Deep Learning Seminar

3. Backpropagation

HA SEUNG HYUN

Contents

1. Optimization

2. Backpropagation

3. Neural Network (FCN) Fully Connected Network

중위적인 표현법

NN가 FCN, CNN, ANN, RNN을 범용적으로 지칭하는 경우가 많다.

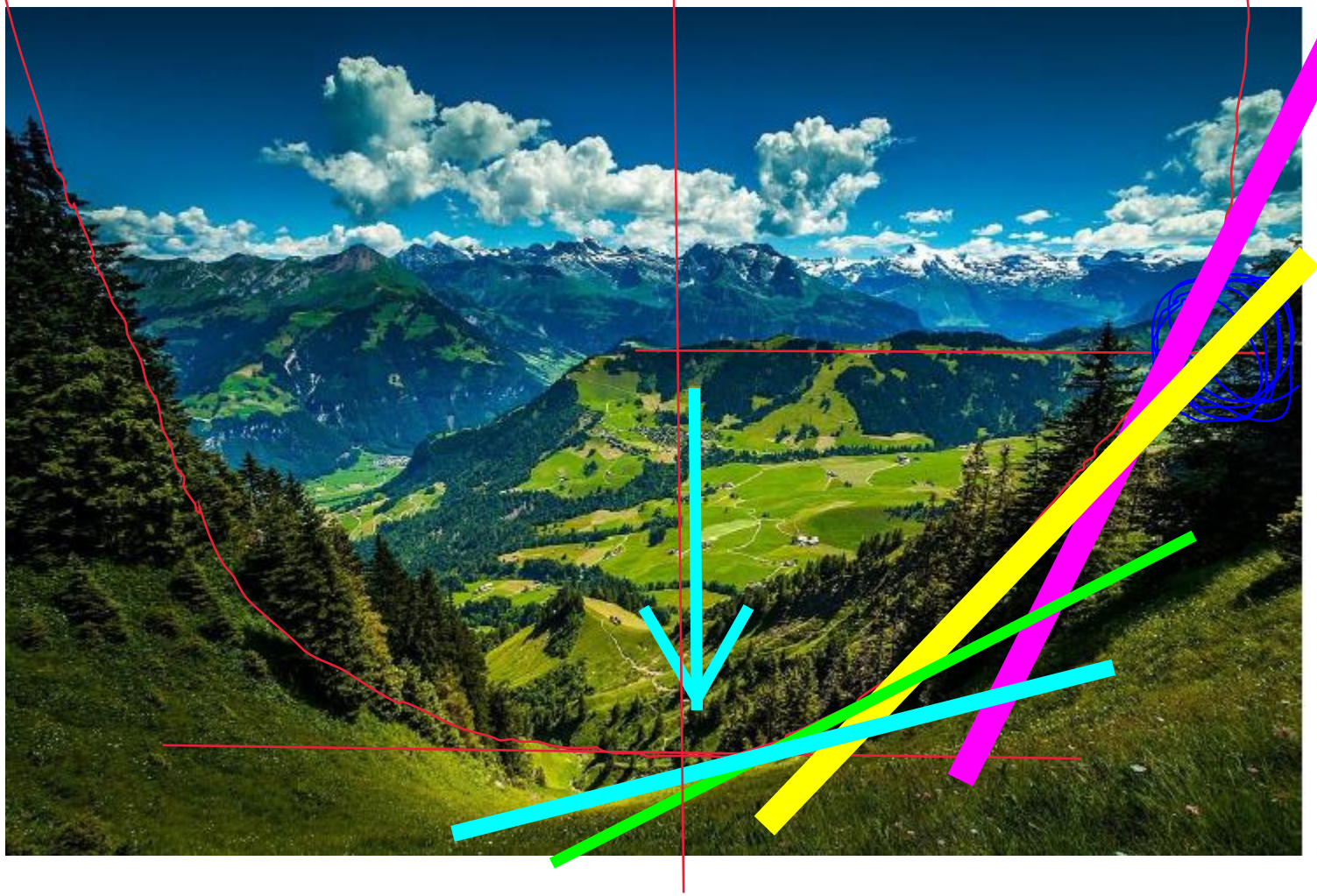
Reference: lecture note (Fei-Fei Li)
lecture note (Andrew Ng)
모두를 위한 머신러닝 (Sung kim)

1. Optimization

Optimization

산을 하산하는 방법 Gradient Descent
Back Propagation을 통해 내려오는 하산 방법

고도를 내리면 기울기 변함.



고도 → loss function

고도 = loss

학습진행
(loss가 0일때까지)
고도 ↓

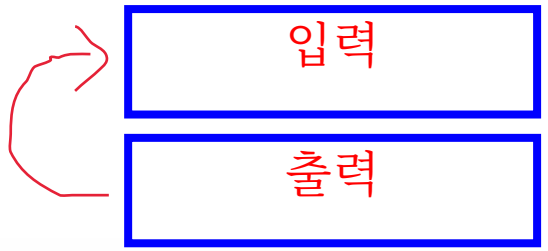
learning rate
→ 고도가 0으로 가까워지면 속도를 줄여야 함. 보폭을 줄여가며

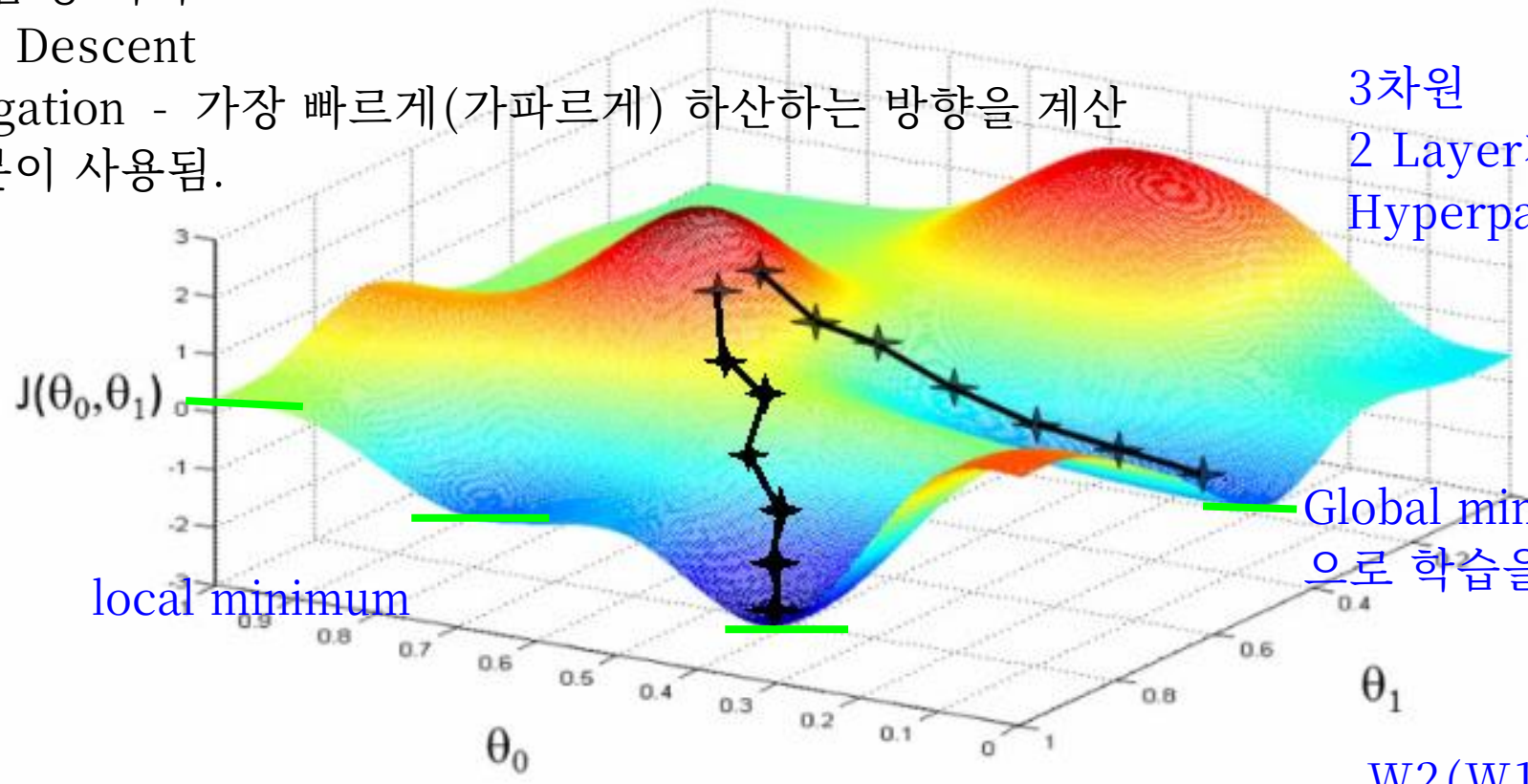
Optimization

"Back Propagation 을 이용해서 하산하는 방법이 Gradient Descent"

Optimization

1. loss function : 현재 내가 위치해 있는 고도
2. Optimization : 하산하는 방법(loss 값이 필요)
3. 하산하는 방법 중 하나
   - Gradient Descent
4. Back Propagation - 가장 빠르게(가파르게) 하산하는 방향을 계산
   미분, 편미분이 사용됨.





3차원
2 Layer가 존재(layer 하나가 Hyperparameters)

local minimum

Global minimum으로 내려오는 것으로 학습을 해야 함

H(x) = W1x + b1

W2(W1x + b1) + b2

Optimization

1) Random Search 가장 최초로 사용되었던 하산법 중 하나 (평지를 찾아내는 것). loss가 0인 부분

무작위로 W 값을 여러 번 넣은 뒤, 그 중에서 최고의 값을 설정 (Deep Learning 적 표현)

-> 정상에서 눈 가리고 여러 번 하산 해본 뒤, 그 중 최고의 길을 선택 (은유)

무작위 식이면 엉뚱한 곳에서 멈출 수도 있다. 이를 계속 반복했을 때 그나마 가장 잘 내려온 지점을 선택

Optimization

2) Random Local Search

무작위 방향으로 W 값 바꿔서 Loss가 감소하는지 확인 한 후, 감소하면 W값을 업데이트

-> 눈을 가리고 무작위 방향으로 정해서 발을 살짝 뻗어서 더듬어 보고 그게 내리막 길이면 한 발짝 내딛음

Random Local Serach는 일단 내려오긴 하지만 기울기가 완만하게 내려올 수도 있고
모든 방향을 한번 씩 찔러봐서 내려가는 방향을 선택하기 때문에 시행착오가 많다.

Optimization

3) Gradient Descent

trial error가 없다.
가장 가파르게 (빠르게) 하산하는 방법

가장 가파르게 Loss를 감소하는 W 방향을 수학적으로 계산 한 뒤, 해당 방향으로 이동

-> 가장 가파르게 내려갈 수 있는 방향을 계산해서 해당 방향으로 한 발짝 이동

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

학습 진행 속도와 연관이 깊다

y = Wx + b W값에 따라서 (방향)과 (속도)가 결정된다.

기울기 2/3 - > 천천히 내려온다. 보폭이 작다.

기울기3 - > 빨리 내려온다. 많이 내려온다. 보폭이 크다

미분.png, 미분2.png 참고

Optimization

섭씨,화씨에서 실습해본 공식

3) Gradient Descent

$$L(y,p) = -\sum_k y_k log p_k \qquad p_t = \frac{e^{o_t}}{\sum_k e^{o_k}}$$

$$\frac{\partial L}{\partial o_i} = -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} = -\sum_k y_k \frac{1}{p_k}\frac{\partial p_k}{\partial o_i}$$

$$= -y_i(1-p_i) - \sum_{k\neq i} y_k \frac{1}{p_k}(-p_k p_i)$$

$$= -y_i(1-p_i) + \sum_{k\neq i} y_k (p_i)$$

$$= -y_i + y_i p_i + \sum_{k\neq i} y_k (p_i)$$

$$= p_i \left(\sum_k y_k\right) - y_i = p_i - y_i$$

$$\frac{dO}{dW} = x \qquad \because O = Wx + b$$

$$\nabla_W L = \frac{dL}{dW} = \frac{dL}{dO} \times \frac{dO}{dW} = (p-y) \times (x)$$

Optimization

- Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

$$w_i^{k+1} = w_i^k - \eta \frac{dL}{dw_i}$$

Learning Rate > 0
(Hyperparameter)

loss가 크면 업데이트가 크고
loss가 작다면 업데이트가 작다
(미분.png참고)

코딩적으로 표현
원래 weight 10.

1. loss 3
2. loss 7

7 = 10 - 3
3 = 10 - 7
(일단 ∏는 제외)

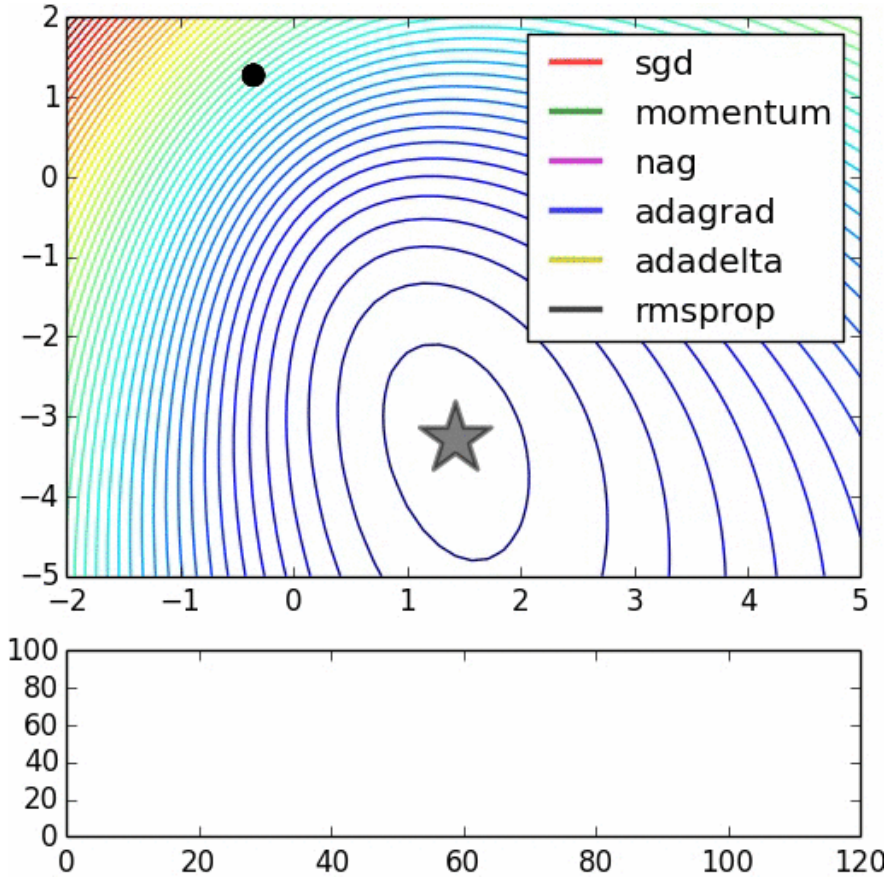
10 → 7
10 → 3

Optimization

(속도)adagrad → 속도 빠르다. 평지 근처 방황

(정확도)sgd → 속도 느리다. 정확하게 찾아감

- Optimizers Comparison



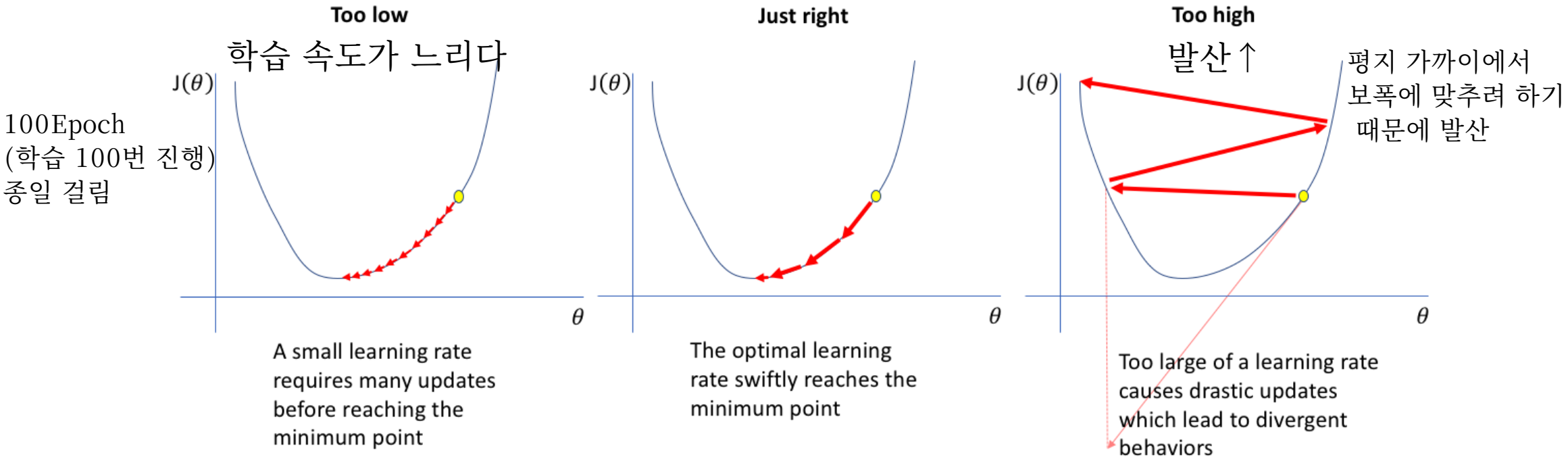
유동적으로 사용하지만
초반엔 adagrad
후반엔 sgd

Optimization

- **Learning rate** 하산시 보폭을 어떻게 가져야 할지를 지정

  Epoch(학습 횟수)을 반복해 갈수록 보폭을 줄여나가야 한다.

  하강보폭 Learning rate Decay



Optimization

- Learning rate tuning

SGD Optimizer default lr:	1e-2 (=0.1)
Adam Optimizer default lr:	1e-4 (=0.001)
Learning rate decay:	½ decay / n epoch

- Optimizer selection

1~10 epoch:	Adam Optimizer	(Fast, Rough)
10~ epoch:	SGD Optimizer	(Slow, Accurate)

Data Transformation
Data Augmentation

Epoch 하이퍼파라미터

Train 5만 validation 1만

문제집 풀고 모의고사 - > 1Epoch

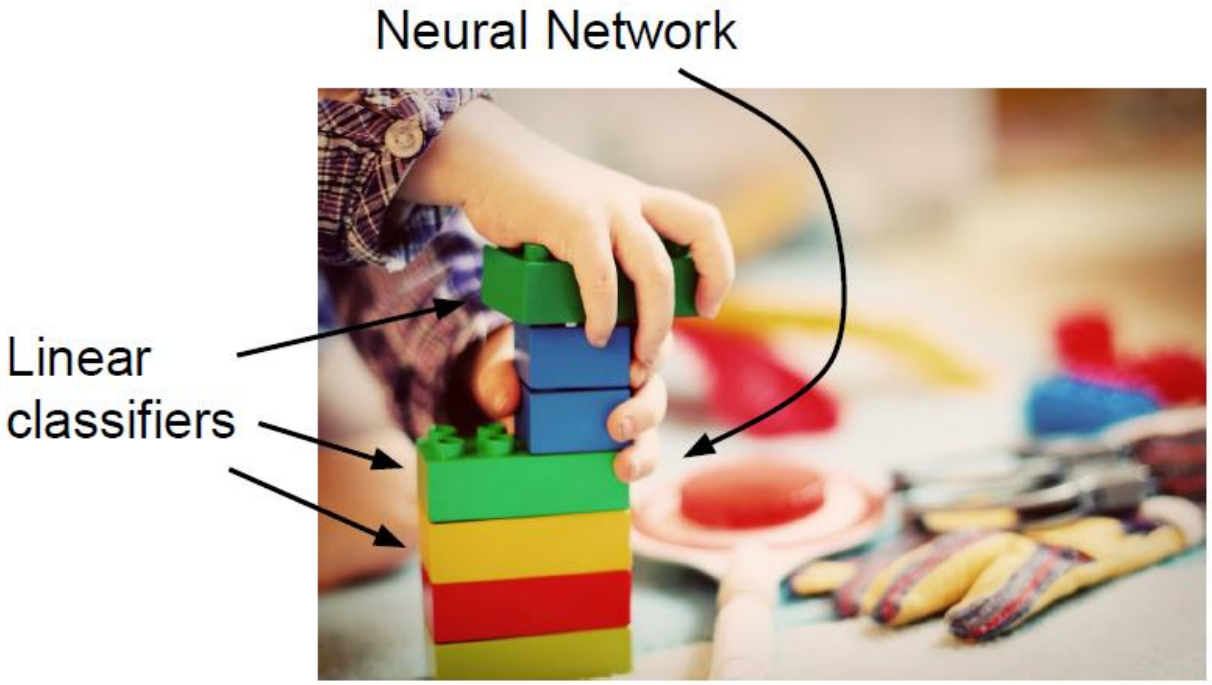
문제집 5만개 풀고 모의고사 1만개 풀고
- > 2 Epoch

※ 데이터 개수가 작을 땐 cross validation

2. Backpropagation

Backpropagation

- Neural Network



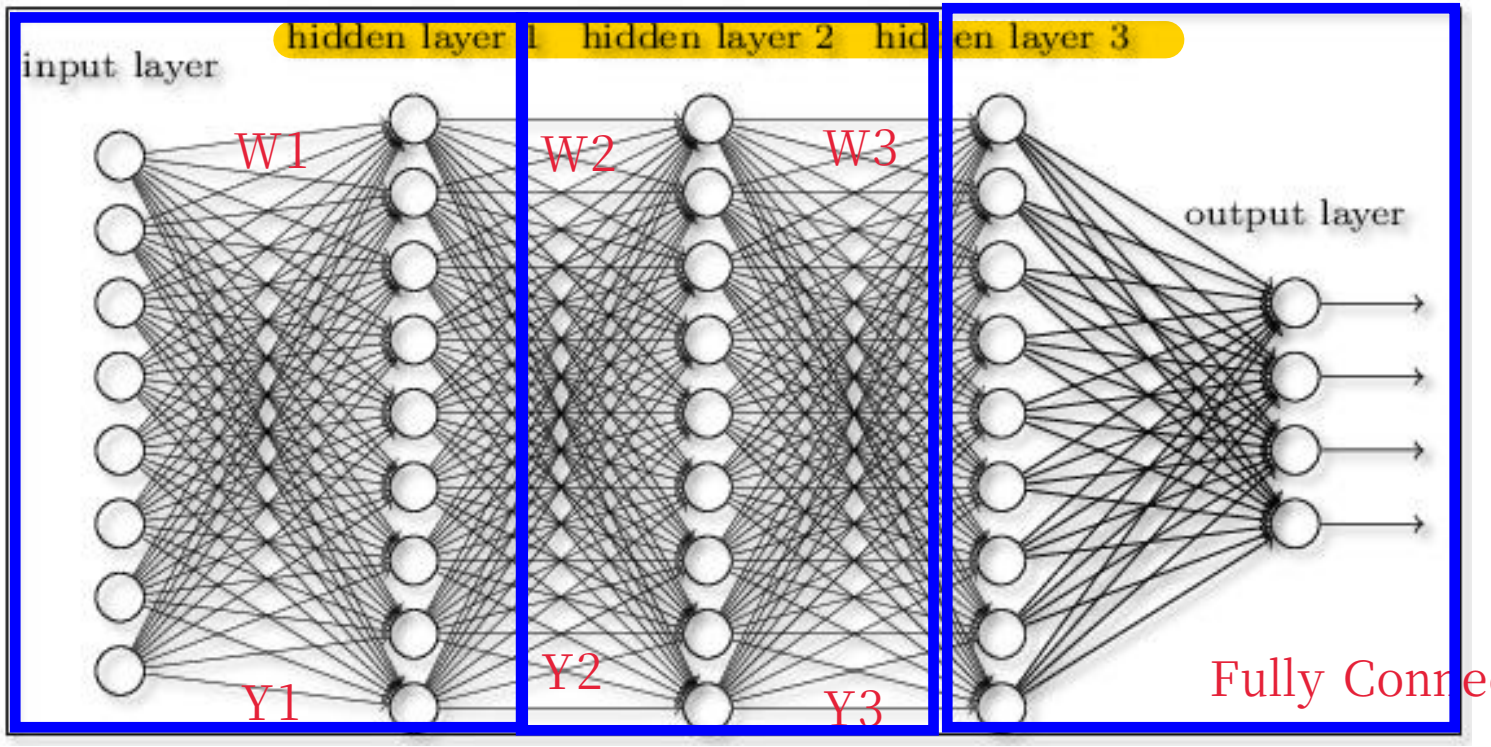
Backpropagation

- Neural Network

FCN - 모든 NN의 가장 기본

CNN, ANN, DNN, RNN
을 포괄해서 NN이라고도 함

히든 레이터 (개수는 하이퍼파라미터)
Feature Map
중간 결과물



Fully Connected
Layer
- > Network

Fully Connected Network(FCN)

Backpropagation

- Backpropagation

  - How to update weight ?

  - W 값이 Loss에 얼마나 영향을 주었는지를 수치화 한 뒤, Loss를 줄이는 방향으로 W값을 업데이트
    (Loss에 영향을 많이 주었으면 크게 W가 크게 변화)

  - Neural Network 의 학습을 위한 핵심적인 개념

3. Neural Network

3-1) Neural Network (Fully-connection network)

3-2) Convolutional Neural Network

인공신경망(Artifical N.N) 생물학적 신경망 영감

Neuron

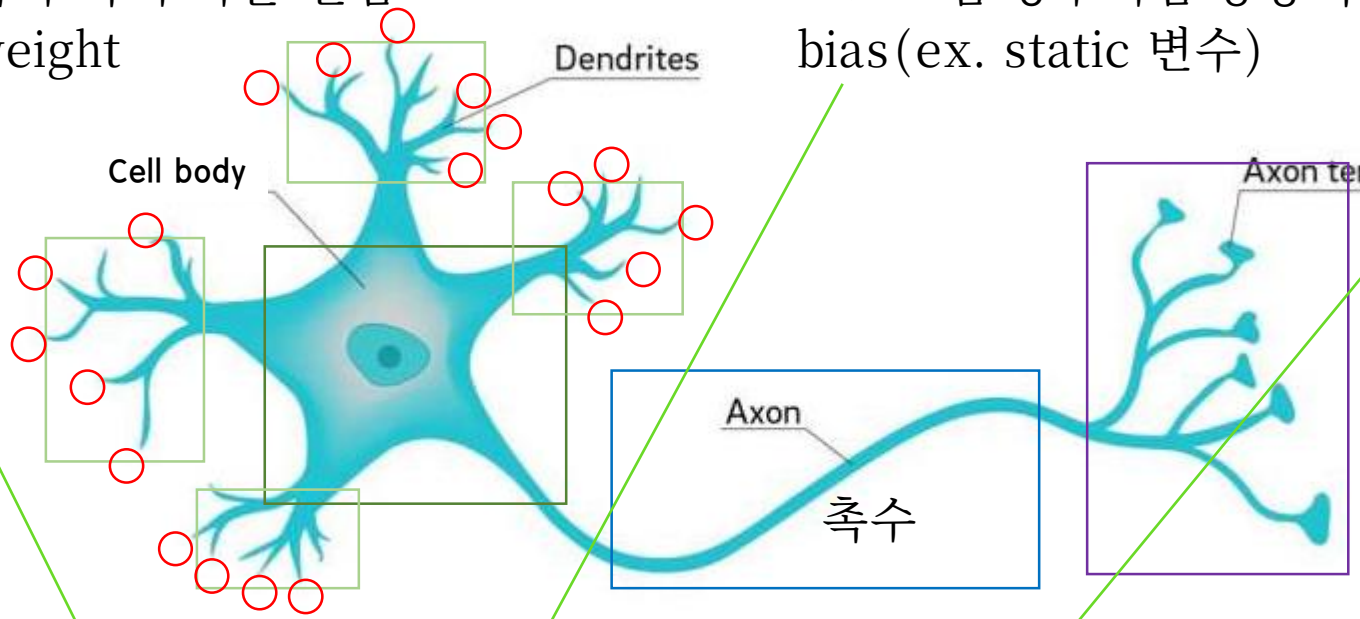
인공신경망
(Artifical)
N.N
(FCN)

인간신경망

- Structure of a neuron

해석1.
들어온 값에 따라 각기 다른 신념으로
중요도 부여 weight

해석2.
프로그램 상수처럼 공통적인 신념으로 지정
bias(ex. static 변수)



해석3.
값이 어느 정도 수치 이상의 강한 자극이면 출력(역치값 12)
미세한(학습에 영향을 주지 않을 꺼 같은) 값은 출력으로 넘기지 않는다.

Input (x)	Functions 1 (wx)	Functions 2 (wx +b)	Functions 3 (activ (wx+b))	Output (active (wx+b))

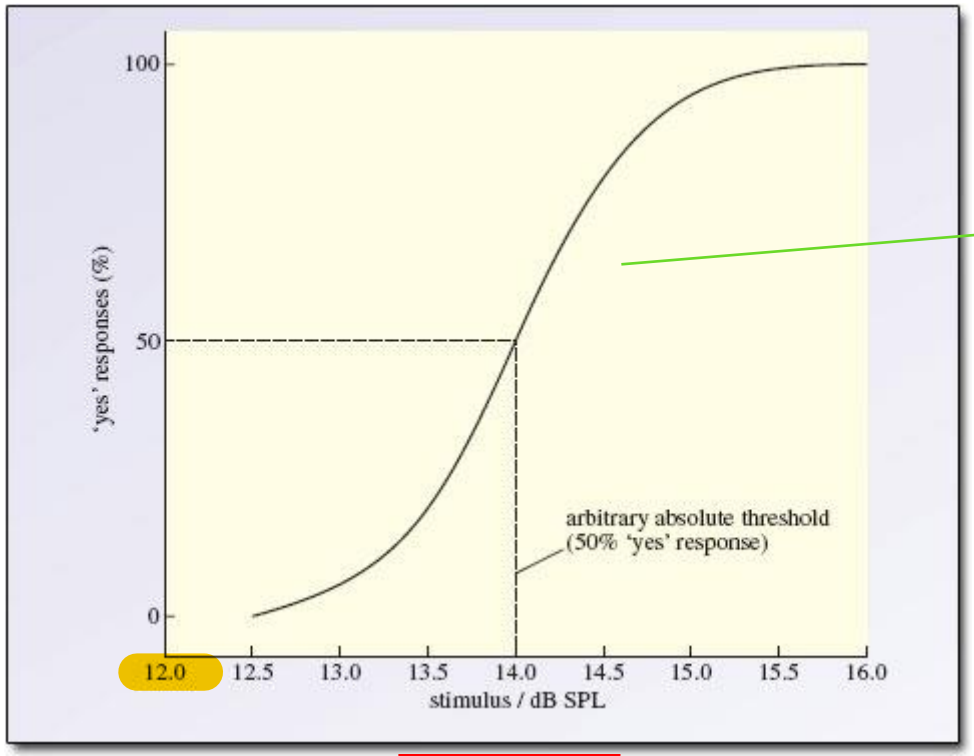
Neuron

임계치

- Neuron threshold

Activation Function

파형이 중요!!



Activation Function
비선형

Output
(activ (wx + b))

임계치 이하의 값은 버림

Input
(wx + b)

Neuron.png참고

Neuron

● : accuracy를 엄청 올려두고 더~올리고 싶을 때 이것들을 사용하면 올라갈 때도 있다.

비선형을 끼워 넣어서 하나하나의 layer를 따로 인식시키기 위해

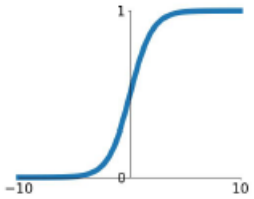
- Activation functions

인간의 활성화 함수와 거의 동일

- > 예전엔 많이 사용했지만 지금은 별로 사용하지 않음

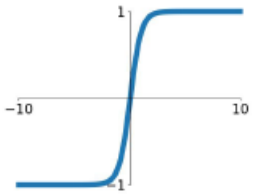
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

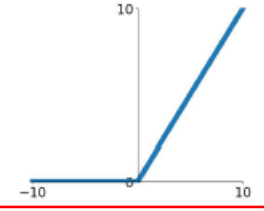
$$\tanh(x)$$



Recurrent Neural Network
Auto-Encoder (Segmentation)

ReLU

$$\max(0, x)$$



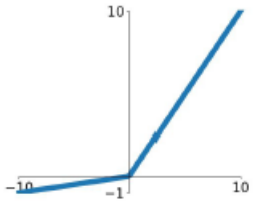
Convolutional Neural Network

0에서 발산하는 함수

Deep에서는 무조건 ReLU사용
(논문에서도~)

Leaky ReLU ●

$$\max(0.1x, x)$$

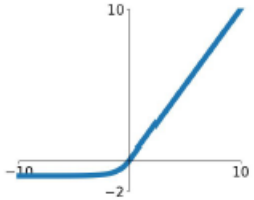


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

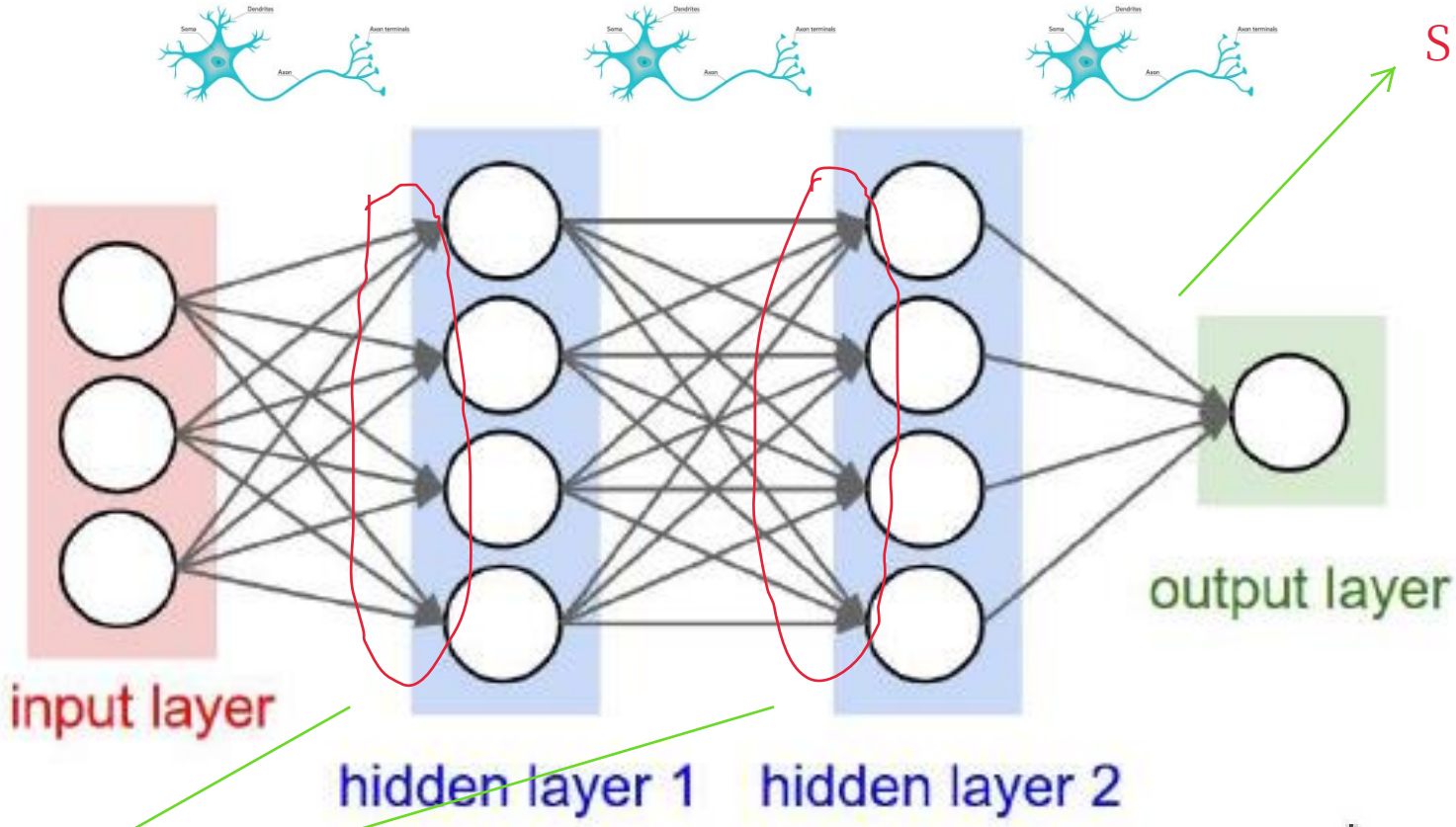
ELU ●

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



선형의 한계.png
선형의 한계 설명.txt 참고

Neural Networks (Fully-connected network)



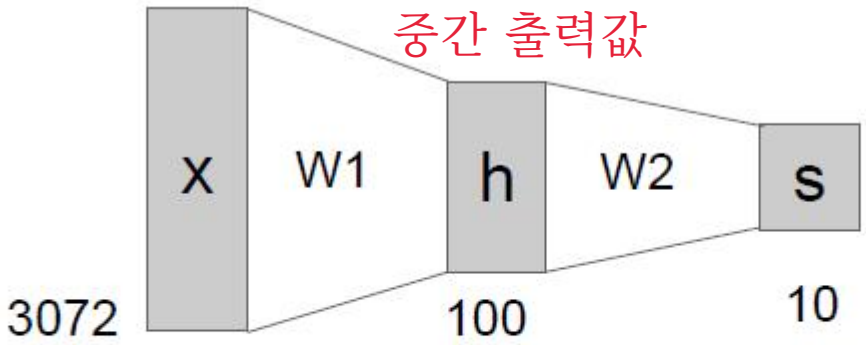
Activation Fuction

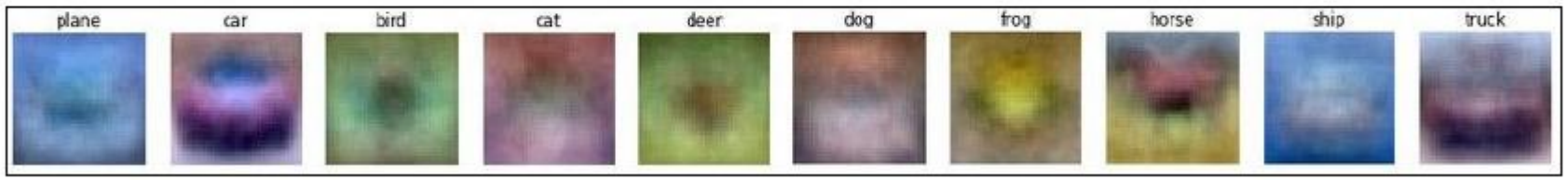
"Fully-connected" layers ⟹ $f\left(\sum_i w_i x_i + b\right)$

Neural Networks (Fully-connected network)

$$f = Wx$$

$$f = W_2 \max(0, W_1 x)$$



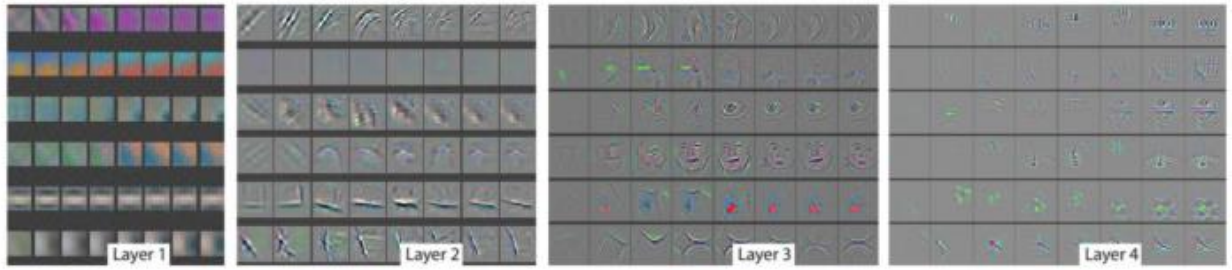


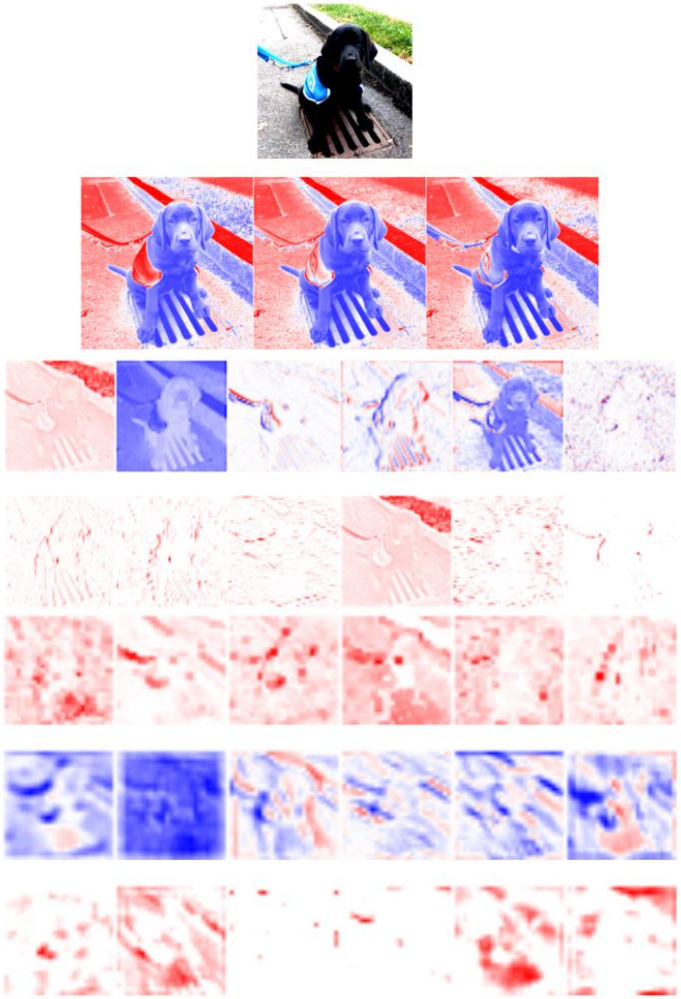
〈 Feature map 〉

Neural Networks (Fully-connection network)

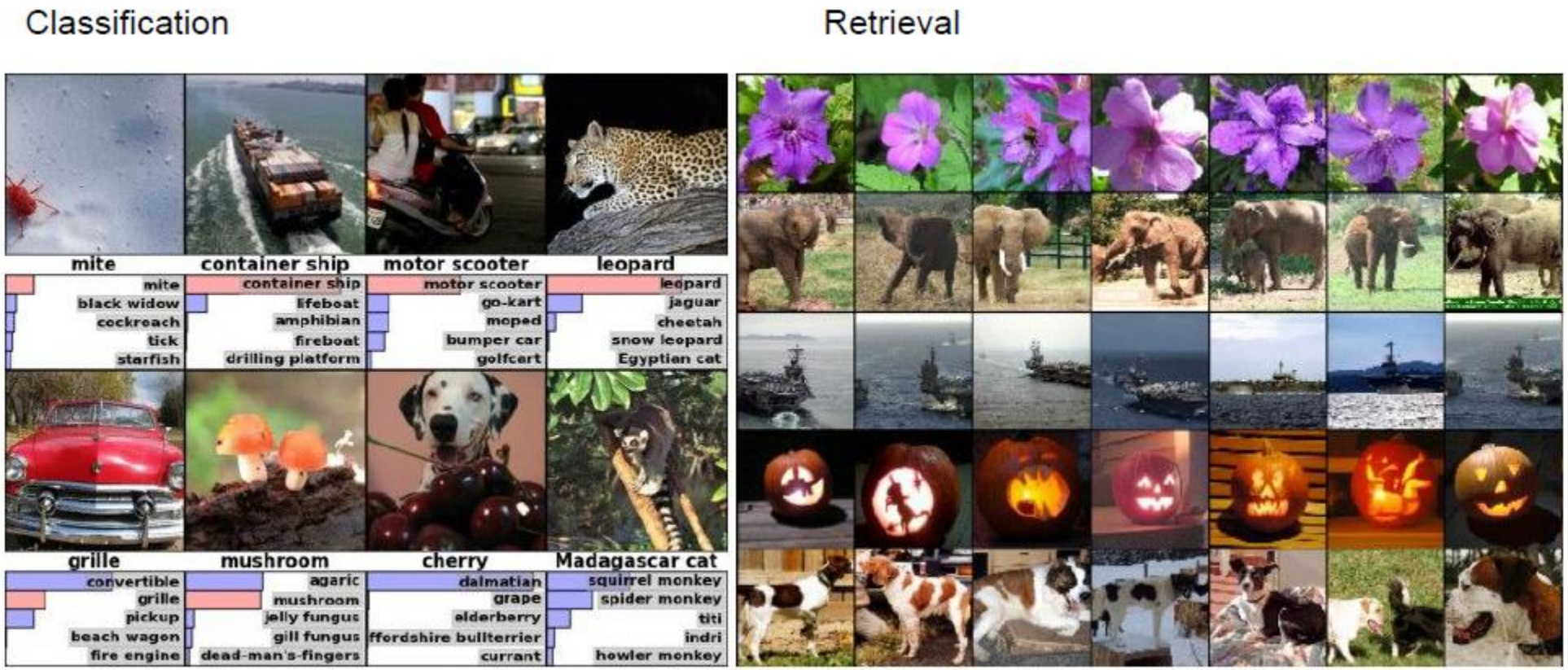
- Feature-map

  - Output of the layer
  - Representation of object



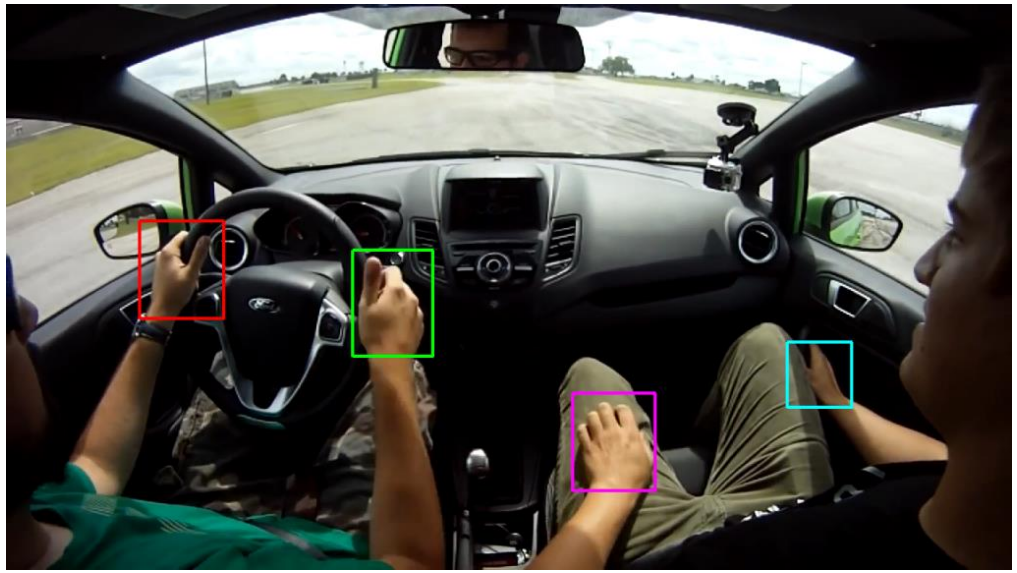


Convolutional Neural Network

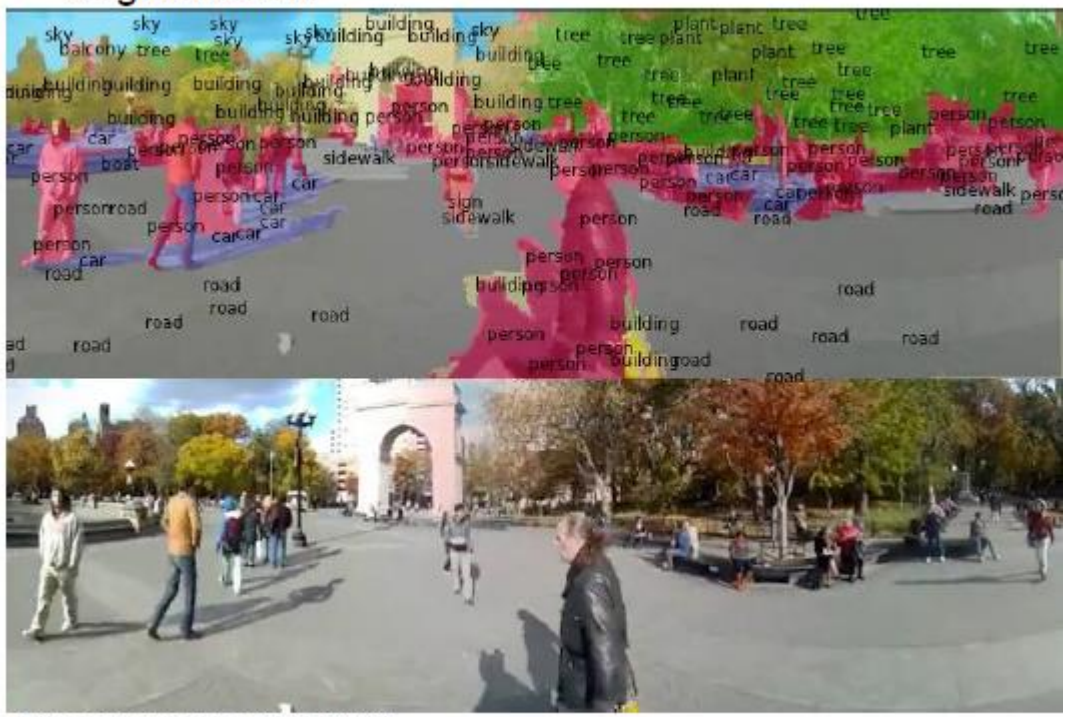


Convolutional Neural Network

Detection



Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]

Convolutional Neural Network

Motion Classification