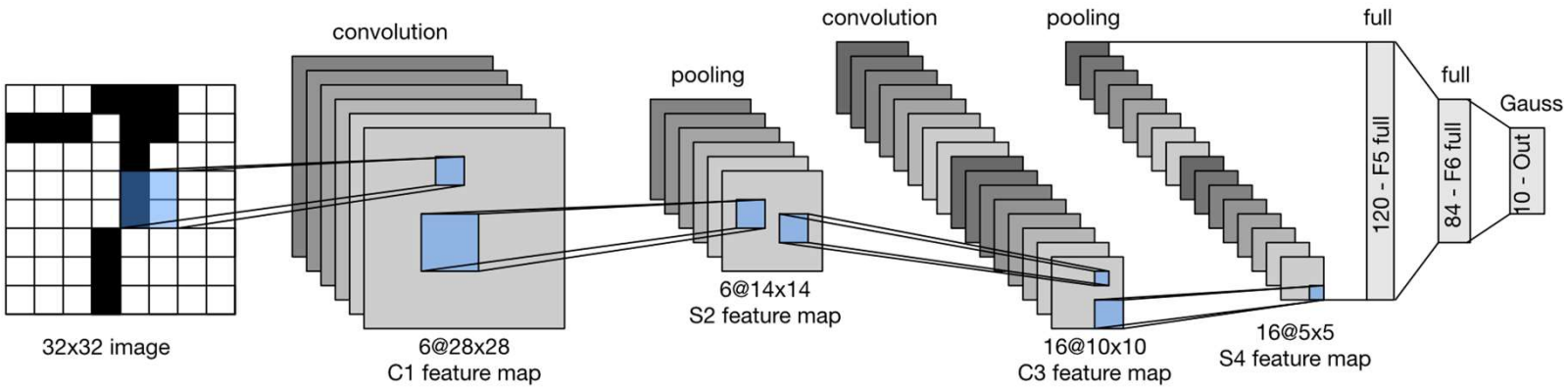
CNN (Convolutional Neural Network)

CNN (Convolutional Neural Network, 합성곱 신경망)

Yan Le Cunn - 1998

- LeNet : 5 개층
- ResNet : 152 개층



입력 **합성곱** **풀링** **합성곱** **풀링** **완전연결**

Multiple Levels of Abstraction

Pixel 정보 인식

↓

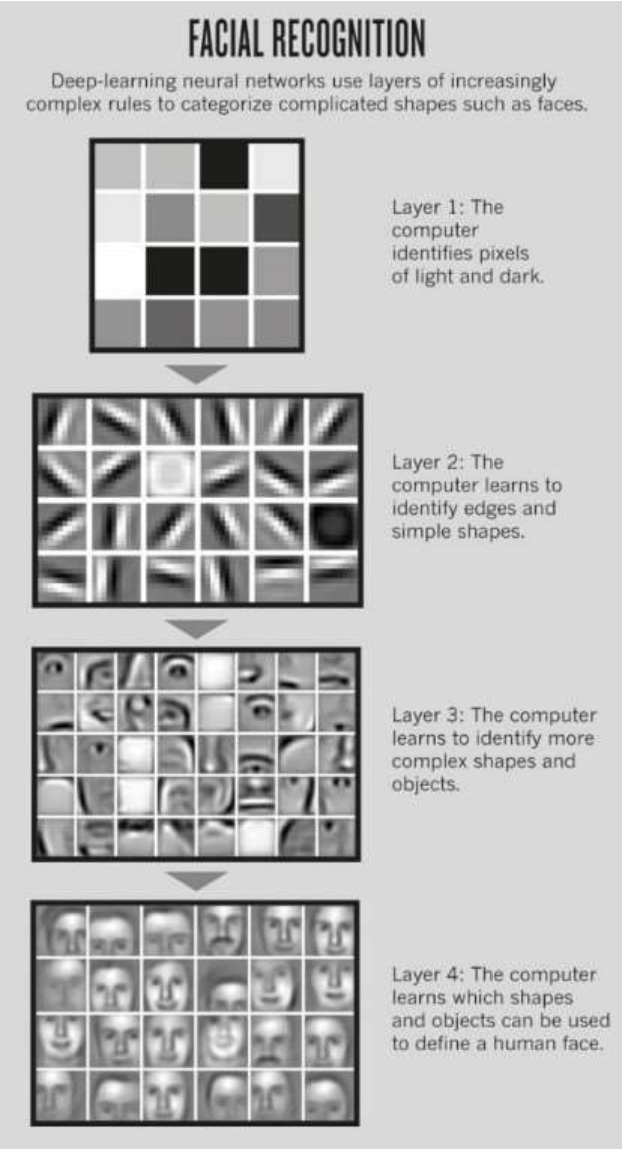
Edge/Simple Shape 특성 학습

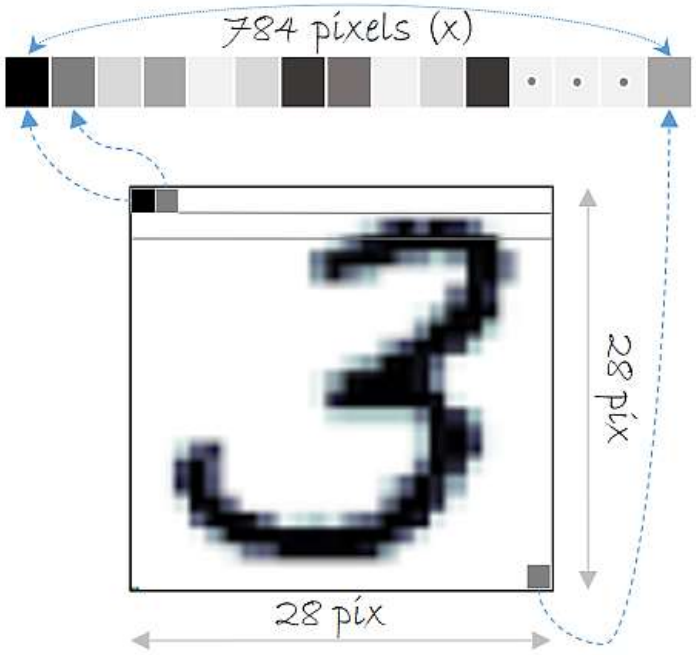
↓

Complex Shape 특성 학습

↓

얼굴인식에 필요한 특성 학습





이전 실습 모델에서의 input image 처리

60000 x **28 X 28** → reshape(60000, **784**)

Image Data 의
공간적, 지역적 특성 상실

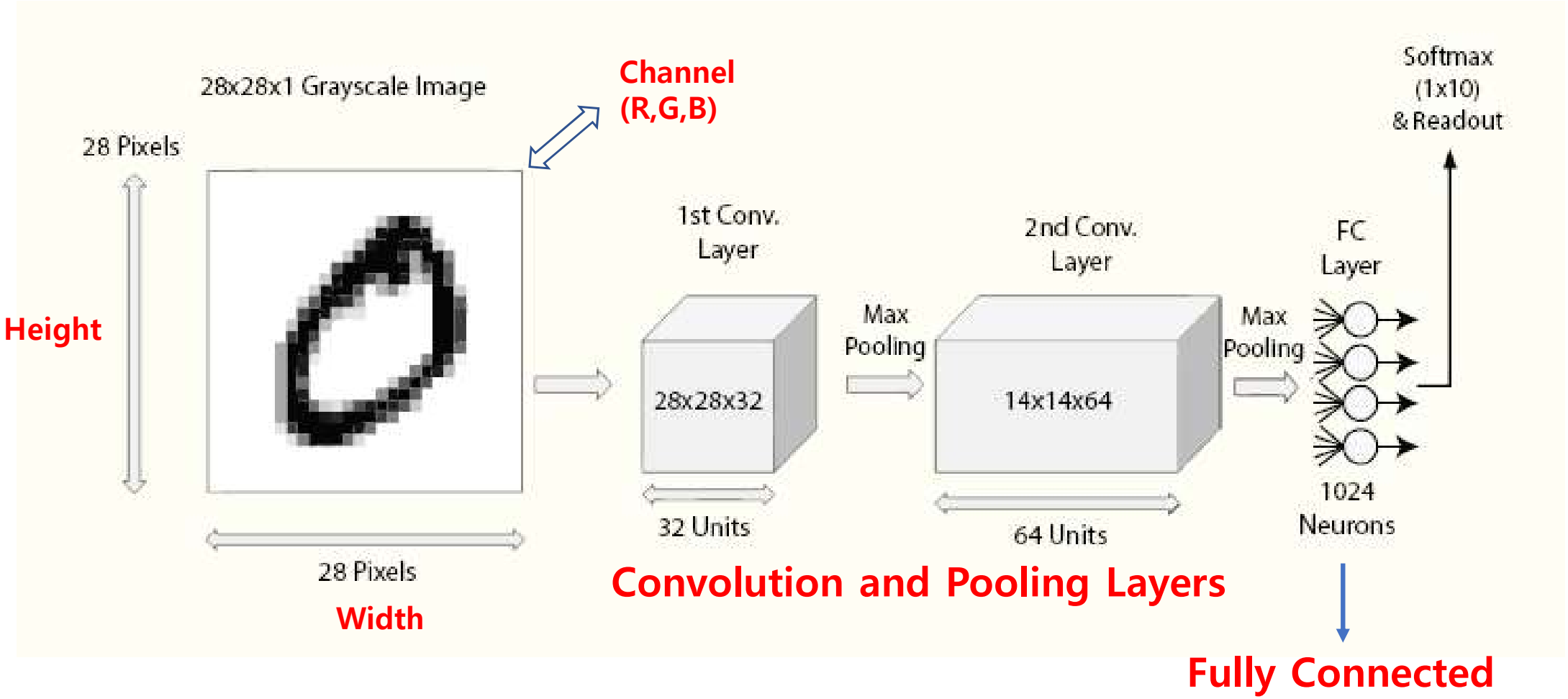
If 1 mega pixel ➔ 1,000 x 1,000 x 3 = 3 million features !!!

➔ **300 만 차원 x Layer 수 x 각 Layer 의 Neuron 수**

➔ **계산량 급증**

➔ Image Data 를 처리하기 위한 특별한 구조의 Neural Network 필요

2. Dimensions of Layers



CNN 의 특별한 Layers

- **Convolutional Layer (합성곱층)**

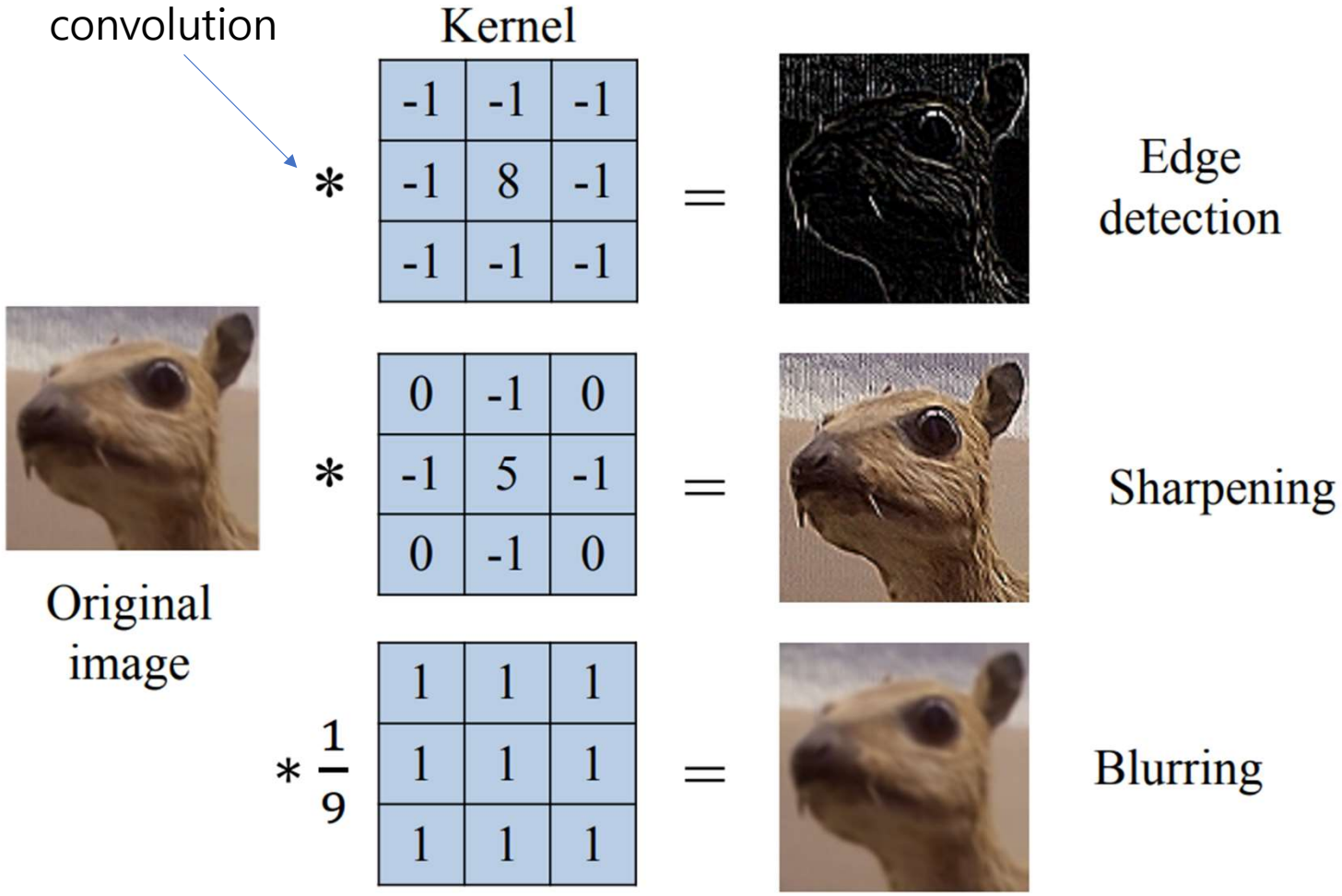
  - Image 정보의 공간적 지역 특성 보존
  - Filter (Kernel) 을 이용한 이미지 특성 추출

- **Pooling Layer (풀링층)**

  - Image data 의 정보 손실 없는 압축

  ➔ 계산량 및 메모리 사용량 축소, 파라미터의 수 감소 (과적합 방지)

Kernel(Filter) 의 특성 추출 예

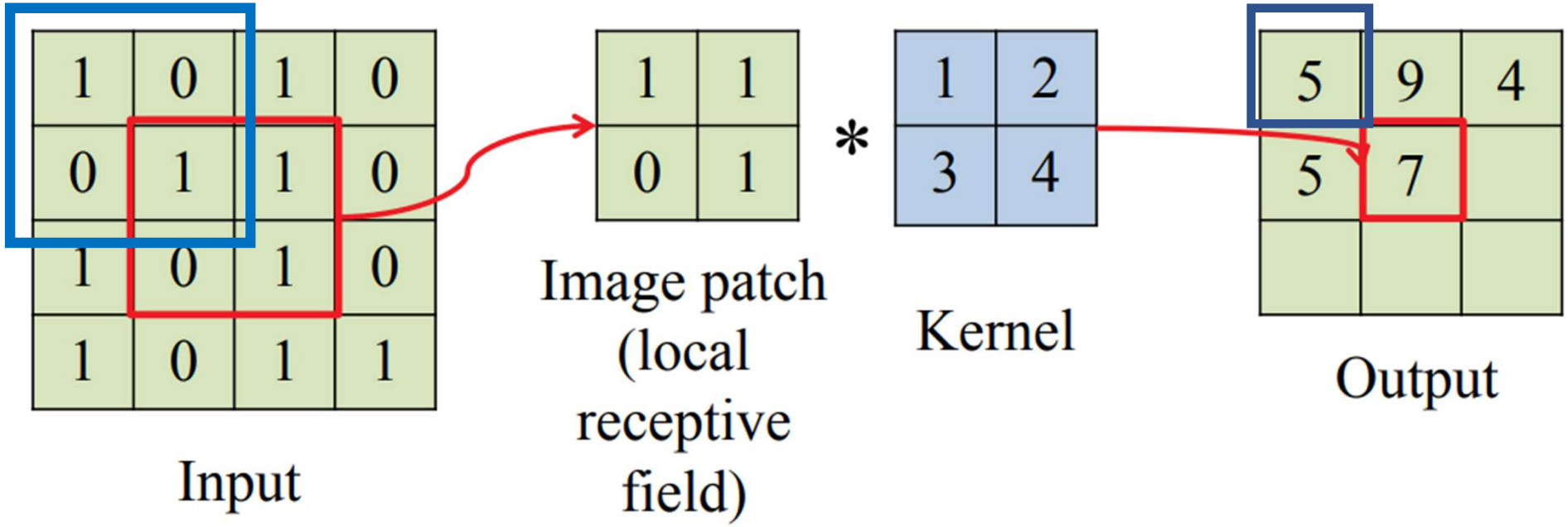


- CNN 이전에는 edge detection filter 를 computer vision 전문 가들이 모두 manually 만들어 줌

  ex) 수직 / 수평 / 45 도 / 명암 구분 filter 등

  ➔ Neural Network 은 훨씬 더 다양한 특성의 filter 들을 back-propagation 을 이용하여 자동으로 학습하고 스스로 만들어 냄

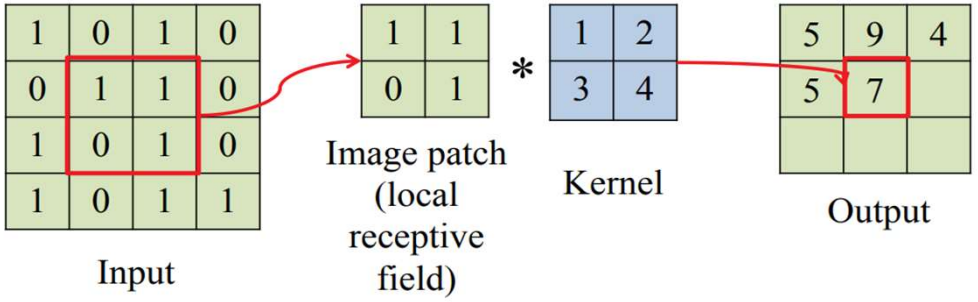
How Convolution works ?



Kernel_size=(2, 2), stride=(1, 1), No padding

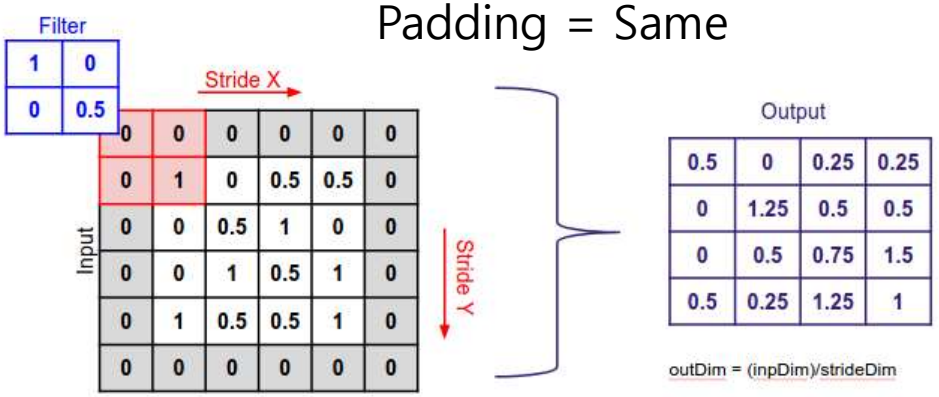
- Convolution 때 마다 image 의 edge 정보가 소실 ➔ padding 적용하여 해결
- “Valid” convolution : No padding
- “Same” convolution : Input size = Output size (➔ Input 의 주위에 0 pixel padding)

Padding = Valid



4 x 4 2 x 2 3 x 3

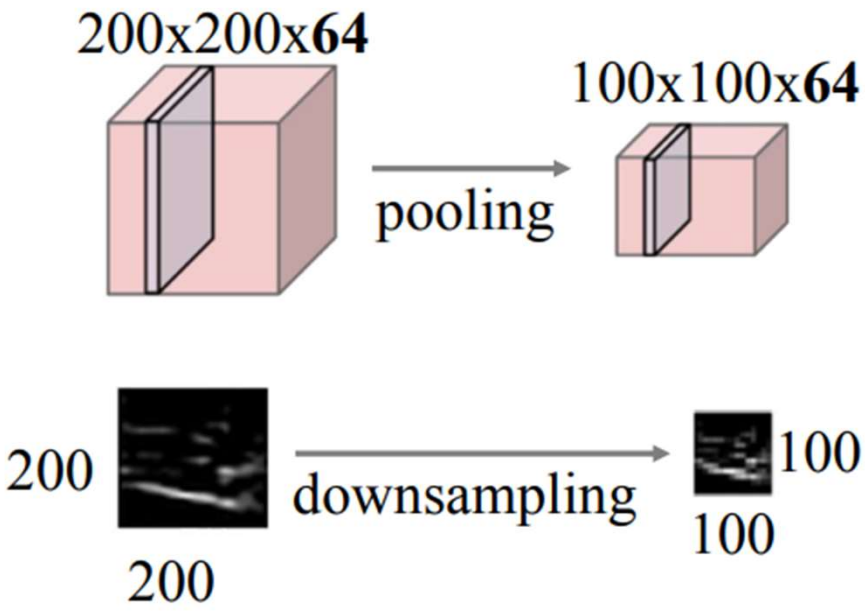
Padding = Same

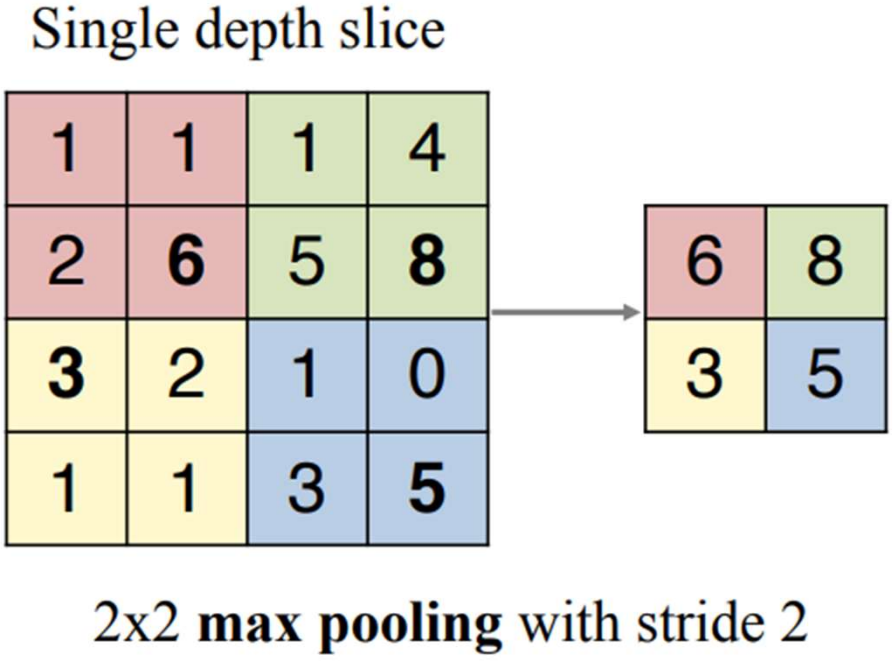


4 x 4 4 x 4

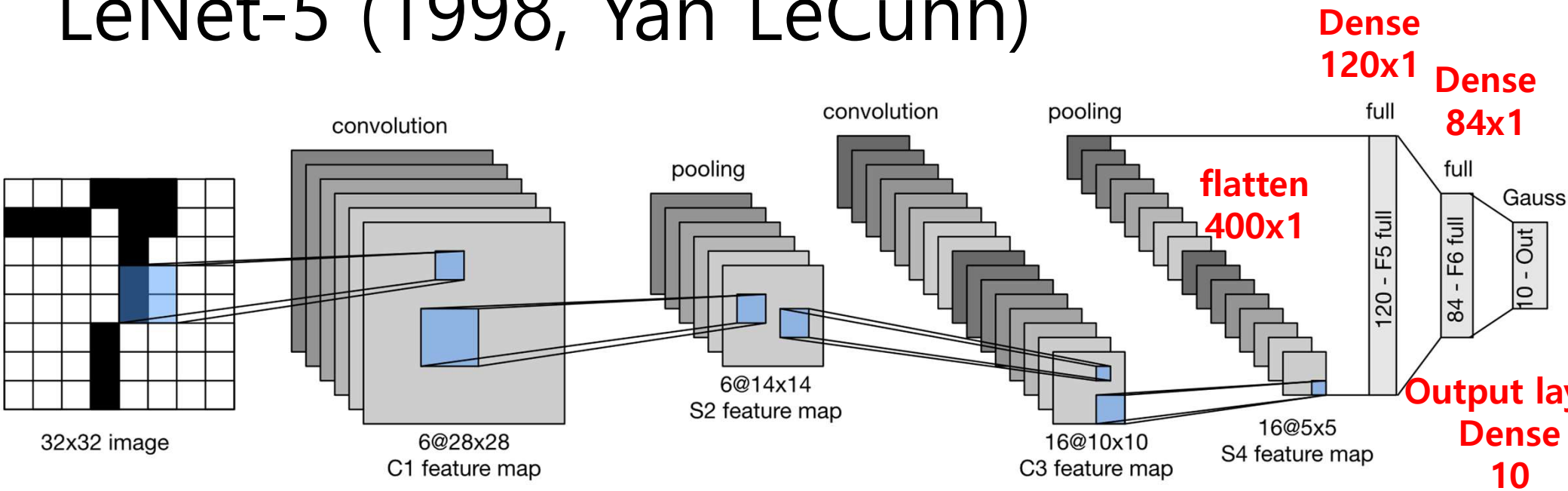
How Pooling works ?

- Pooling 의 뉴런은 가중치가 없음
- 최대, 평균을 이용한 이미지 subsampling (부표본 작성)





LeNet-5 (1998, Yan LeCunn)



Dense 120x1

Dense 84x1

Output layer Dense 10

1st layer

2nd layer

Input layer
32 x 32
Gray Scale Image

Convolution
6 filters
(5 x 5,
stride 1)
28 x 28 x 6

Average pooling
Filter
(2 x2, stride 2)

14 x 14 x 6

Convolution
16 filters
(5 x 5,
stride 1)
10 x 10 x 16

Average pooling
Filter
(2 x2, stride 2)

5 x 5 x 16

Famous CNN models

- Alex Net – 2012 년 ILSVRC(ImageNet Large Scale Visual Recognition Competition)대회 우승
- GoogleLeNet(Inception Net) – 2014 년 ILSVRC 대회 우승
- ResNet – 2015 년 ILSVRC 대회 우승 (152 개 층)
- MobileNet – mobile device 용 pre-trained model (ImageNet 20,000 개 classes)
- VGG-16 : Keras built-in pre-trained model (2014 년, 16 layers)

실습 : Le Net 을 이용한 손글씨 인식

1. Keras 를 이용한 Le Net 구축

2. 구축한 Le Net model 을 이용하여 Mnist 손글씨 분류

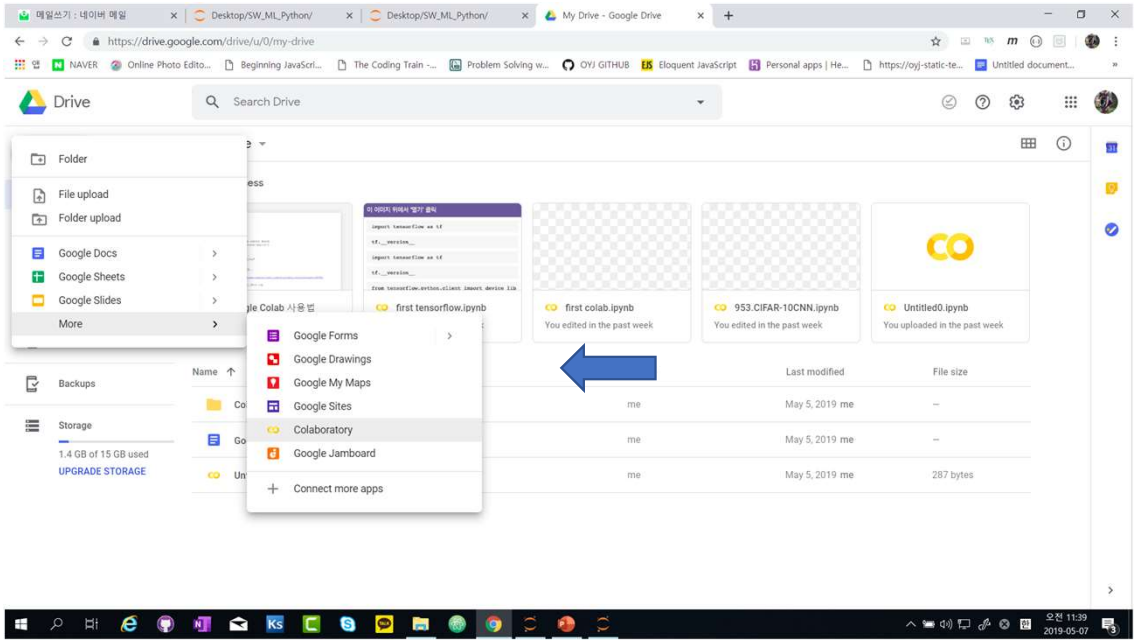
Google Colab

- Free GPU 제공
- Google Drive 와 연동
- Jupyter Notebook 환경
- Deep Learning beginner 를 위한 최적의 환경
- 각종 snippet 제공

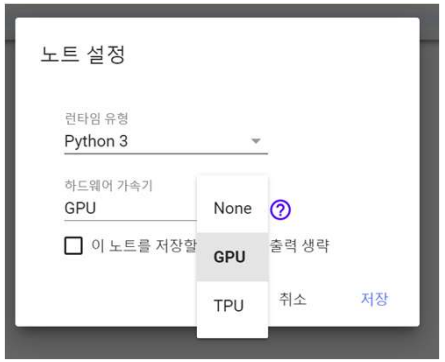
Google Colab 사용하기

- Google drive

  ➔ Colaboratory 연결



- 런타임 → 런타임 유형변경 → GPU 선택



실습 : Deeper CNN 을 이용한 CIFAR-10 분류

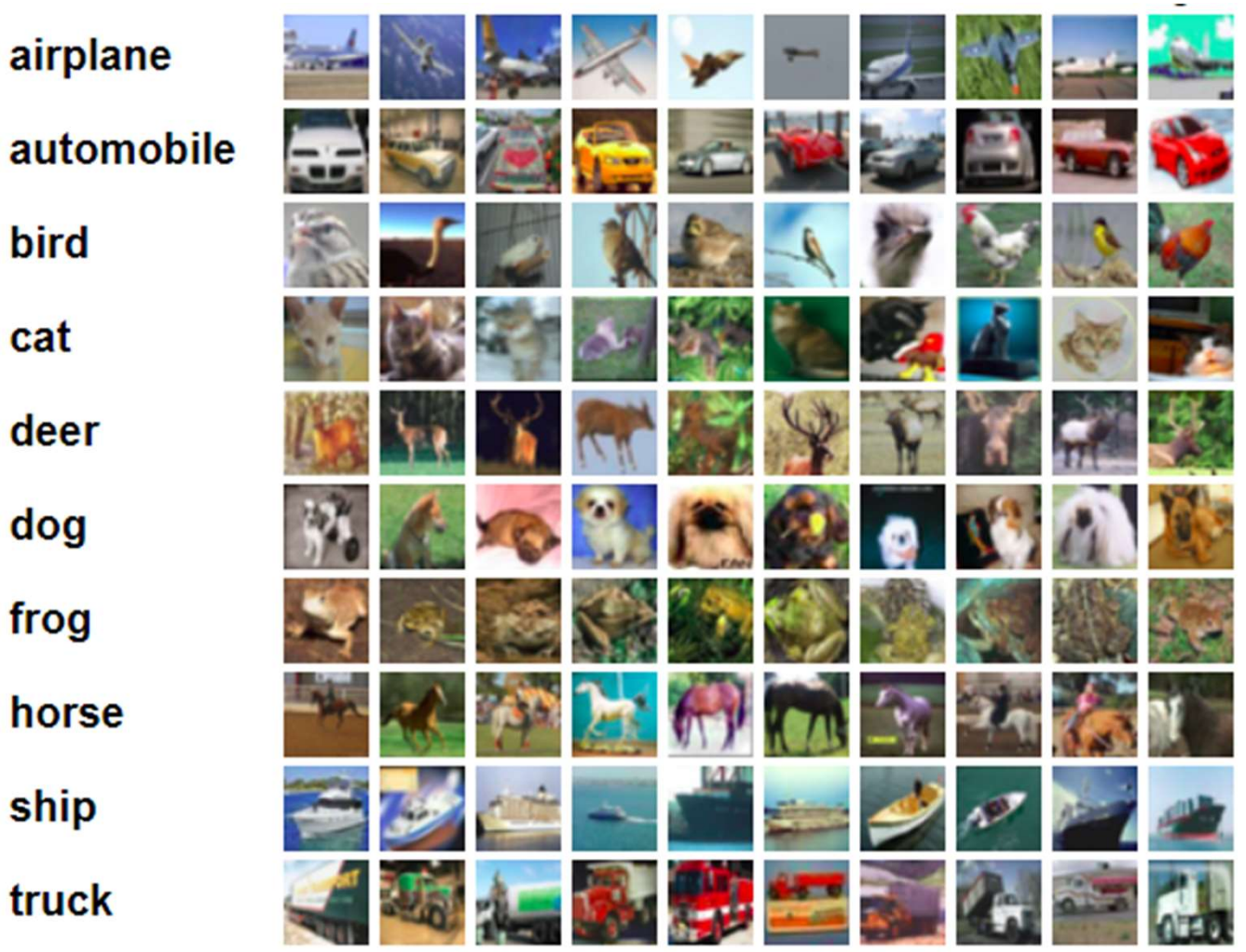
1. CIFAR-10 dataset 은 32x32 color image 를 가진 10개의 class (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)

2. 각 class 별 6,000 개씩 total 60,000 개 image

3. Image 가 blur 하여 난이도 높음
   (최근 성적 : https://en.wikipedia.org/wiki/CIFAR-10)

4. Google Colab GPU 환경 이용

CIFAR-10 image dataset



Neural Network Architecture

Input layer
32 x 32 x 3

Conv2D
16 filters
(3, 3)
Padding=same
Activation : relu

MaxPooling2D
(2, 2)
Padding=same

Dropout : 0.2

Conv2D
32 filters
(3, 3)
Padding=same
Activation : relu

MaxPooling2D
(2, 2)
Padding=same

Dropout : 0.2

Conv2D
64 filters
(3, 3)
Padding=same
Activation : relu

MaxPooling2D
(2, 2)
Padding=same

Dropout : 0.2

Flatten

Dense
256
Activation : relu

Dropout : 0.5

Output

Dense
10
Activation : softmax

실습 : 구축한 model 의 weight 와 architecture 저장 및 loading

1. JSON / YAML file 로 model architecture 저장

2. JSON / YAML file 로 부터 model 복원

3. HDF5 file 로 model 의 parameter 저장 및 복원

What is JSON / YAML ?

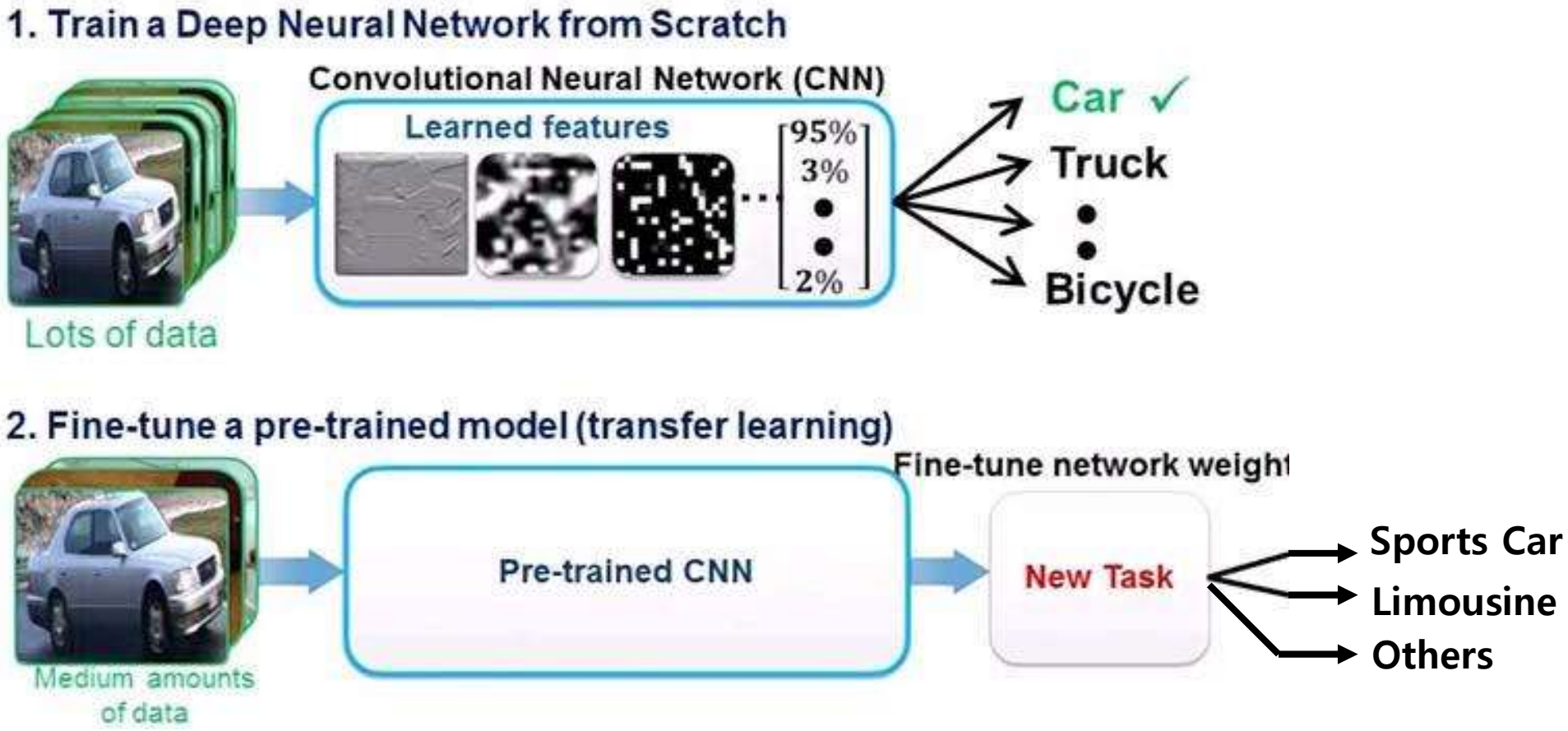
- JSON (**J**ava**S**cript **O**bject **N**otation)

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
    ]
}
```

- YAML (YAML Ain't Markup Language)

```
invoice: 34843
  date : 2001-01-23
  bill-to: &id001
  given : Chris
  family : Dumars
  address: lines: | 458 Walkman Dr. Suite #292
  city : Royal Oak
  state : MI
  postal : 48046
```

Transfer Learning



Transfer Learning 고려사항

- **목적에 맞는 dataset 선택**

  ex) Cat & Dog 구분 → ImageNet 에 포함

  Cancer cell 구분 → ImageNet 에 없음

- **보유 Data 의 Volume 고려**

  1. 모든 weight 새로이 training (Large Data 보유)
  2. Weight 의 일부만 training
  3. 마지막 layer 만 Fine-tuning (Small Data 보유)

실습 : Pre-trained model 을 이용한 image 분류

1. Keras 에 내장된 VGG-16 pre-trained model 을 이용
2. 임의의 image 를 VGG-16 의 입력 spec 에 맞도록 resize
3. decode_predictions 를 이용한 결과값 비교

VGG16 Structure

(224, 224, 3)

(1, 1000)

Conv Layer : 3x3 filters, stride=1, "same"

Max-pool : 2x2 filters, stride=2

Deep Learning

Sequence Model

What is sequence data ?

- Speech recognition : 파동의 연속 → 단어의 연속으로 변환
- Music generation : 연속된 음표 출력
- Sentiment classification : Text → 평점, 부정/긍정 판단
- DNA 분석 : 염기서열 → 질병유무, 단백질 종류 등
- 자동 번역 : 한국어 → 영어
- Video activity recognition : 연속된 장면 → 행동 판단
- Financial Data : 시계열자료 → 주가, 환율 예측 등

Problem of Standard Neural Network

- 입력의 길이와 출력의 길이가 고정되어 있음

  → 따라서 standard NN 에서는 maximum input length 정하고
  초과하면 truncate, 모자라면 padding 을 해 주어야 한다.

- 입력 데이터의 순서를 무시 (모든 observation 은 독립적임)

RNN (Recurrent Neural Network)

RNN (Recurrent Neural Network)

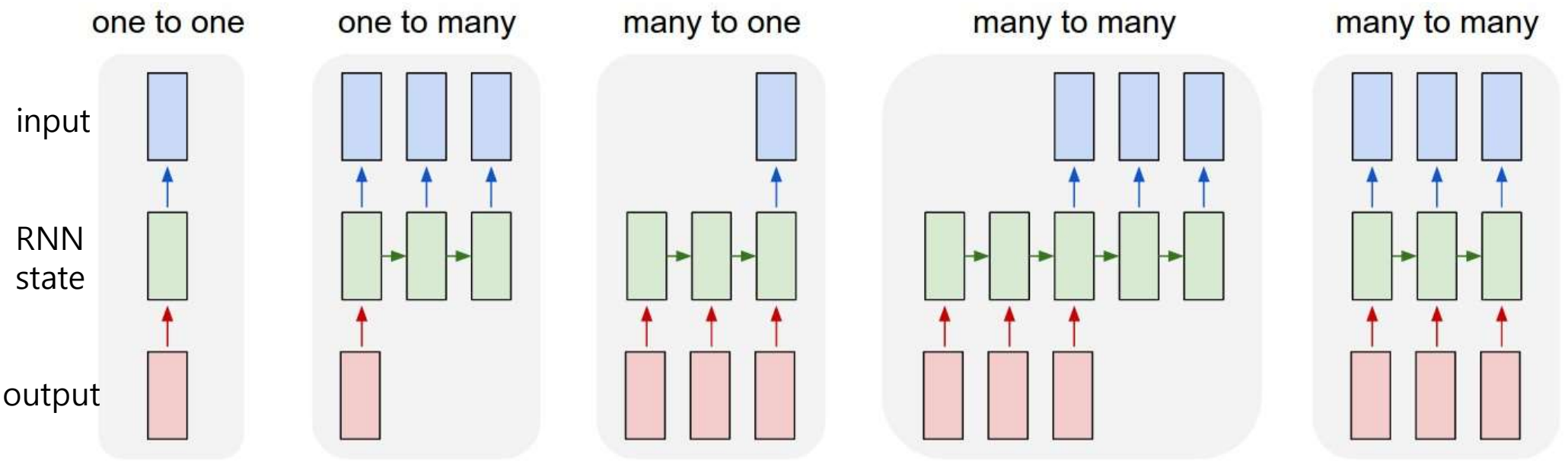
- 시퀀스 데이터에 특화

- '기억' 능력을 갖고 있음
  * 네트워크의 기억 - 지금까지의 입력 데이터를 요약한 정보
  (새로운 입력이 들어올때 마다 네트워크는 자신의 기억을 조금씩 수정)

- 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보
  (사람의 시퀀스 정보 처리 방식과 비슷, 기억을 바탕으로 새로운 단어 이해)

- 이 과정은 새로운 단어마다 계속해서 반복 → Recurrent (순환적)

Different Types of RNN



예) 이미지 분류

이미지로 부터 문장 생성, 작곡

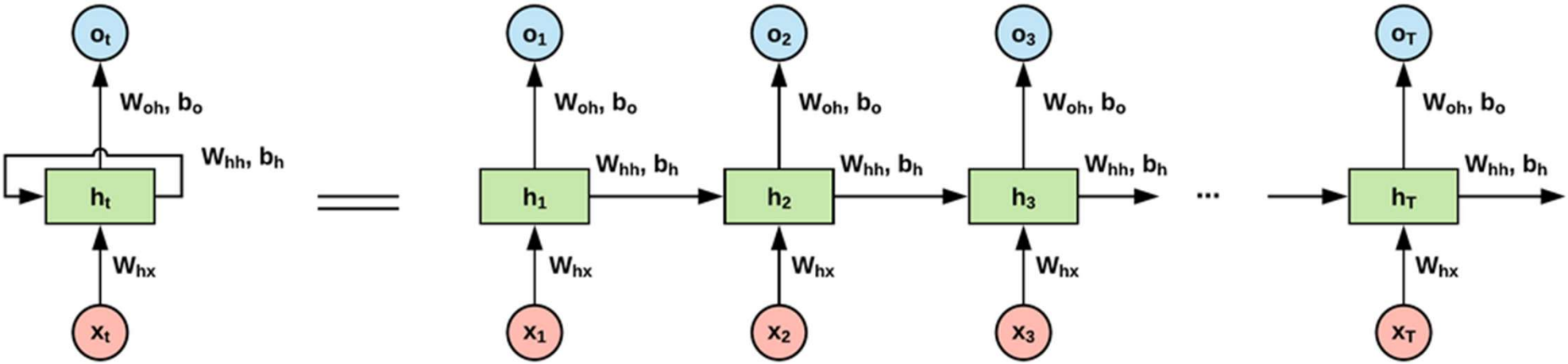
감성 분석 (sentence → positive/negative)

기계 번역 (영어문자 → 한국어 문장)

video 각 frame 에 label 생성

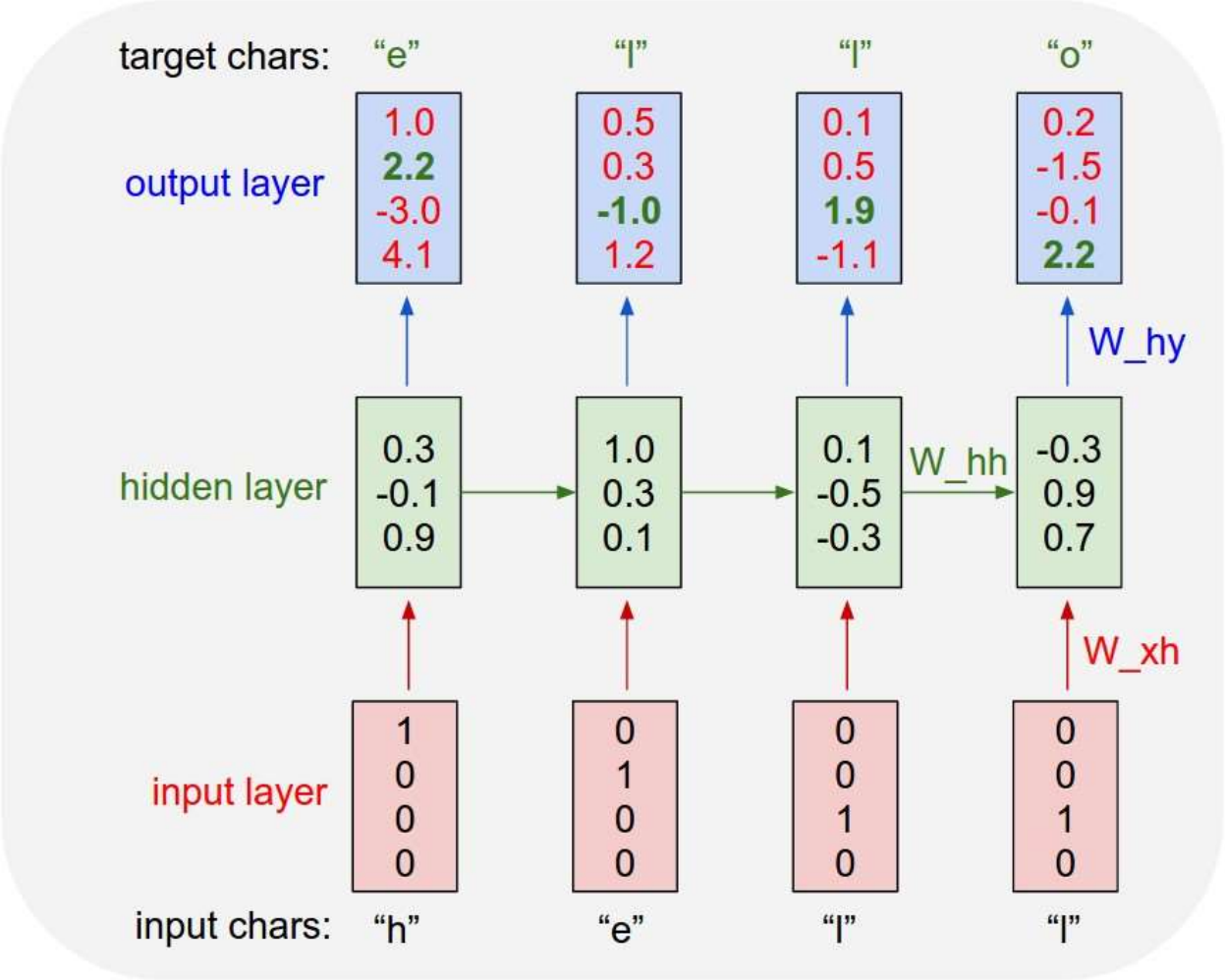
RNN (Unfold 표시)

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$
$$O_t = softmax(W_o h_t)$$



- RNN 을 순서대로 펼쳐 놓으면 weight 를 공유하는 매우 deep 한 neural network 이 된다.
- BPTT (Backpropagation Through Time) 으로 parameter 학습

How RNN is trained ?



- "h", "e", "l", "o" 4 개 문자만 있다고 가정
- 각 문자를 One-hot encoding
- "h" 를 시작 문자로 주면 "hello" 가 출력 되도록 훈련
- 각 time step 의 target character 는 "hello" 내의 next character
- Gradient descent 와 backpropagation 을 통해 output layer 의 target score 가 증가되도록 함 (green color)
- "l" 다음의 character 는 현재의 "l" 만으로 판단할 수 없음 (history 필요)

실습 : 이상한 나라의 엘리스 연속 단어 생성기

- SimpleRNN 이용, Many-to-One type

- 161793 글자로 이루어진 text 를 10 글자 단위로 잘라 input data 를 만들고 뒤 따라오는 글자를 label data 로 만들어 supervised learning

  ex) “alice lear” - “n”

  “lice learn” - “e”

  “ice learne” – “d”

- Validation 은 seed 가 되는 10 글자 data 를 주고 이어서 만드는 100 글자 문장이 의미 있는지 여부 육안으로 검토

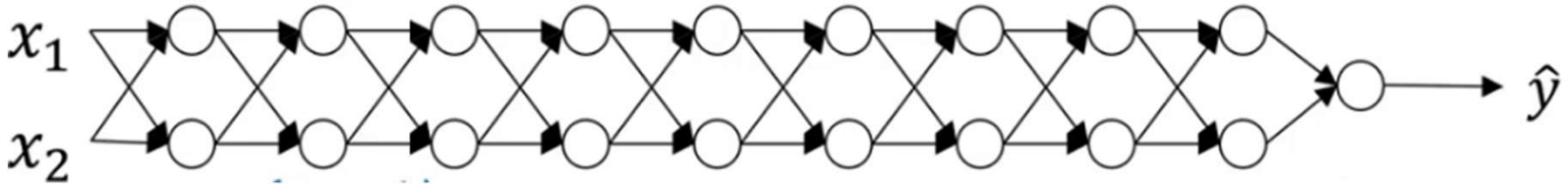
  ex) seed : “alice look”

  output : “alice looked at the mouse was a trite than she ..”

- 개개의 character 를 단어로 확장하면 문장 쓰기 가능

Vanishing & Exploding Gradient

- 매우 deep 한 network 을 훈련시킬 경우 앞부분 Layer weight 의 미분값 크기가 매우 작아지거나 커지는 현상



$$\hat{y} = w^{[L]}\, w^{[L-1]}\, w^{[L-2]} \ldots\ldots\ldots\ldots\ldots\ldots\ w^{[3]}\, w^{[2]}\, w^{[1]} \mathrm{X}_1$$

if $w^{[1]} = 1.5 \rightarrow 1.5^L$ (Explode)
if $w^{[1]} = 0.5 \rightarrow 0.5^L$ (Vanish)

- 문제점 – Exploding – 훈련 자체가 안됨
  Vanishing – 경사하강법이 매우 느리게 진행

Solutions of Vanishing Gradient

- Relu 사용

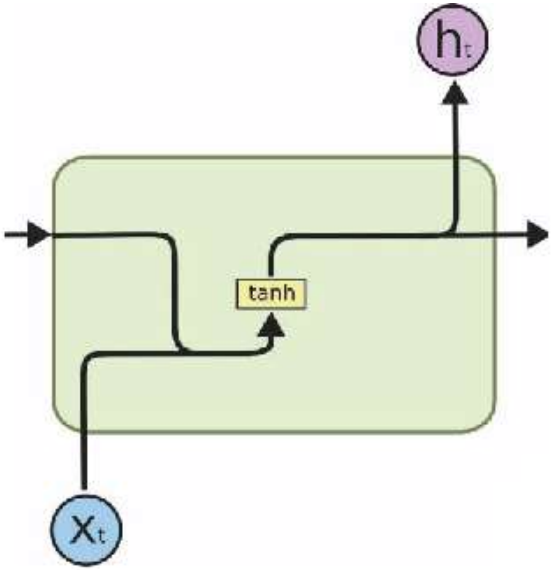
- Weight 의 신중한 초기화

  Ex) Xavier (Glorot) Initializer with Tanh, He Initializer with Relu

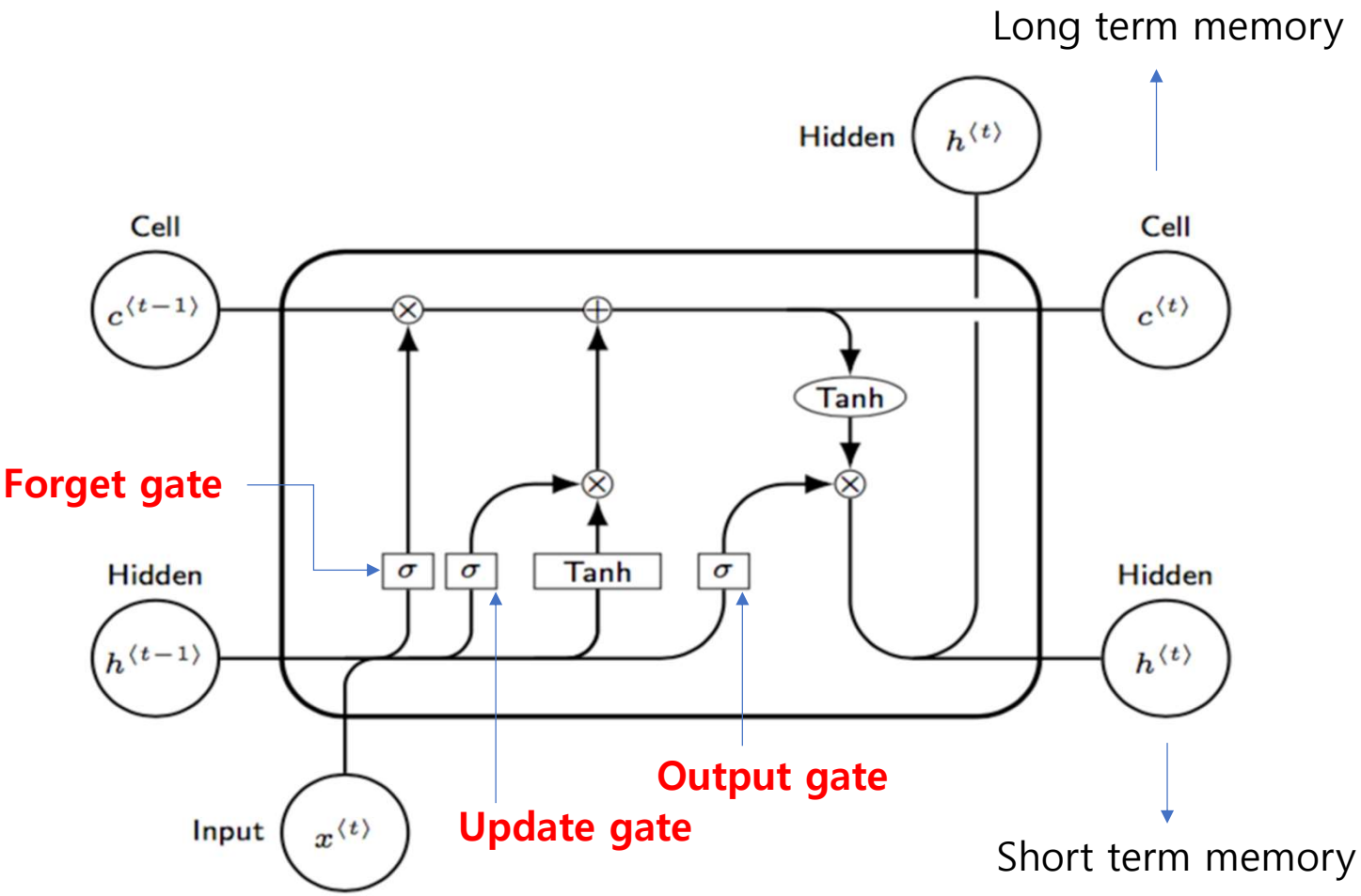
- LSTM (Long Short Term Memory) /
  GRU (Gated Recurrent Unit) 사용

  ➔ Vanishing Gradient + memory

SimpleRNN



LSTM (Long Short-Term Memory)



LSTM 내부 구조

- Input – 이전 step 의 output + new data

  $\tilde{C}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$ ➔ 새로운 cell status 후보

- Update gate – 새로운 input 을 어느정도 받아들일지 결정 (0-무시, 1-전체)

  $\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$

- Forget gate – 내부 state 를 어느정도 기억할지 결정 (0-forget, 1-전체 기억)

  $\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$

- Output gate – cell state 의 어느 부분을 output 으로 보낼지 결정

  $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$

- $C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + \Gamma_f * C^{<t-1>}$

- $a^{<t>} = \Gamma_o * \tanh C^{<t>}$

실습 : 이상한 나라의 엘리스 연속 단어 생성기 - LSTM

- 이전 연습문제의 SimpleRNN 을 LSTM 으로 변경
- 이전 결과물과의 성능 비교
- GPU 환경 필요 (장시간 소요)

NLP (Natural Language Processing)

with LSTM

NLP(자연어처리)의 기본 용어

- Corpus (말뭉치)
  - 자연언어 연구를 위해 특정한 목적을 가지고 언어의 표본을 추출한 집합

- Tokenize
  - 문자열을 여러 개의 조각, 즉 여러 개의 Token(토큰, 단어)들로 쪼개는 것을 말한다.
    특수 token : <START>, <EOS>, <UNK>, <PAD>, etc

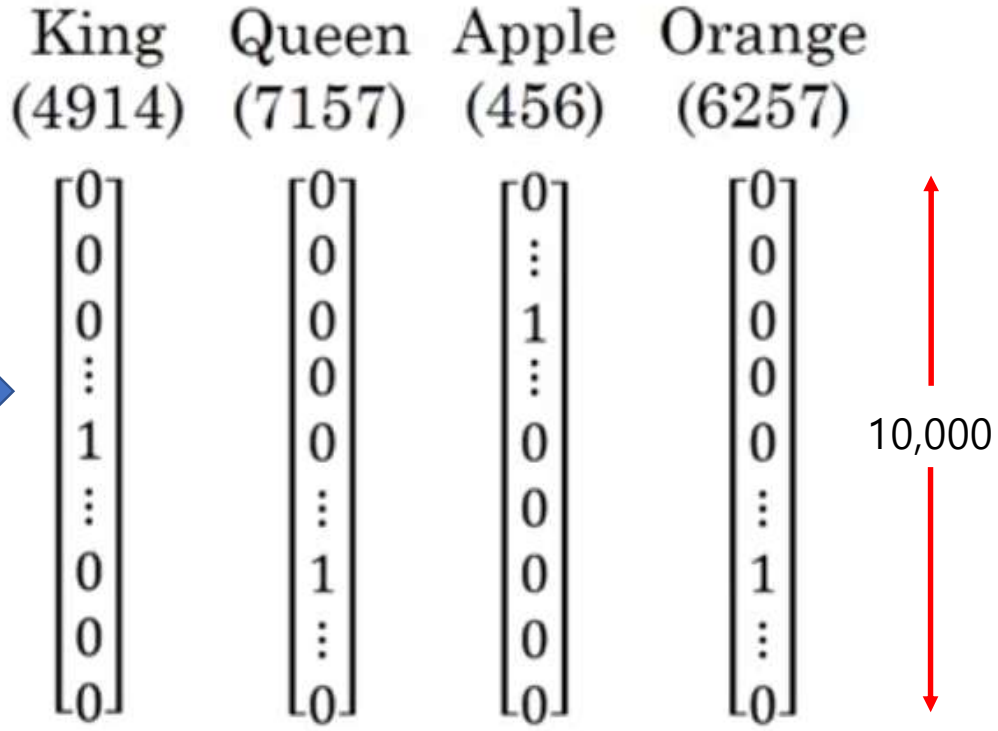
NLP (Natural Language Processing)

- 단어의 표시 방법

Vocaburary 사전

a -1
aaron – 2
.
apple – 456
.
king – 4914
.
orange – 6257
.
queen – 7157
.
zebra – 9,999
<UNK> - 10,000

One-Hot encoding



Word Embedding (Feature 화 표시)

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮				
size						
cost						
alive						
verb						

300

I like a glass of orange _____

I like a glass of apple _____

Man (5931) 의 300 dimension vector 표시

Embedding matrix (example)

학습된 featrues

words

	0	1	2	3	4	5	6	7	8	9	...	290	291	292	
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...	-0.283050	0.270240	-0.654800	0.10
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...	0.464470	0.481400	-0.829200	0.35
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...	-0.015404	0.392890	-0.034826	-0.72
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...	-0.285540	0.104670	0.126310	0.12
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...	-0.107030	-0.279480	-0.186200	-0.54
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...	-0.232860	-0.139740	-0.681080	-0.37
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...	0.048760	0.351680	-0.786260	-0.36
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970	-0.27
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730	-0.03
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931	-0.03
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167	0.05
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910	-0.26
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250	-0.31
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300	-0.46
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...	-0.470090	0.063743	-0.545210	-0.19
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878	0.00
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980	-0.29
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670	-0.24
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.099469	0.00
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543	0.15

20 rows × 300 columns

Word2Vec

- One-Hot-Encoding 의 문제점 - 단어간의 유사도가 표시되지 않음
- 단어를 vector 화 (Word Embedding) 하여 단어의 의미 표현
- 매우 큰 Corpus (ex, 10억, 100 억 단어) 에서 word embedding 자동 학습

ex) I love **king** of Korea. (skip-gram : window size = 2)
input : king → [0,0,0,0,….1,..0]
target : I love → [0,0,0,0,….1,..0] [0,0,0,0,0,0,…1,….0]
of Korea → [0,0,0,….1,..0] [0,0,0,0,0,…1,….0]

CBOW (Continuous Bag of Words) vs. Skip-gram

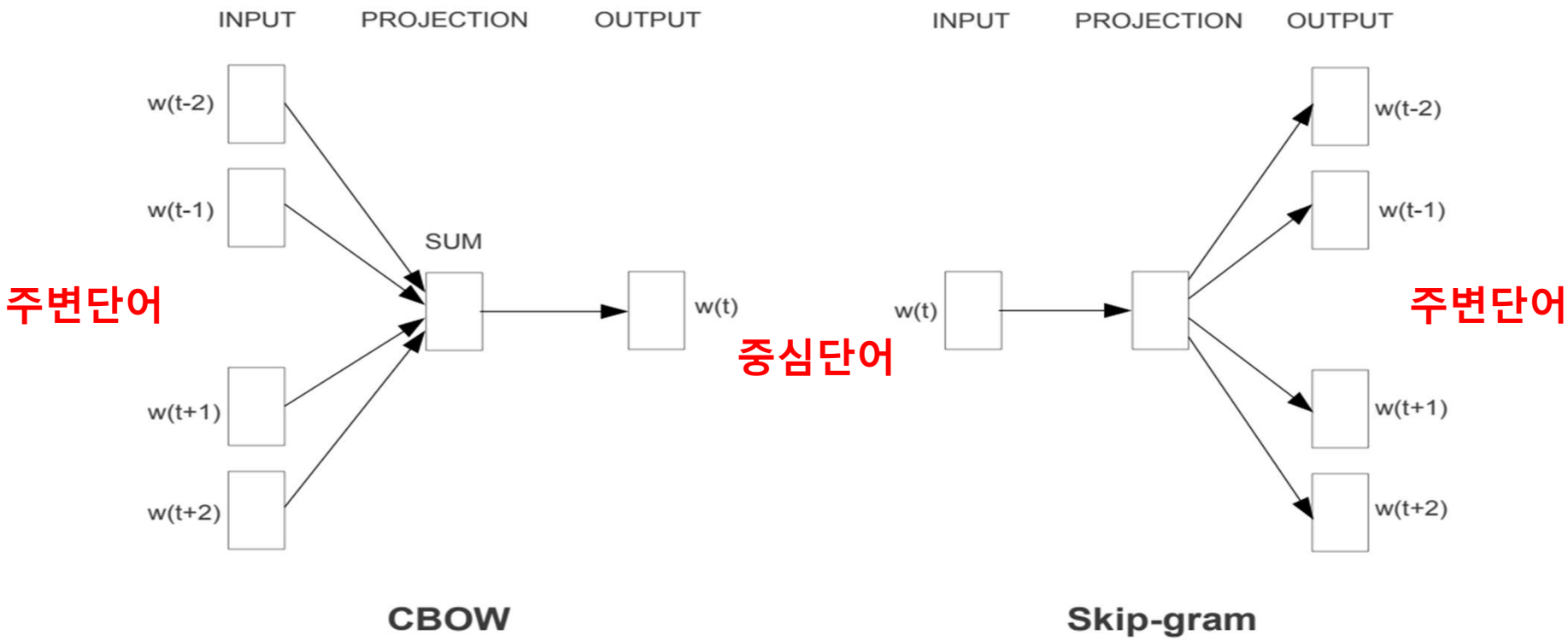
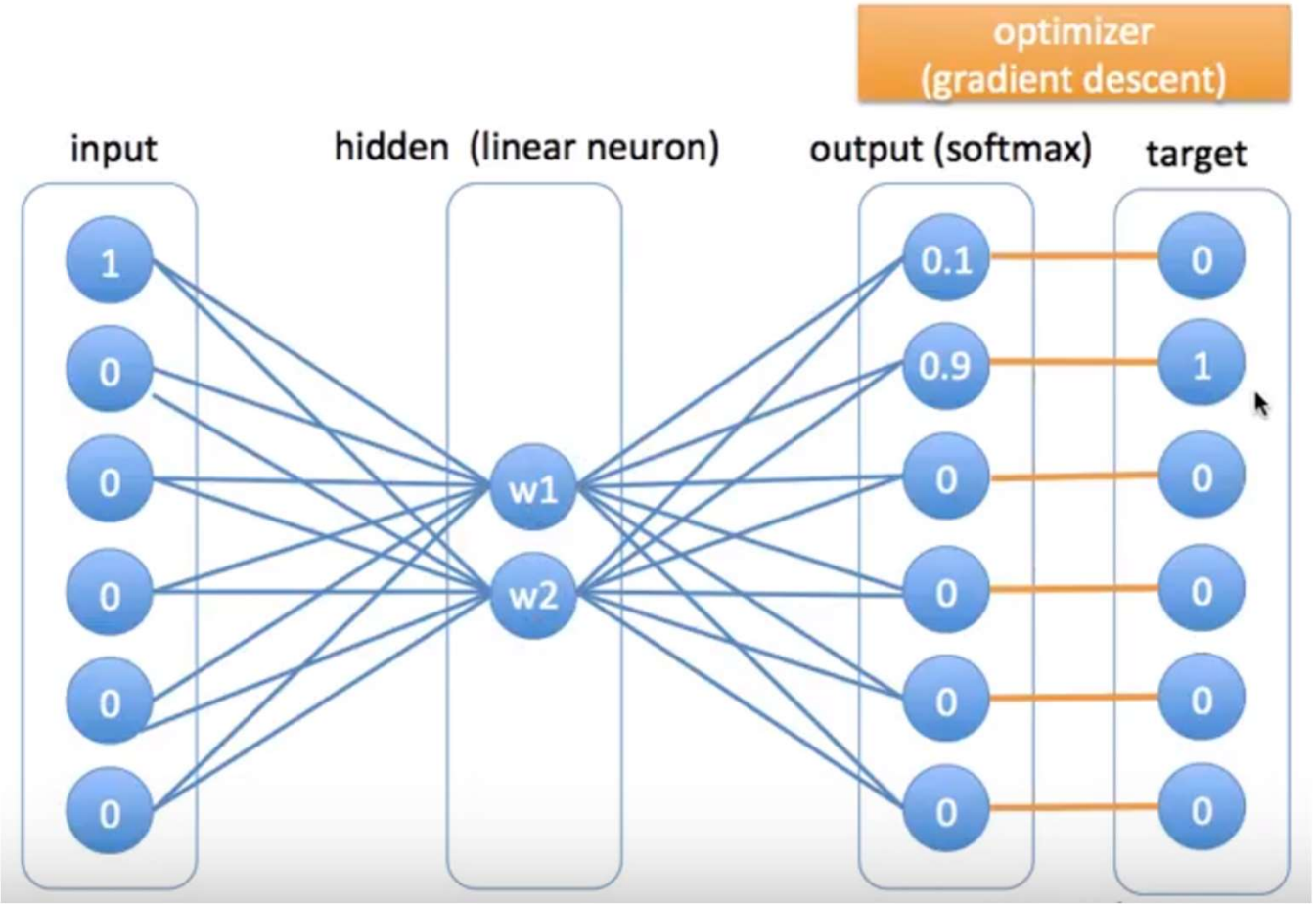


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Word2Vec training network

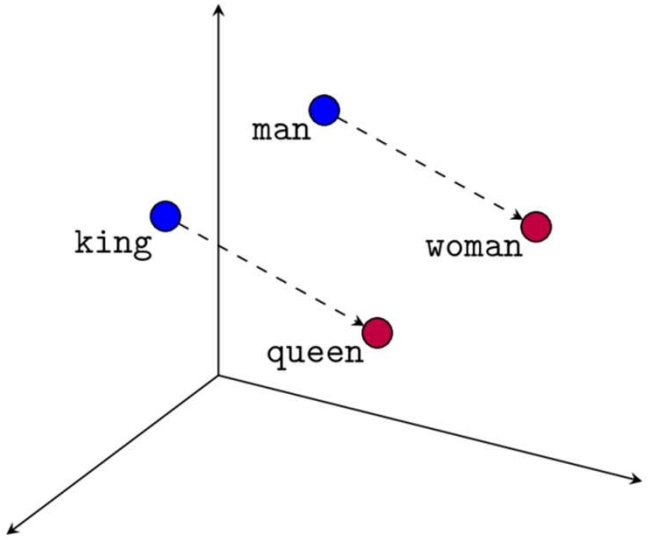
king



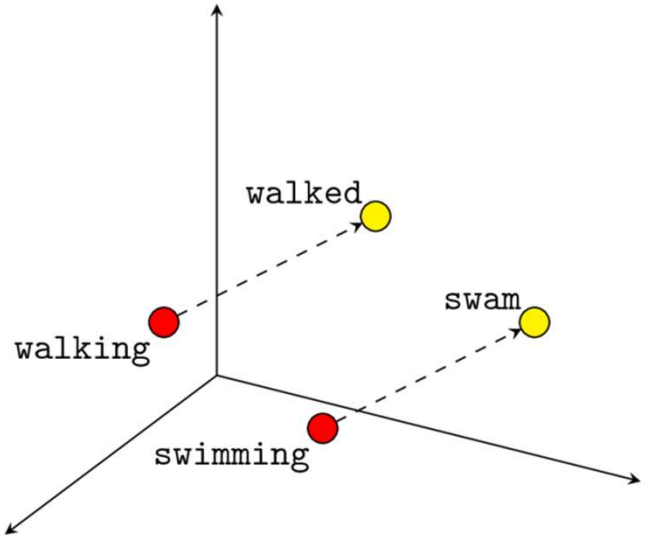
love

Word2Vec Vectorized (Word Embedding)

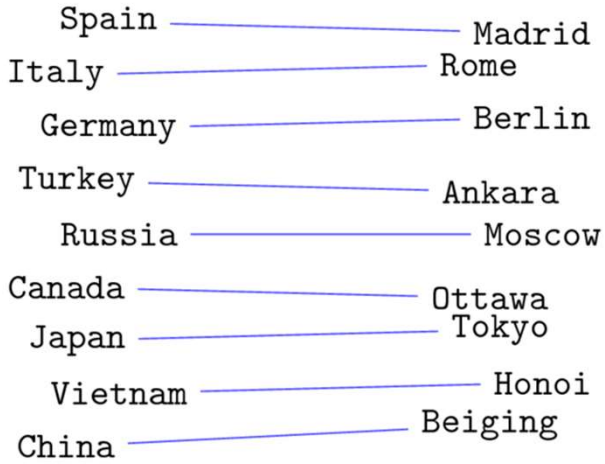
king – queen ≈ man - woman



성별



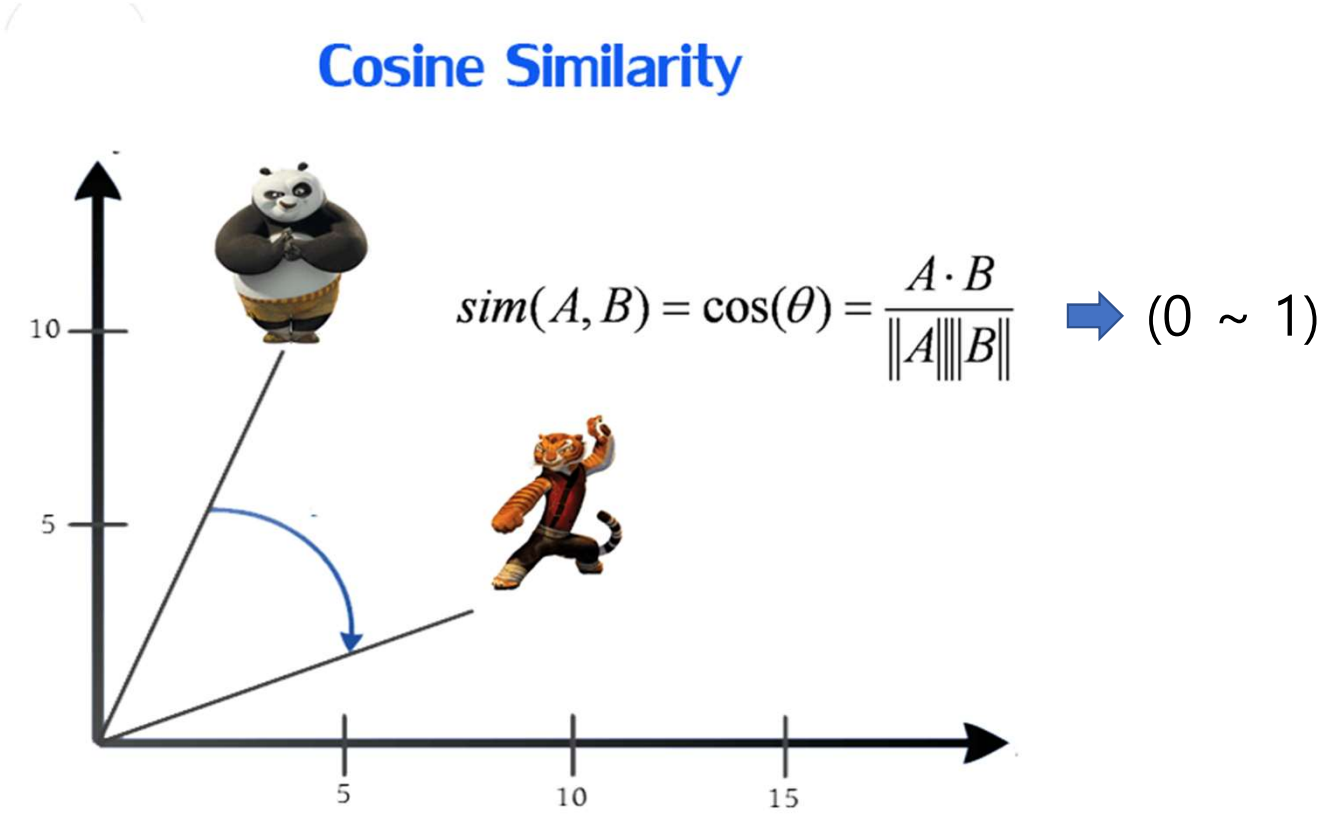
시제



국가-수도

https://ronxin.github.io/wevi/

유사도 측정 (Cosine Similarity)

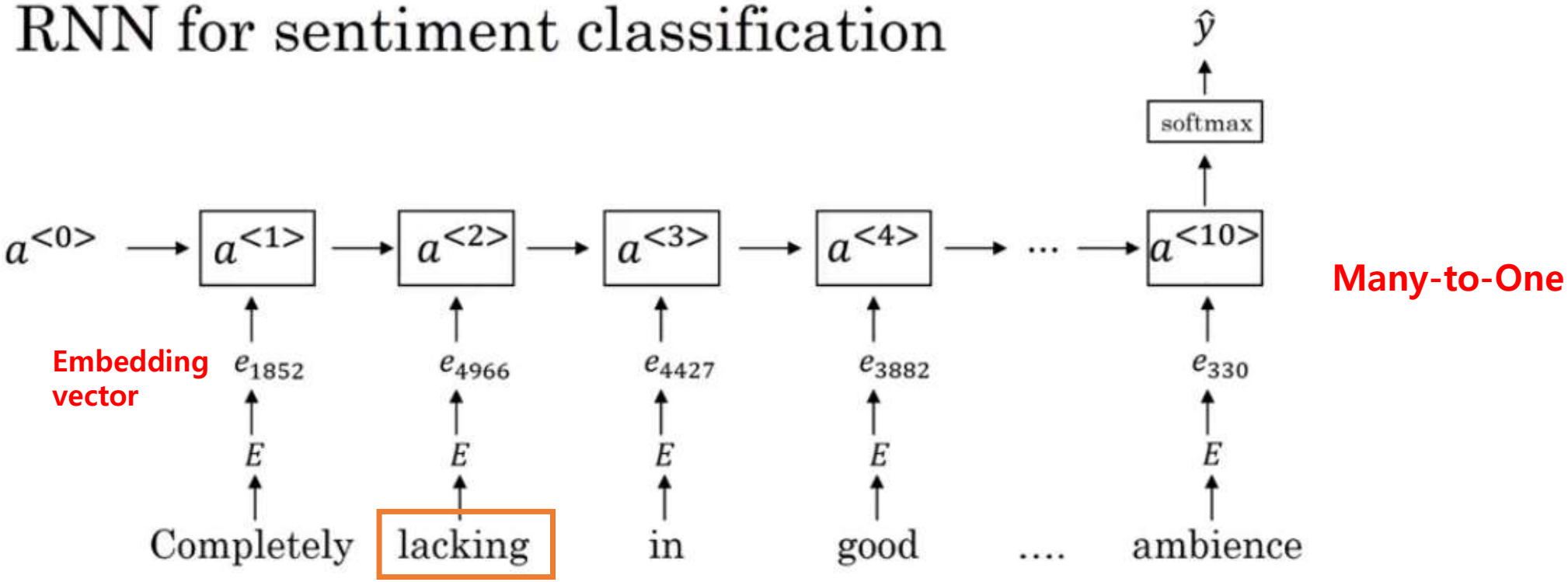


실습 : word2vec 작성

- Sample corpus 를 이용하여 small word2vec model 작성
- word vectorization 의 개념 이해
- Vector 화된 단어 간의 cosine 유사도 시각화

감성분석 (Sentiment Analysis)

RNN 이전 : 단순히 단어의 출현 빈도를 count 하여 positive / negative 분류
➔ 단어의 순서 무시 (Bag of Words)



실습 : 영화 관람평 분류 (sentiment 분석)

- Keras IMDB (Internet Movie Database) 이용
  - 25,000 개 영화관람평
- Movie reviews sentiment classification
  - Label : positive, negative
- Preprocessing 되어 있고 모든 review 는 word indexes (integers) 로 표시
  (인덱스 순서는 빈번히 나타나는 단어 순서. ex. 3 : 3 번째로 빈번히 나타나는 단어)
- LSTM 을 이용한 Many-to-One type 의 RNN 으로 구현

Keras Processing Tips

Early Stopping

- Epoch 이 반복되어도 더 이상 성능 향상이 이루어지지 않을 경우 조기에 training 종료

Callback

- Training 중에 keras 의 behavior 를 변경할 수 있는 customization 기능

실습 : 자동차 연비 예측 Regression

- UCI Machine Learning Data 이용
- 의도적으로 과적합을 만들고, early stopping 기능 test
- Early stopping 전, 후의 loss plot 하여 비교
- Checkpoint 추가 및 model save / reload

과적합 방지 방법

- More Data – Best solution but 항상 가능하지 않음
- Network 의 size 축소
- Weight Regularization (주로 L2 사용)

$$J(w,b) = \frac{1}{m}\,\Sigma\, L(prediction\ -true) + \frac{\lambda}{2m}\Sigma\ \left||w|\right|^2$$

- Dropout
  - Neural Network 에서 가장 일반적으로 많이 사용

실습 : Regularization 을 이용한 IMDB 영화평 성능 개선

- IMDB 이용
- Weight Regularization 과 Dropout 이용
- Small, medium 및 Big model 에 대하여 서로 비교