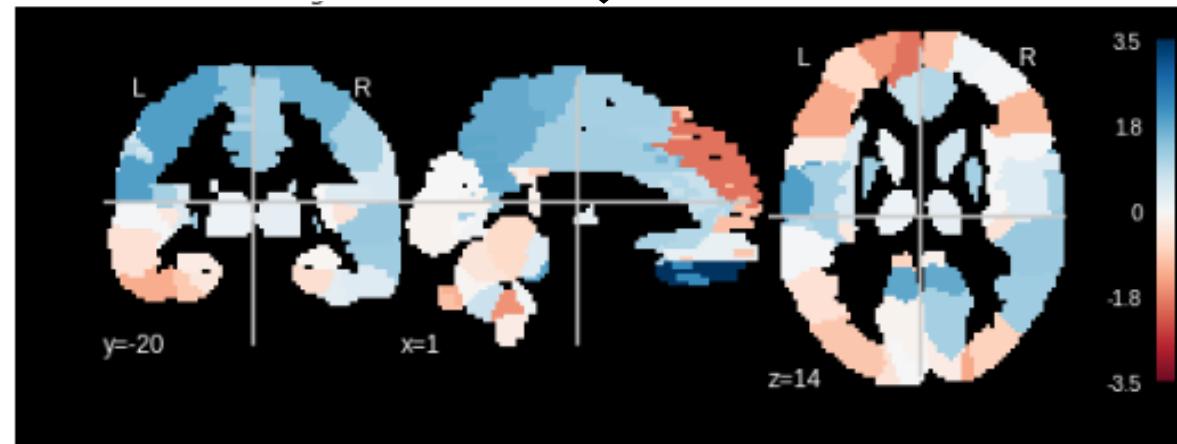
# Visualization : Auto-encoder Input & Output

Input  
fMRI  
(subject1)



Auto-Encoder

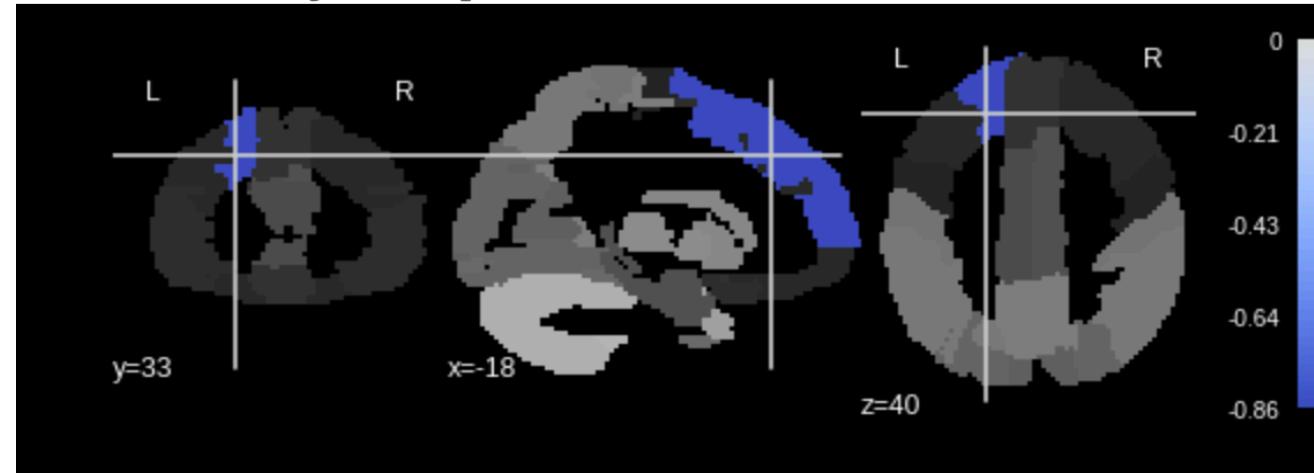
Output  
fMRI  
(subject1)



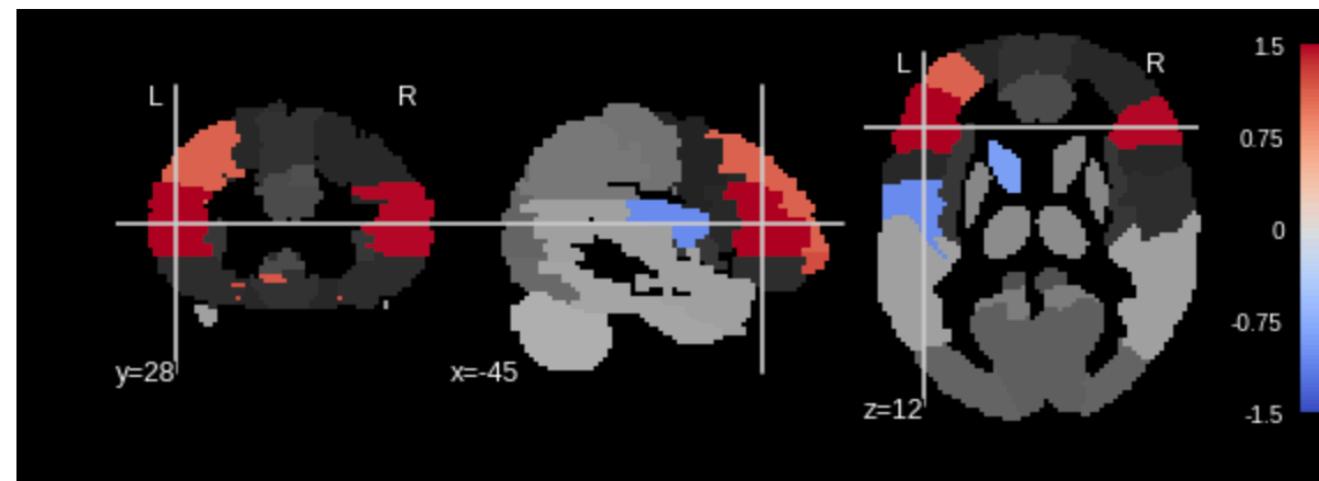
# Visualization : Auto-encoder Decoder Weight

- Thresholded image

**Weight Component  
16**



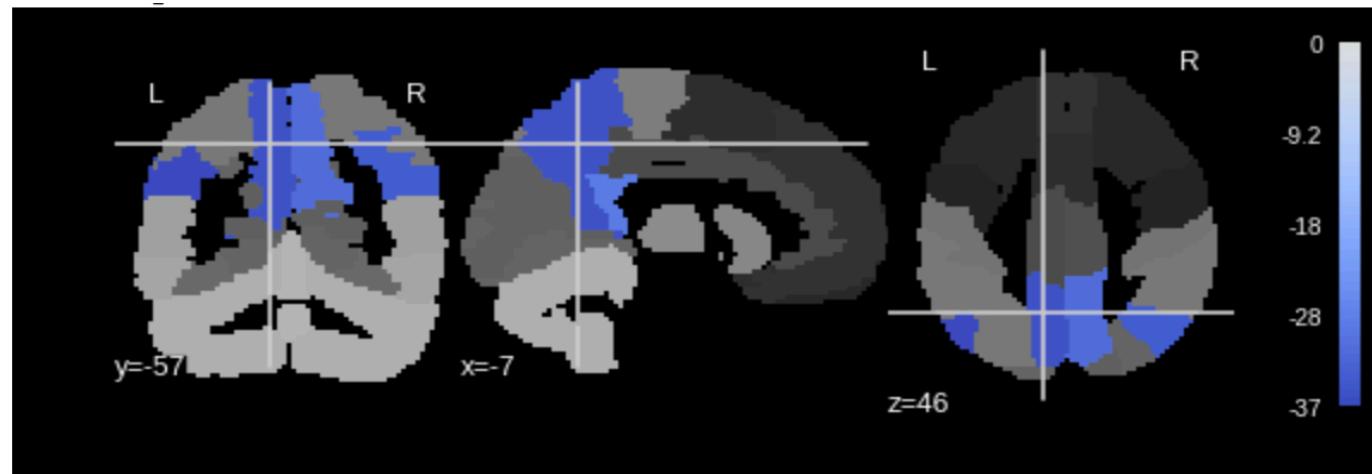
**Weight Component  
20**



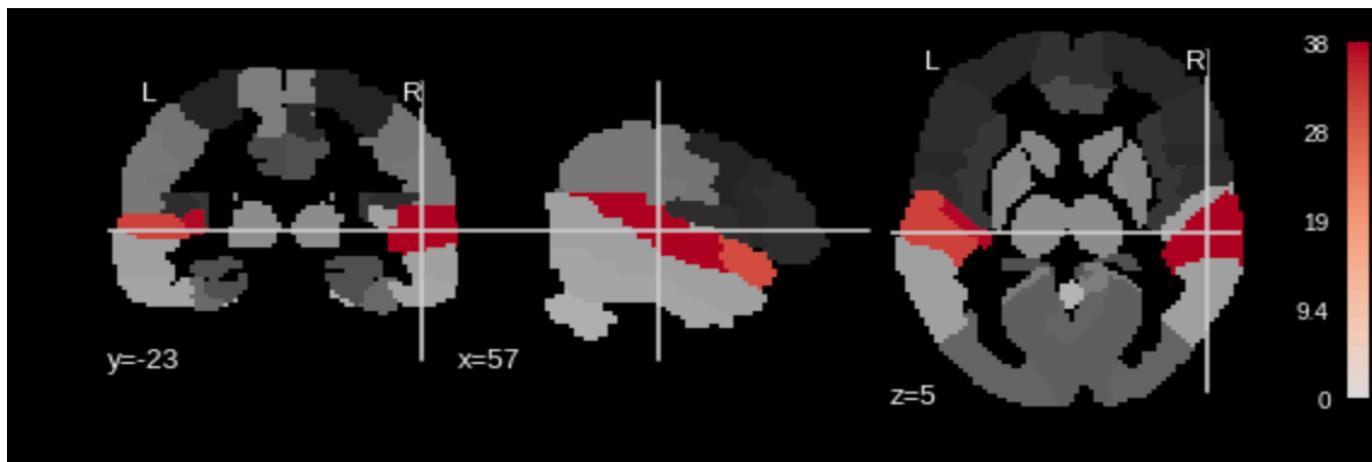
# Visualization : ICA Spatial Map

- Thresholded image

**ICA  
Component  
20**

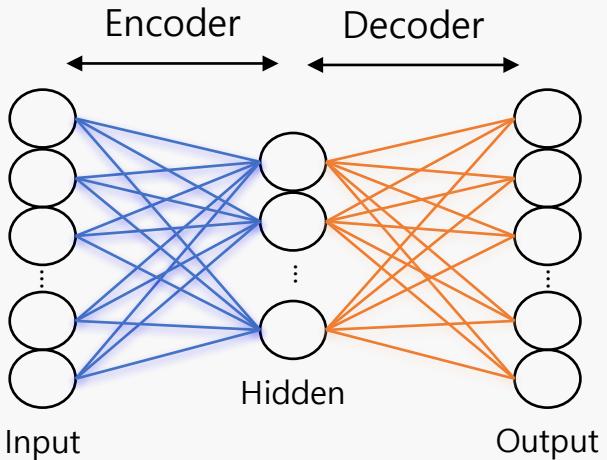


**ICA  
Component  
24**

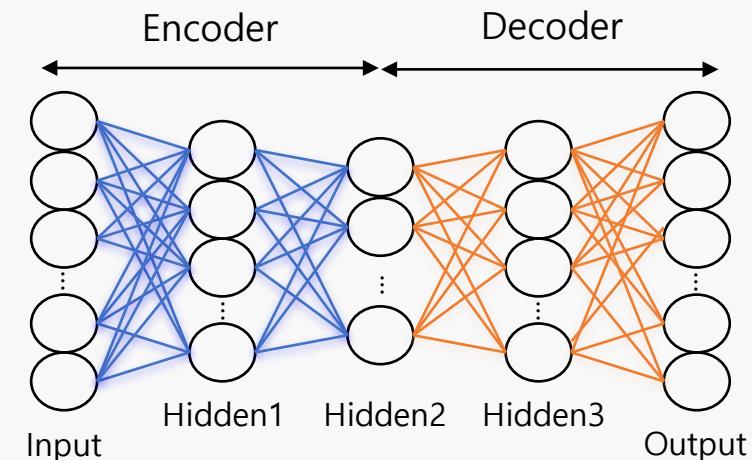


# 요약

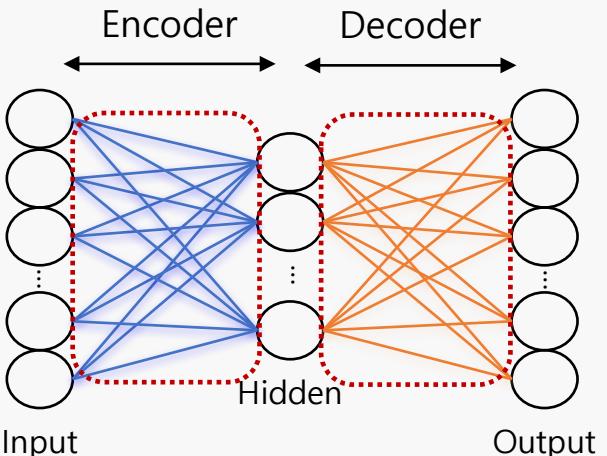
**Auto-encoder**



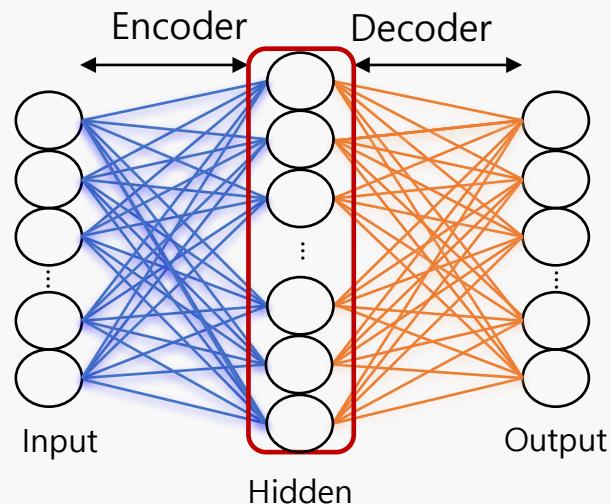
**Stacked auto-encoder**



**L2 regularization**



**Sparse auto-encoder**



# Thank You

## Q & A

eunsong1210@korea.ac.kr  
<https://ku-milab.org>



(Summer School) 딥러닝 기반 Neuroimaging 데이터 분석

# 컨볼루션 네트워크 기반 EEG 신호 분석 및 실습

\*본 실습은 [TensorFlow v1.9](#) 기준으로 진행됩니다.



조교: 고원준

강사: 석홍일

wjko@korea.ac.kr

<http://www.ku-milab.org>



고려대학교 뇌공학과  
기계지능연구실

2018년 8월 18일

# 개요

**목표** 컨볼루션 네트워크를 이용한 EEG 신호 분석 및 공간축 컨볼루션층 해석

## 데이터

BCI Competition IV



BCI Competition IV-IIa  
Motor Imagery EEG

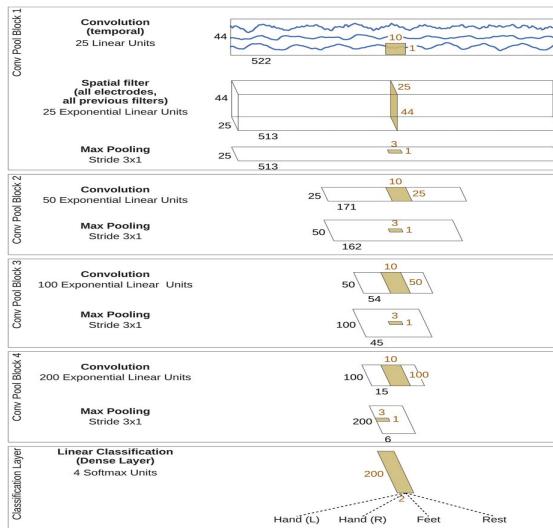
<http://www.bbci.de/competition/iv/>

9 subjects  
22 channels  
4 classes

실습을 위해 subject 1의 왼손,  
오른손 motor imagery EEG를 사용

## 컨볼루션 네트워크 Deep ConvNet

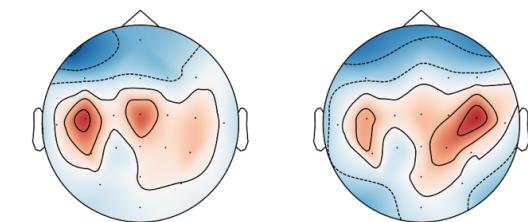
[Schirrmeister et al., HBM, 2017]



## 컨볼루션층 해석 Activation Pattern

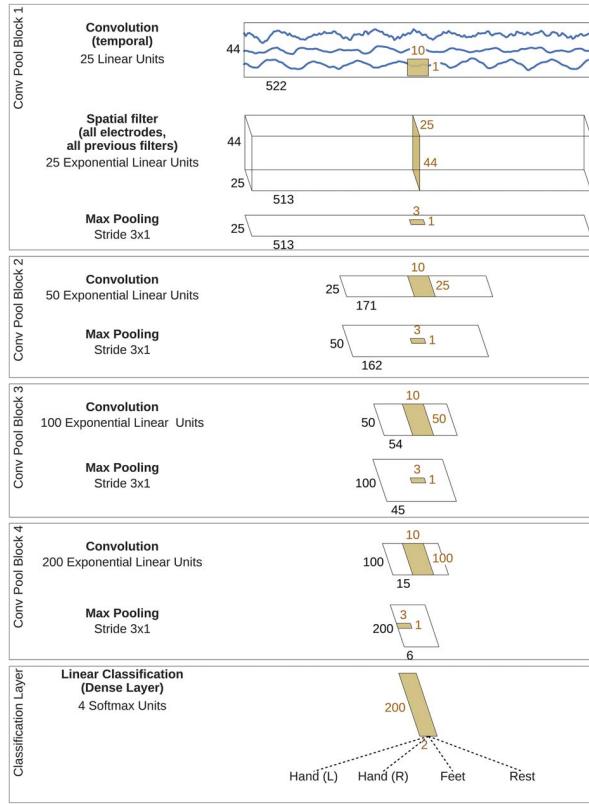
[Haufe et al., NeuroImage, 2014]

$$\mathbf{A} = \sum_{\mathbf{x}} \mathbf{W} \mathbf{\Sigma}_{\hat{\mathbf{s}}}^{-1}$$

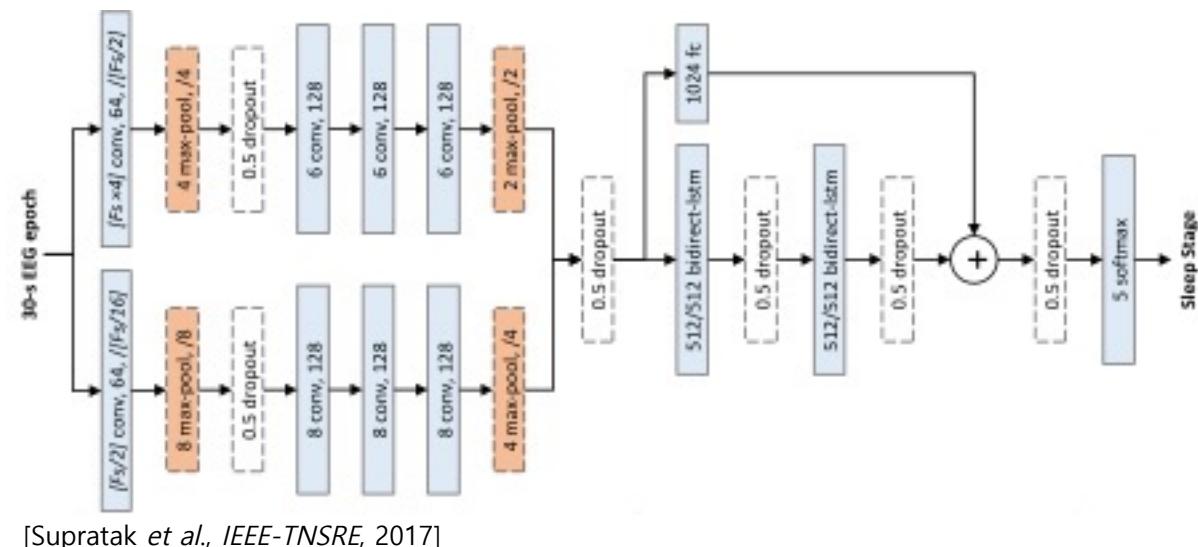


오른손, 왼손 motor imagery EEG에 활성화된 컨볼루션층의 지형도

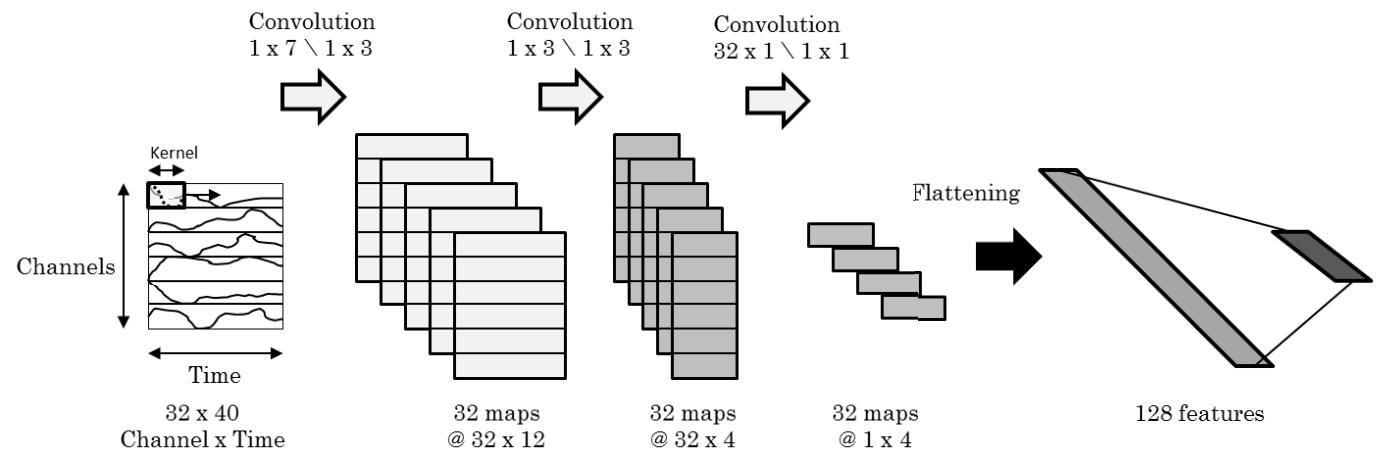
# 컨볼루션 네트워크를 이용한 EEG 분석 예시



[Schirrmeister et al., HBM, 2017]



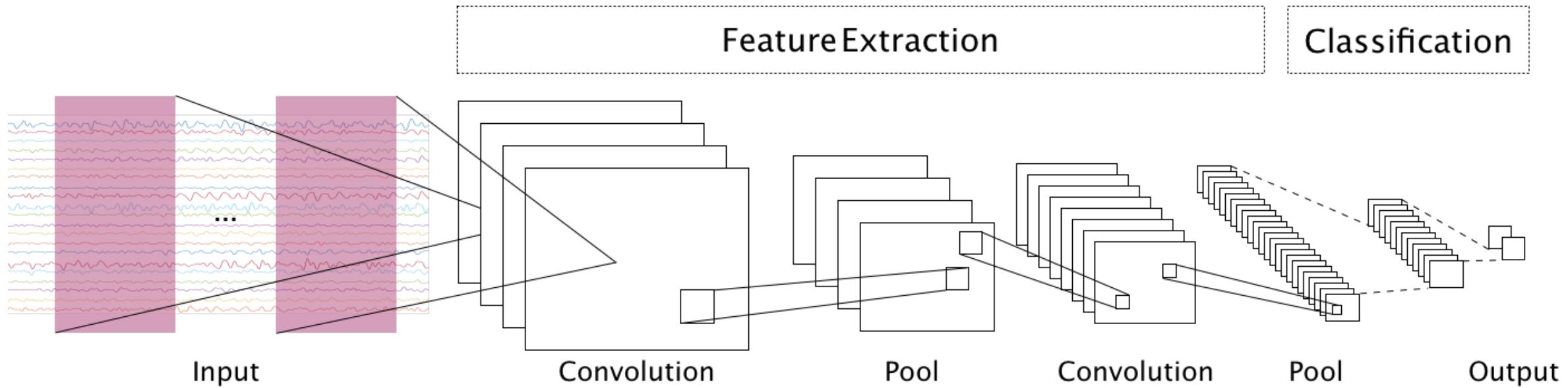
[Supratak et al., IEEE-TNSRE, 2017]



[Sakhavi et al., IEEE-TNNLS, 2018]

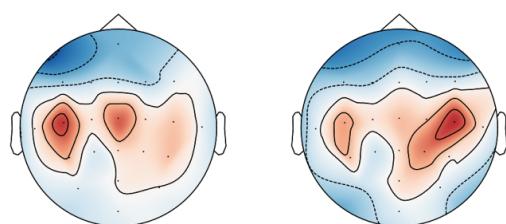
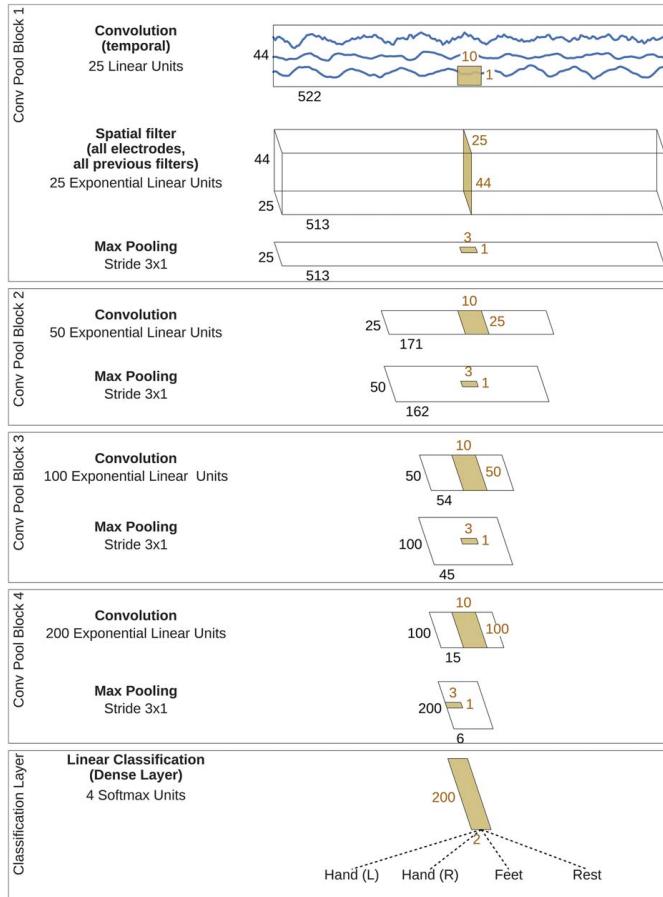
# Fully Connected Layer vs. Convolutional Neural Network

- Fully Connected Layer는 1차원의 입력 데이터를 다루는 데에 최적화되어 있음
  - 컨볼루션 네트워크(CNN)는 다차원의 정보를 다루기에 용이



- 컨볼루션 네트워크는 EEG 신호의 반복되는 패턴들을 parameter sharing을 통해 효율적으로 찾아낼 수 있음
- EEG 신호에 함축되어 있는 공간축, 시간축 정보를 추출하기가 용이함

# 개요



Activation pattern을 이용한 해석

시간축, 공간축 컨볼루션을 이용하여 정보 추출 후  
pooling을 이용하여 downsampling  
(해석을 위한 공간축 컨볼루션 필터 추출)

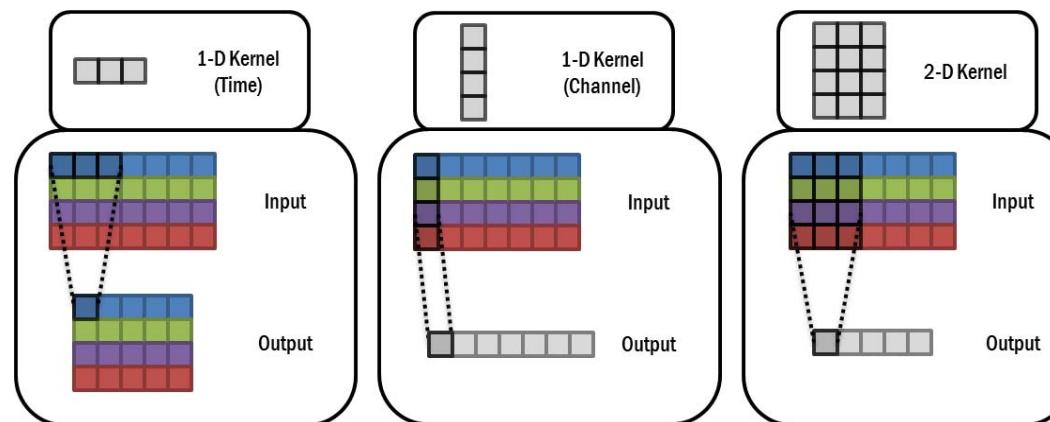
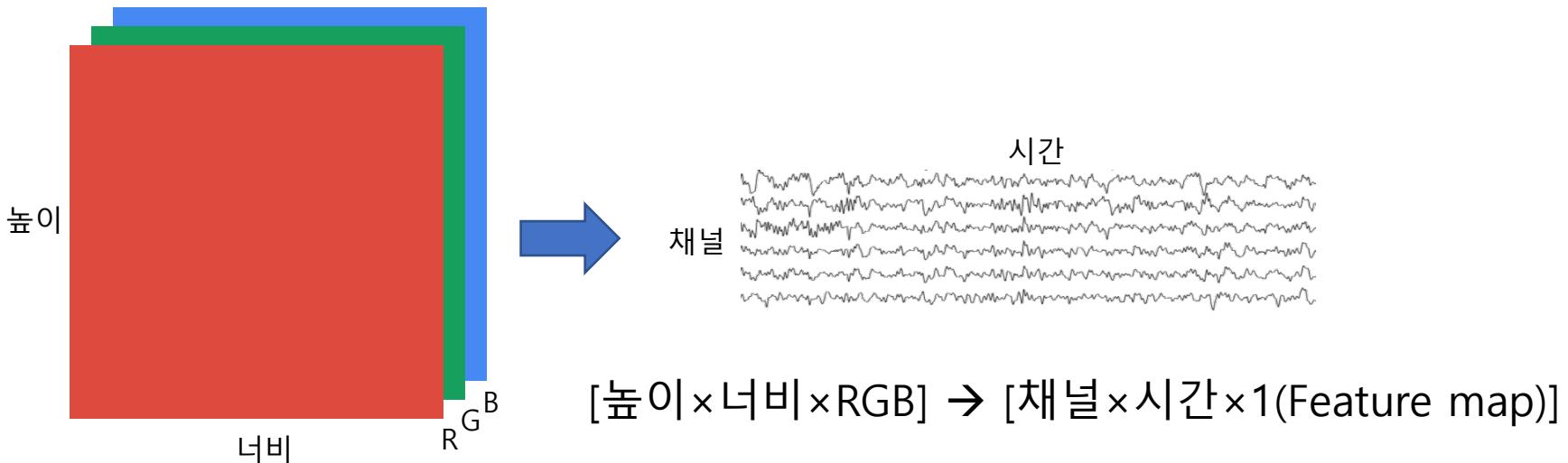
시간축 정보 추출 후 downsampling

시간축 정보 추출 후 downsampling

시간축 정보 추출 후 downsampling  
(여러 층에 걸친 컨볼루션으로 데이터의 hierarchical한 정보 추출)

Dense layer로 분류

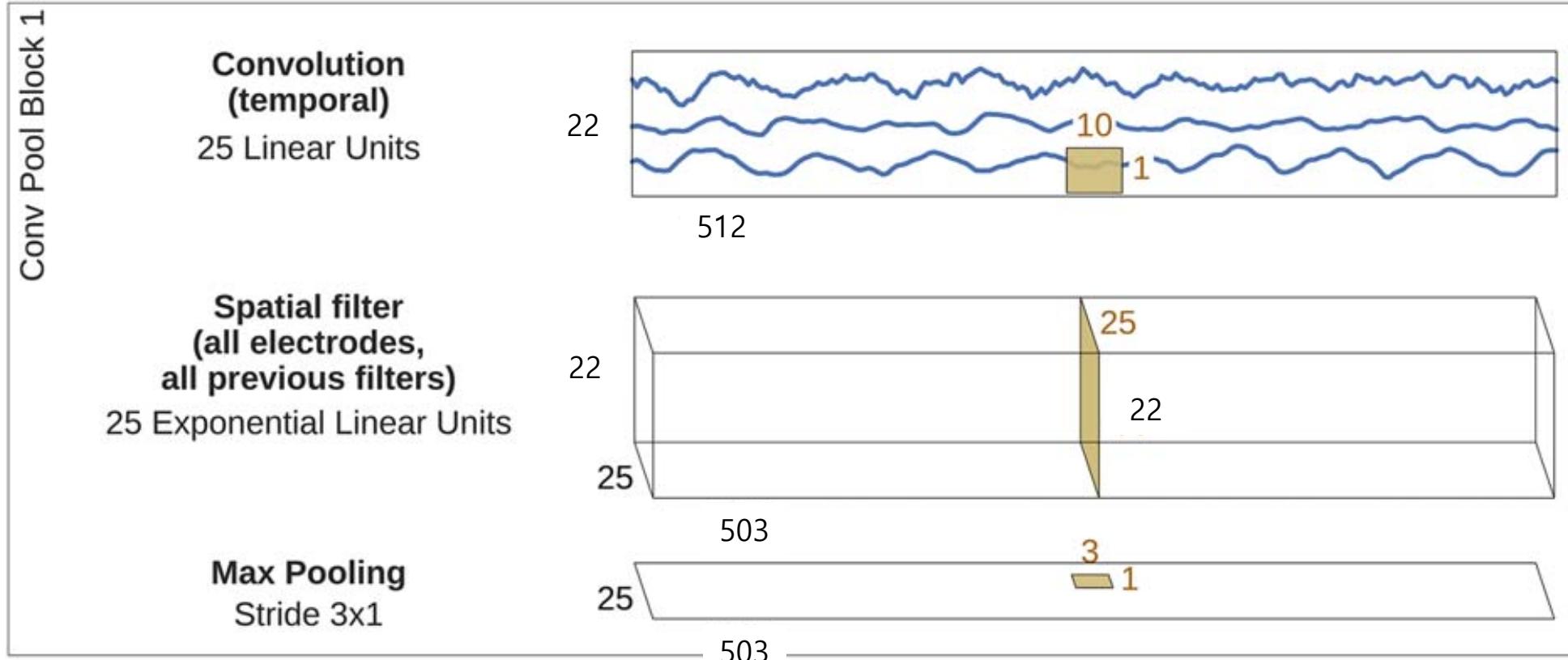
# 컨볼루션 네트워크를 위한 EEG 표현법



[Sakhavi et al., IEEE-TNNLS, 2018]

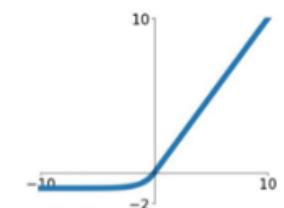
컨볼루션 커널 모양에 따라  
시간축(temporal),  
공간축(spatial),  
2차원 컨볼루션으로 나눔

# DeepConvNet [Schirrmeister et al., 2017]



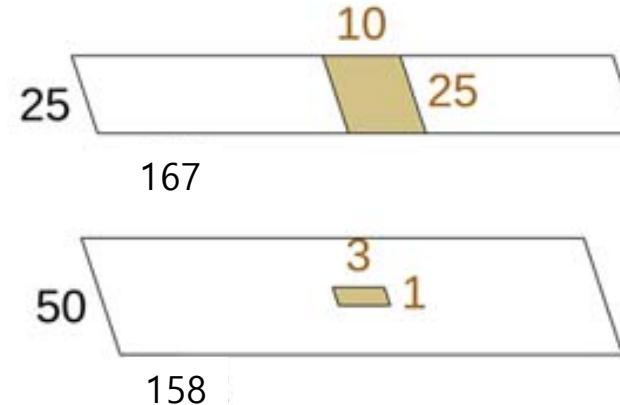
1. 시간축 정보 추출( $1 \times 10$  커널), Feature Map의 개수 증대( $1 \rightarrow 25$ )
2. 공간축 정보 추출( $22 \times 1$  커널), Batch normalization, ELU 활성화
3. Max Pooling을 통한 downsampling, Dropout

$$\text{ELU} = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



## Conv Pool Block 2

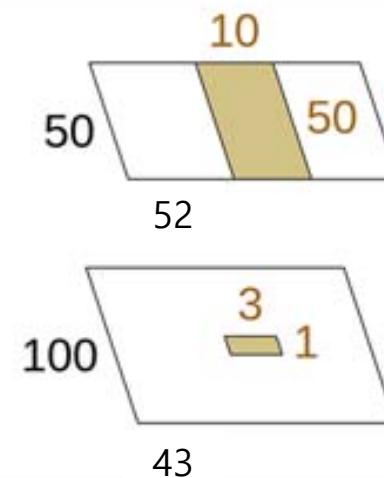
**Convolution**  
50 Exponential Linear Units



## Conv Pool Block 3

**Convolution**  
100 Exponential Linear Units

**Max Pooling**  
Stride 3x1

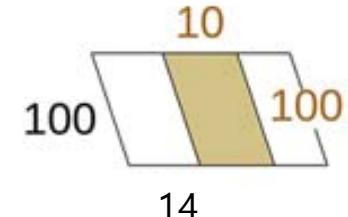


4. 시간축 정보 추출( $1 \times 10$  커널), Batch normalization, ELU 활성화, Feature Map의 개수 증대( $25 \rightarrow 50$ )
5. Max Pooling을 통한 downsampling, Dropout
  
6. 시간축 정보 추출( $1 \times 10$  커널), Batch normalization, ELU 활성화, Feature Map의 개수 증대( $50 \rightarrow 100$ )
7. Max Pooling을 통한 downsampling, Dropout

#### Conv Pool Block 4

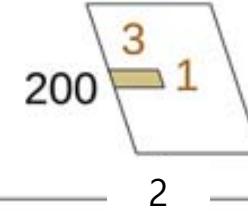
##### Convolution

200 Exponential Linear Units



##### Max Pooling

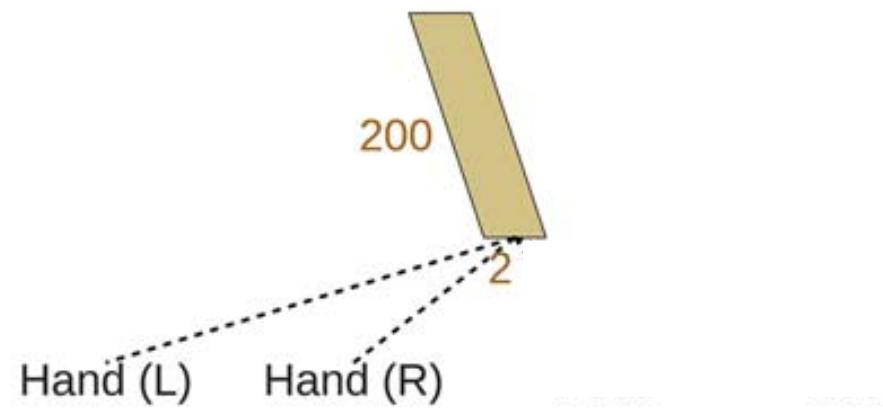
Stride 3x1



#### Classification Layer

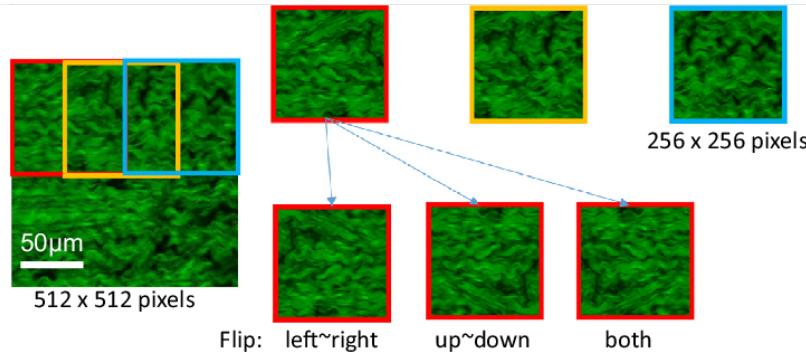
##### Linear Classification (Dense Layer)

Sigmoid Unit



8. 시간축 정보 추출( $1 \times 10$  커널), Batch normalization, ELU 활성화, Feature Map의 개수 증대( $100 \rightarrow 200$ )
9. Max Pooling을 통한 downsampling, Dropout
10. Dense layer(sigmoid 활성화)로 Classifier 구성

# 분류 실험



슬라이딩 윈도우(sliding window)를 통한 데이터 증대(data augmentation)

$$CE = - \sum_x p(x) \log q(x)$$

Sigmoid 함수를 거쳐 나온 예측값과 실제 레이블 값과의 cross-entropy 값을  
목적함수로 사용

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Adam optimizer를 이용하여 loss의 기울기값을 통해 최적화된 parameter를 찾음

- 학습 시, 매 epoch마다 training 데이터를 무작위로 뒤섞어서, 알고리즘이 데이터의 순서를 외우지 않도록 조정
- 매 epoch마다 loss를 출력하여, 학습이 잘 진행되는지 확인

# 네트워크 해석

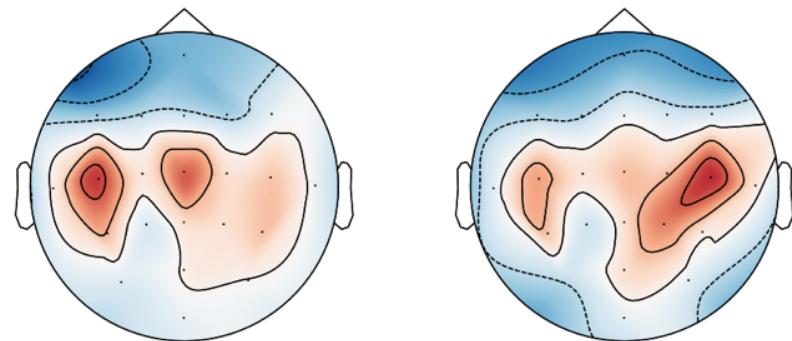
추출한 공간축 컨볼루션 필터는 신경생리학적으로 무의미함, 즉 해석의 의미가 없음

$$\mathbf{A} = \sum_{\mathbf{x}} \mathbf{W} \mathbf{\Sigma}_{\hat{\mathbf{s}}}^{-1}.$$

[Haufe et al., NeuroImage, 2014]

Haufe et al.은 필터에 입력의 공분산, 추측 레이블의 공분산의 역을 곱해서 activation pattern을 계산하는 방법을 제안함

Blind source의 source들이 활성화 되는 정도를 계산



계산한 activation pattern 값을 이용하여 뇌  
지형도(topography)를 그려 오른손, 왼손 동작 상상  
뇌신호에 따른 활성화 정도를 관찰할 수 있음

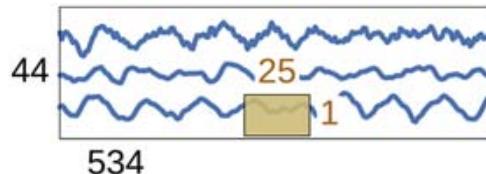
# Thank You

## Q & A

wjko@korea.ac.kr  
<https://ku-milab.org>

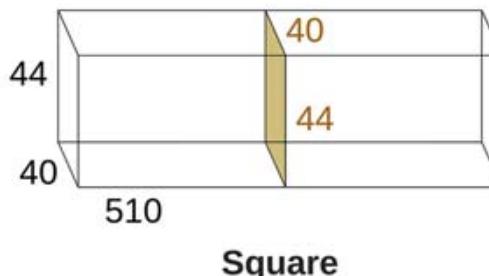
# (Backup) ShallowConvNet [Schirrmeister *et al.*]

Convolution  
(temporal)  
40 Units



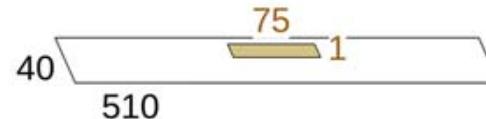
```
w1, b1 = get_params(name="conv1", shape=(1, 25, 1, 40), n_filter=40)
conv1 = conv(input=input, w=w1, b=b1)
```

Spatial filter  
(all electrodes,  
all previous filters)  
40 Units



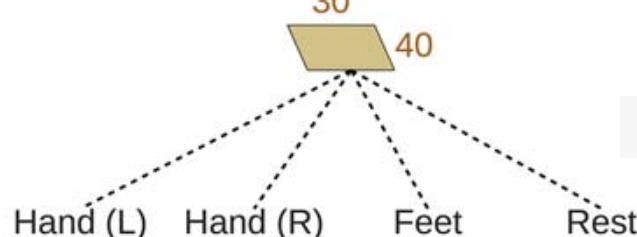
```
w2, b2 = get_params(name="conv1", shape=(22, 1, 40, 40), n_filter=40)
conv2 = conv(conv1, w=w2, b=b2)
conv2 = tf.square(conv2)
```

Mean Pooling  
Stride 15x1



```
conv2 = tf.nn.avg_pool(value=conv2, ksize=(1, 1, 75, 1),
                      strides=(1, 15), padding= "VALID")
conv2 = tf.log(conv2)
```

Linear Classification  
(Dense Layer+Softmax)  
4 Units



```
output = fclayer(input=conv2, name="output", num_output=4)
```