Deep Learning Contents

1. (Intro) Why Deep Learning is needed ?

2. Deep Neural Network

- 2 Layers DNN
- Prediction & Cost Function
- Gradient Decent
- Backpropagation
- **Exercise #1. (Numpy) 2 Layers DNN**
- Keras
- **Exercise #2. (Keras) 2 Layers DNN**

3. Convolutional Neural Network

- Convolutional Neural Network 3 Layers
- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer
- **Exercise #3. (CNN) MNIST Classification**
- Famous CNN Models
- Residual Network
- **Exercise #4. (ResNet) MNIST Classification**

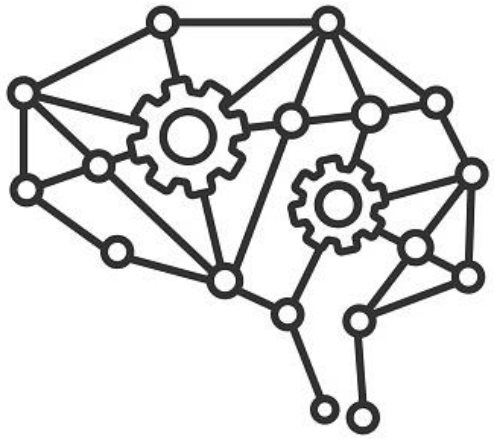
4. Optimization

- Batch Normalization
- Learning Rate Schedule
- Early Stopping
- Ensemble
- **Exercise #5. BN, LRS, Early Stopping**
- Parallel Processing using GPU
- **Exercise #6. Parallel Processing test using GPU**

5. Recurrent Neural Network

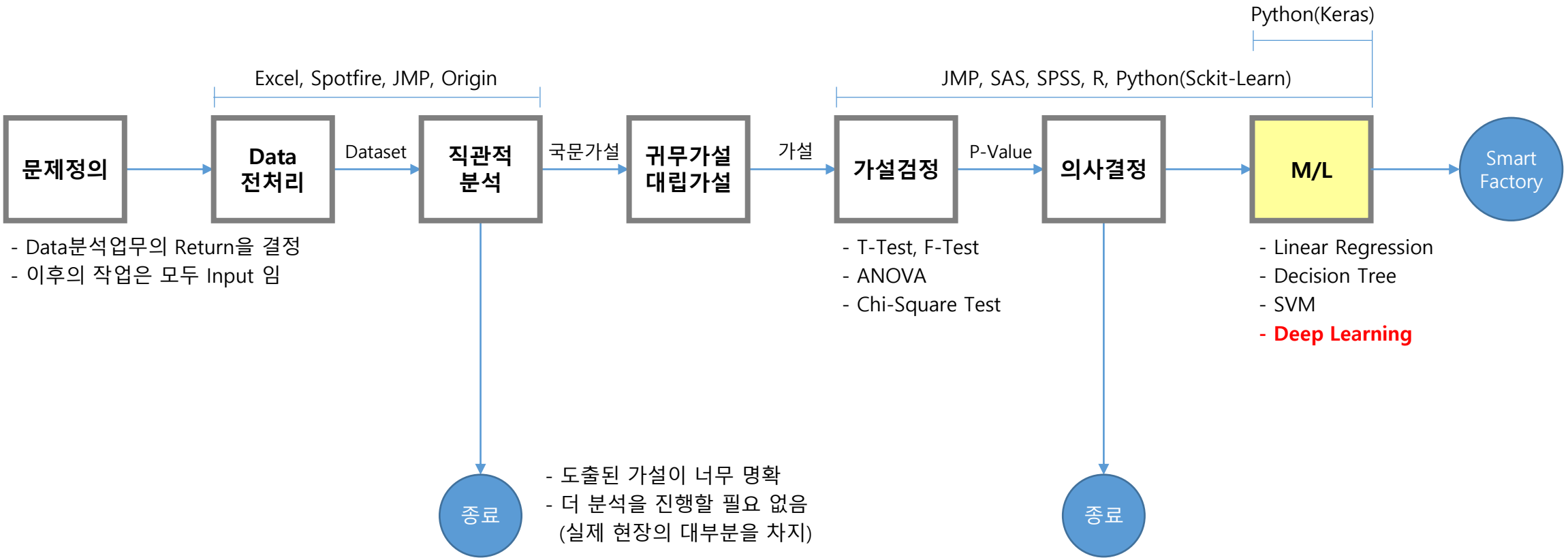
- Sequential Data
- Many-to-Many, Many-to-One, One-to-Many
- **Exercise #7. (Many-to-Many) One-hot Encoding**
- **Exercise #8. (Many-to-One) Token Embedding**
- Vanilla RNN
- LSTM
- **Exercise #9. (LSTM) One-hot Encoding**

Appendix



1. (Intro) Why Deep Learning is needed ?

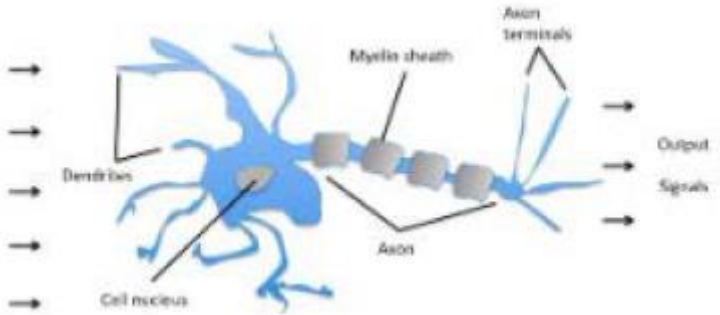
Data Analysis Workflow



1. (Intro) Why Deep Learning is needed ?

Perceptron

- 1958년에 Frank Rosenblatt가 제작
- 생물학적 뇌의 뇌세포(Neuron)을 모방하여 만든 Artificial Neural Network의 기본 단위
- 뇌세포



① 다수의 입력돌기와 1개의 출력 돌기가 존재

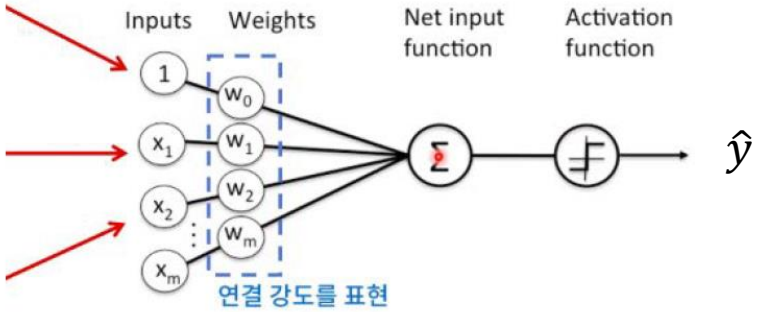
② 연결된 강도와 Soma의 크기가 변함

③ 수집된 신호물질이 Soma에 저장됨

④ 신호물질이 Soma의 Capa를 초과하면 출력

⑤ 뇌세포 860억, Synapse 100조 개

Perceptron



① n차원의 입력 인자와, $\hat{y}$라는 출력 인자가 존재

② Weights와 Bias 등의 Parameter로 처리

③ $x_1 w_1 + x_1 w_{1+} \ldots + x_n w_n = W^T X$

④ $sigmoid(W^T X)$ 으로 Identity 함수를 대체함.

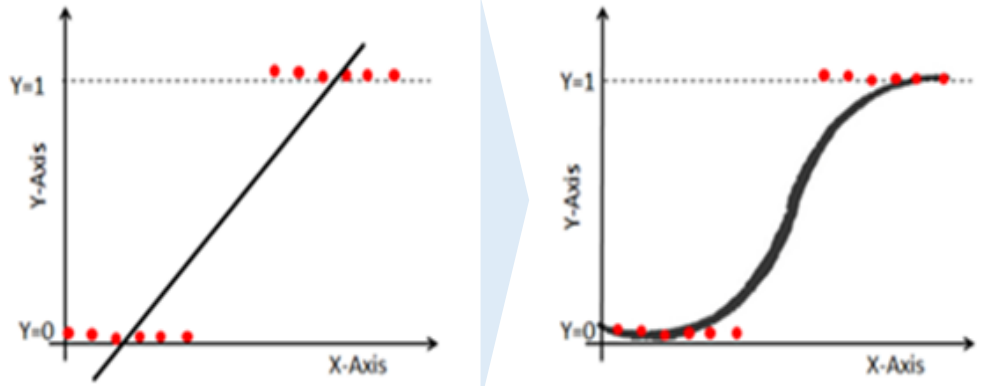
⑤ ResNet의 경우 0.005억 개의 Parameter로 구성

1. (Intro) Why Deep Learning is needed ?

Limits of Linear Regression

- LR은 Parameter의 차수는 1차이며, 그것들의 관계식은 +, -로만 이루어져 있음.
- Close한 방법이 없음 → Data에 따라 적절한 수학적 Trick을 사용

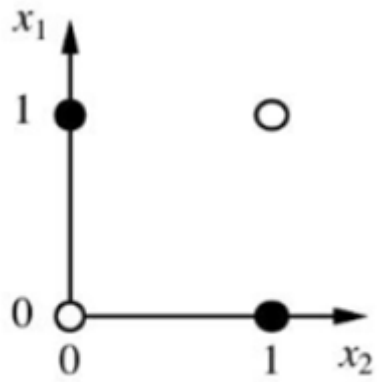
Binary Classification



$$y = b_0 + b_1 x$$

$$y = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$

XOR Gate 문제



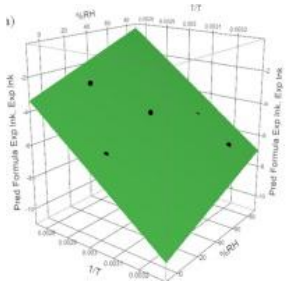
x_1 **xor** x_2

?

1. (Intro) Why Deep Learning is needed ?

- High dimension Data Problem (음성, 영상 등 …)
- Bigdata 시대가 도래한 이후, 고차원 Data 비중이 급격히 증가

High Dim Linear Regression



Stepwise Fit for Y

Stepwise Regression Control

SSE	DFE	RMSE	RSquare	RSquare Adj	Cp	p	AICc	BIC
57.904483	10	2.406335	0.9787	0.9744	2.6782416	3	69.31239	66.57219

Current Estimates

Lock	Entered	Parameter	Estimate	nDF	SS	"F Ratio"	"Prob>F"
☑	☑	Intercept	52.5773489	1	0	0.000	1
☐	☑	X1	1.46830574	1	848.4319	146.523	2.69e-7
☐	☑	X2	0.66225049	1	1207.782	208.582	5.03e-8
☐	☐	X3	0	1	9.793869	1.832	0.20889
☐	☐	X4	0	1	9.931754	1.863	0.2054

Model	Number	RSquare	RMSE	AICc	BIC	Cp	
X4	1	0.6745	8.9639	100.411	99.4389	138.7308	●
X2	1	0.6663	9.0771	100.737	99.7652	142.4864	○
X1,X2	2	0.9787	2.4063	69.3124	66.5722	2.6782	●
X1,X4	2	0.9725	2.7343	72.6341	69.8939	5.4959	○
X1,X2,X4	3	0.9823	2.3087	72.4377	66.6910	3.0182	●
X1,X2,X3	3	0.9823	2.3121	72.4750	66.7283	3.0413	○
X1,X2,X3,X4	4	0.9824	2.4460	79.8367	69.2264	5.0000	●

차원이 늘어나도 Error가 줄지 않음!

Flattened Image

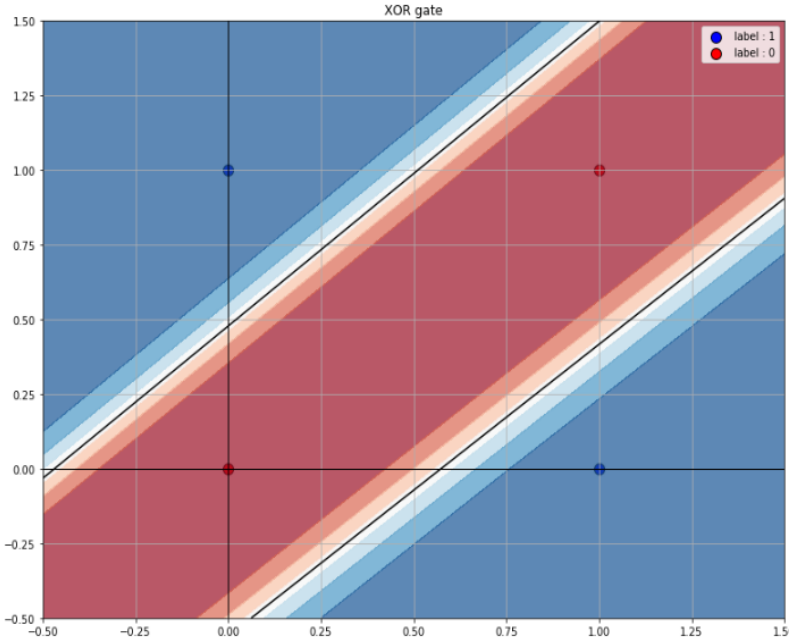
Img#	0,0,R	0,1,G	…	32,32,G	32,32,B
1	12	12		12	45
2	54	54		54	14
3	14	14		14	14
4	11	11		11	11
5	0	0		0	66
6	123	123		123	95
7	255	255	…	255	67
8	254	254		254	85
9	11	11		11	125
10	8	8		8	201
11	3	3		3	222
12	2	2		2	187

32 x 32 x RGB → 3,072 차원

1. (Intro) Why Deep Learning is needed ?

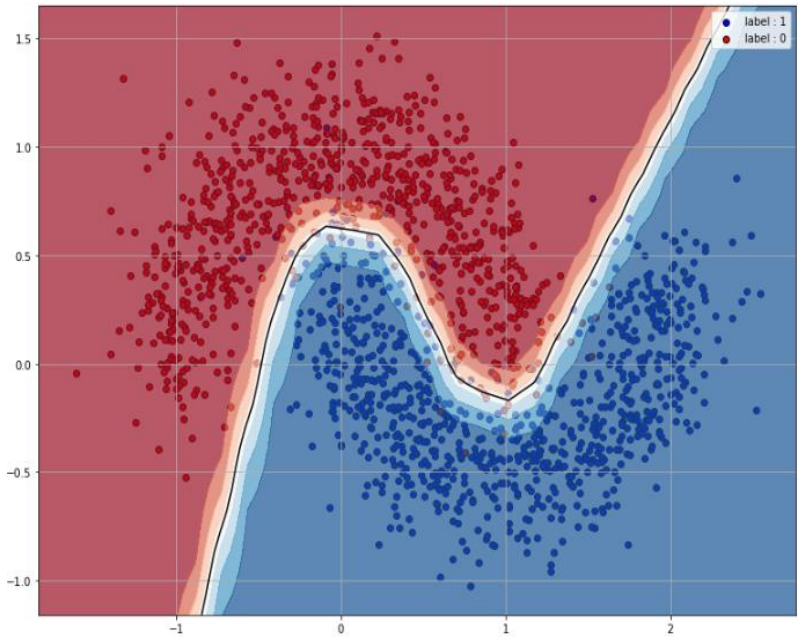
- Deep Artificial Neural Network Machine Learning → (줄여서) Deep Learning
- Deep Learning can do all of THESE !

XOR Gate Problem



→ 기하학적으로 선형으로 풀 수 없는 Data

Moon Shape Scatter



→ Parameter간의 관계를 알 수 없는 Data

Image Data

→ 고차원 Data 처리

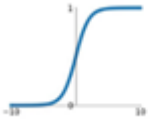
2 Layers DNN

- Hidden Layer가 1개 있는 Basical DNN
- DNN 표기법
  - ① 선 → Weights, 원 → nodes (Bias는 따로 표현하지 않음)
  - ② 아래 첨자 → Dimensions, 윗 첨자 = Index
  - ③ 아래 첨자 없음 → Vector (e.g. $a_1^{[1]} + a_2^{[1]} + a_3^{[1]} + a_4^{[1]} = a^{[1]}$)

- Forward (Probability 계산과정)
  - ① Input → Hidden : $a^{[1]} = max(0, W^{[1]}x + b^{[1]})$
  - ② Hidden → Output : $\hat{y} = sigmoid(W^{[2]}a^{[1]} + b^{[2]})$
  - ③ Input → Output : $\hat{y} = sigmoid(W^{[2]}max(0, W^{[1]}x + b^{[1]}) + b^{[2]})$

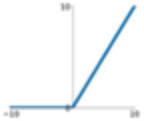
Sigmoid

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

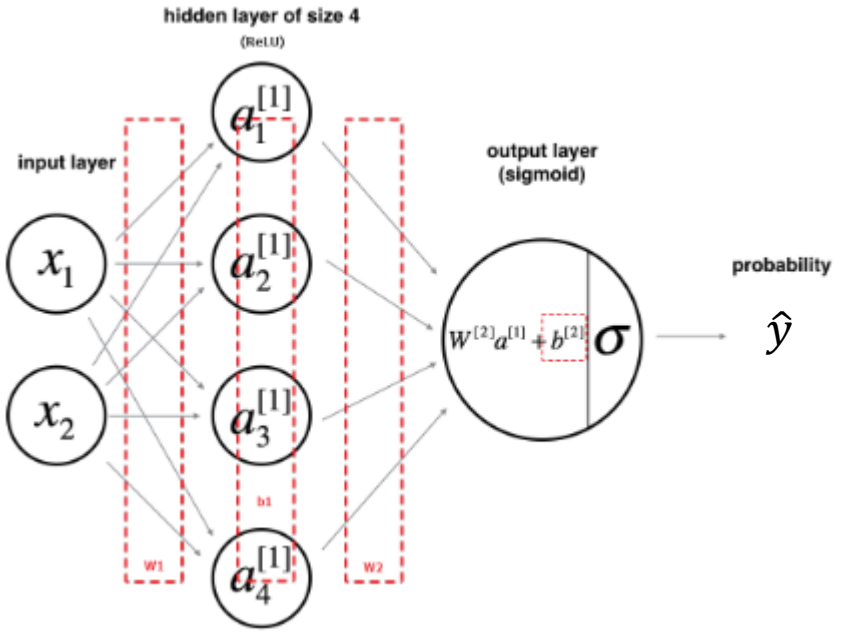


ReLU

$\max(0, x)$



2 Layers DNN



$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

2. Deep Neural Network

- Why ReLU(Rectified Linear Unit) ? Because Vanishing Gradient (★) !

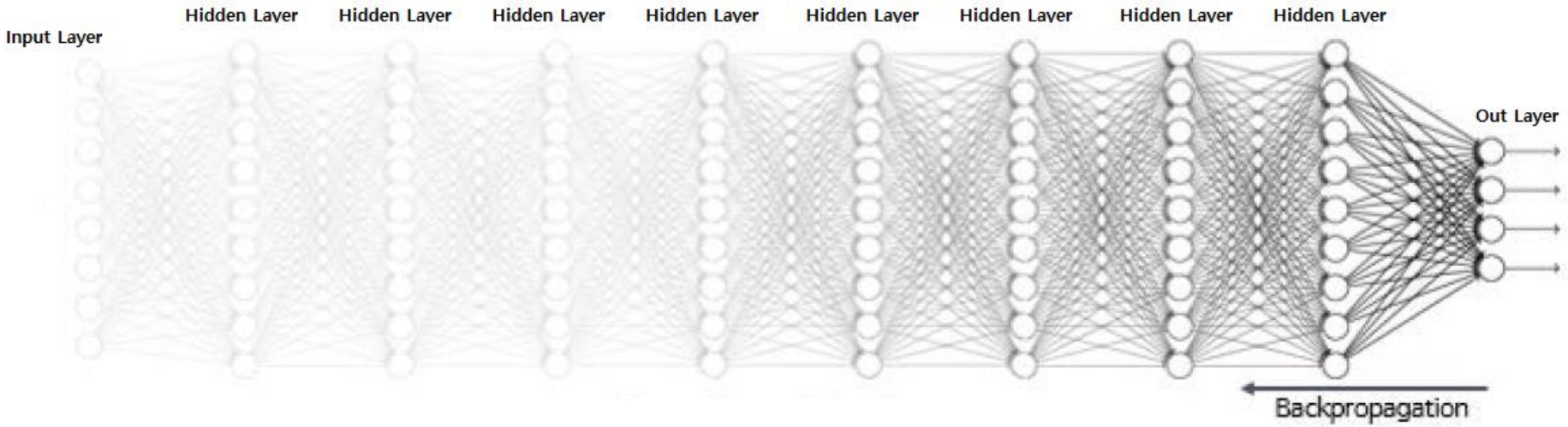
  - Activation Function is sigmoid ? 학습(Backpropagation)을 수행 시킬 때, input값들이 layer를 거치면서 작아짐

    ($0.0 \leq sigmoid(x) \leq 1.0$)

  - 작아진 값들은 다음 layer의 input으로 입력되고 이러한 과정이 반복되면 해당 값들이 결국 0.0에 수렴.

  - 결과적으로, 0에 수렴된 값이 다음 layer에 영향을 미치지 않게 됨

    (학습을 아무리 수행해도 Cost가 떨어지지 않게 됨)



Prediction & Cost Function

- Cost Function이란?
  - 예측값과 Label 값의 차이를 정량화 하는 함수
  - 연속형 → MSE, MAE, MAPE, KLD, Cosine Similarity
  - 범주형 → BCE, Sparse categorical CE, Hinge

- 학습이란?

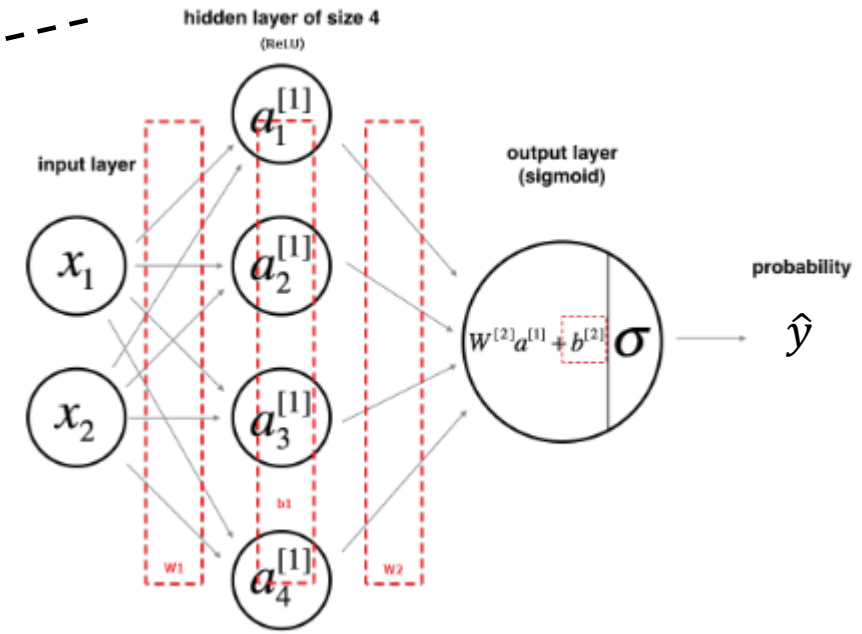
x_1	x_2	y	$\hat{y}$
23	4	0	0
12	14	0	0
45	14	1	1
52	32	0	0
1	14	0	0
2	11	?	
4	5		

Learning

→ $J(w)$를 최소화하는 $W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}$를 찾는 것

→ $J(w) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$

2 Layers DNN



$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

Gradient Decent

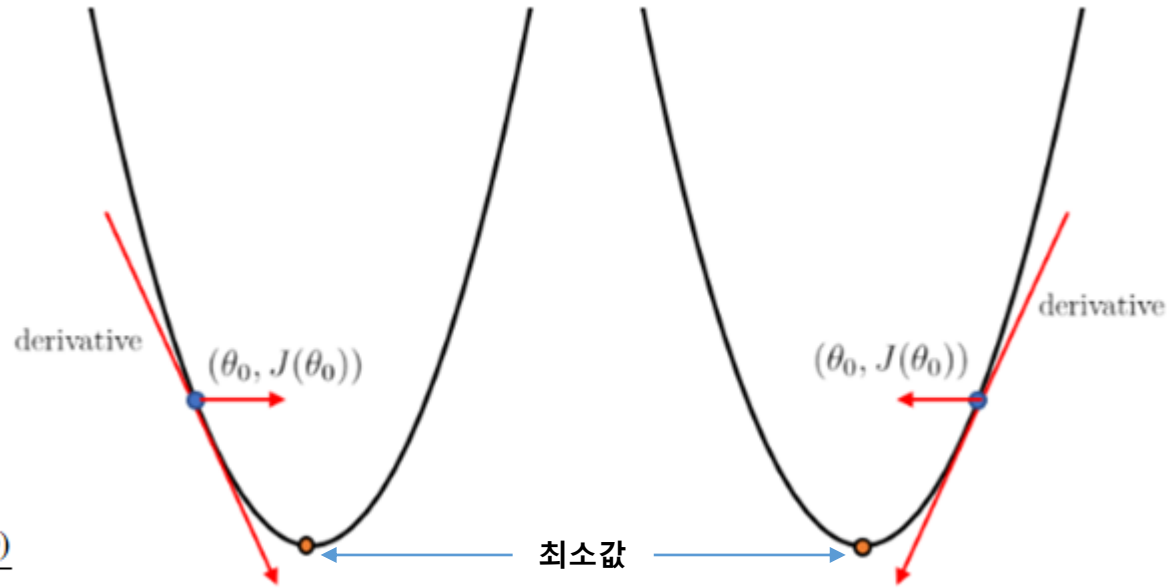
- Is Convex or not ?
  - Convex한 방정식은 미분하여 0이 되는 지점이 최소값
  - BCE는 Convex하지만 Close Form은 존재하지 않음 → Gradient Decent !
- Steps of Gradient Decent
  ① 임의의 지점에서 미분값을 구함
  ② 미분 값이 음(-), 더 큰 데에서 미분
  ③ 미분 값이 양(+), 더 작은 데에서 미분

$$\theta_n = \theta_{n-1} - \alpha \frac{\partial J(w)}{\partial \theta_{n-1}}$$

- 4개 Vector Parameters를 어떻게?

$$W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]} \longrightarrow \frac{\partial J(w)}{\partial W^{[1]}}, \frac{\partial J(w)}{\partial b^{[1]}}, \frac{\partial J(w)}{\partial W^{[2]}}, \frac{\partial J(w)}{\partial b^{[2]}}$$

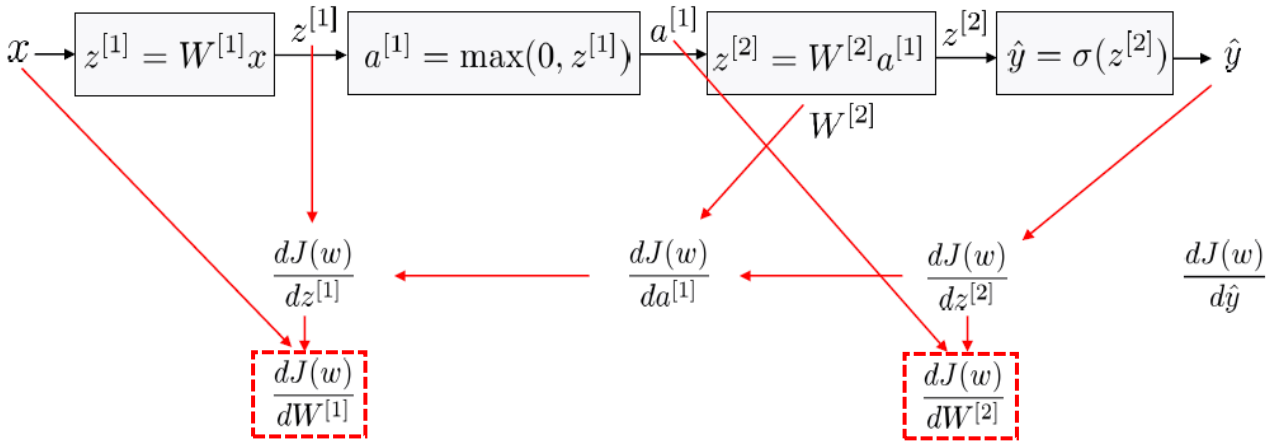
GD



Backpropagation

- 역 전파 알고리즘
  - 1982 Paul Werbos, 1986 Jeffry Hinton이 재발견 함

- 출력층의 결과 오차를 입력층까지 거슬러 전파하면서 계산
  - Forward 과정에서 계산한 값들을 Cash에 잘 저장해두면 $\left(\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}\right)$ 를 이용하여 편미분을 쉽게 할 수 있다.



- Vector Parameters에 대한 편미분 과정 (Backpropagation Proof)

$$\min_{w=(W^{[1]},W^{[2]})} \underbrace{\sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)})}_{=:J(w)}$$

$$\frac{dJ(w)}{d\hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{dJ(w)}{dz^{[2]}} = \frac{dJ(w)}{d\hat{y}} \frac{d\hat{y}}{dz^{[2]}} \overset{(a)}{=} \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \hat{y}(1-\hat{y}) = \hat{y} - y$$

$$\frac{d\sigma(z)}{dz} = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{0-(-e^{-z})}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} = \sigma(z)(1-\sigma(z)).$$

$$\frac{dJ(w)}{dW^{[2]}} = \frac{dJ(w)}{dz^{[2]}} \frac{dz^{[2]}}{dW^{[2]}} \overset{(a)}{=} \frac{dJ(w)}{dz^{[2]}} a^{[1]T}$$

$$\frac{dJ(w)}{da^{[1]}} = \frac{dJ(w)}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \overset{(a)}{=} W^{[2]T} \frac{dJ(w)}{dz^{[2]}}$$

$$\frac{dJ(w)}{dz^{[1]}} = \frac{dJ(w)}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}} \overset{(a)}{=} \frac{dJ(w)}{da^{[1]}} .* \mathbf{1}\{z^{[1]} \geq 0\}$$

$$\frac{dJ(w)}{dW^{[1]}} = \frac{dJ(w)}{dz^{[1]}} \frac{dz^{[1]}}{dW^{[1]}} \overset{(a)}{=} \frac{dJ(w)}{dz^{[1]}} x^T$$

$$\frac{dJ(w)}{db^{[2]}} = \frac{dJ(w)}{dZ^{[2]}} \frac{dZ^{[2]}}{db^{[2]}} = \begin{bmatrix} \left[\frac{dJ(w)}{dZ^{[2]}}\right]_1 & \cdots & \left[\frac{dJ(w)}{dZ^{[2]}}\right]_m \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \sum_{i=1}^{m} \left[\frac{dJ(w)}{dZ^{[2]}}\right]_i$$

$$\frac{dJ(w)}{db^{[1]}} = \frac{dJ(w)}{dz^{[1]}}$$

간단식 !

① $\frac{\partial J(w)}{\partial z^{[2]}} = \frac{1}{m}(\hat{Y} - Y)$

② $\frac{\partial J(w)}{\partial W^{[2]}} = \frac{\partial J(w)}{\partial z^{[2]}} a^{[1]T}$

③ $\frac{\partial J(w)}{\partial b^{[2]}} = \sum_{i=1}^{m} \left[\frac{\partial J(w)}{\partial z^{[2]}}\right]_i$

④ $\frac{\partial J(w)}{\partial a^{[1]}} = W^{[2]T} \frac{\partial J(w)}{\partial z^{[2]}}$

⑤ $\frac{\partial J(w)}{\partial z^{[1]}} = \frac{\partial J(w)}{\partial a^{[1]}} \cdot *\mathbf{1}\{z^{[1]} \geq 0\}$

⑥ $\frac{\partial J(w)}{\partial W^{[1]}} = \frac{\partial J(w)}{\partial z^{[1]}} x^T$

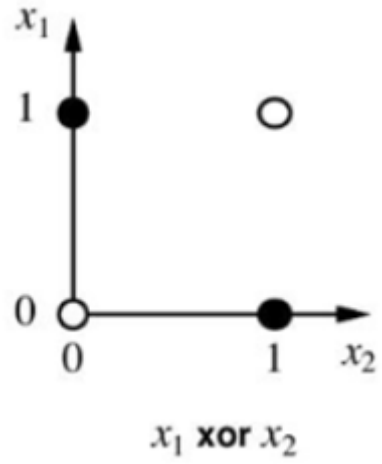
⑦ $\frac{\partial J(w)}{\partial b^{[1]}} = \sum_{i=1}^{m} \left[\frac{\partial J(w)}{\partial z^{[1]}}\right]_i$

Exercise #1. (Numpy) 2 Layers DNN

1) Numpy를 이용하여 주어진 2 Layers DNN을 만들어라.

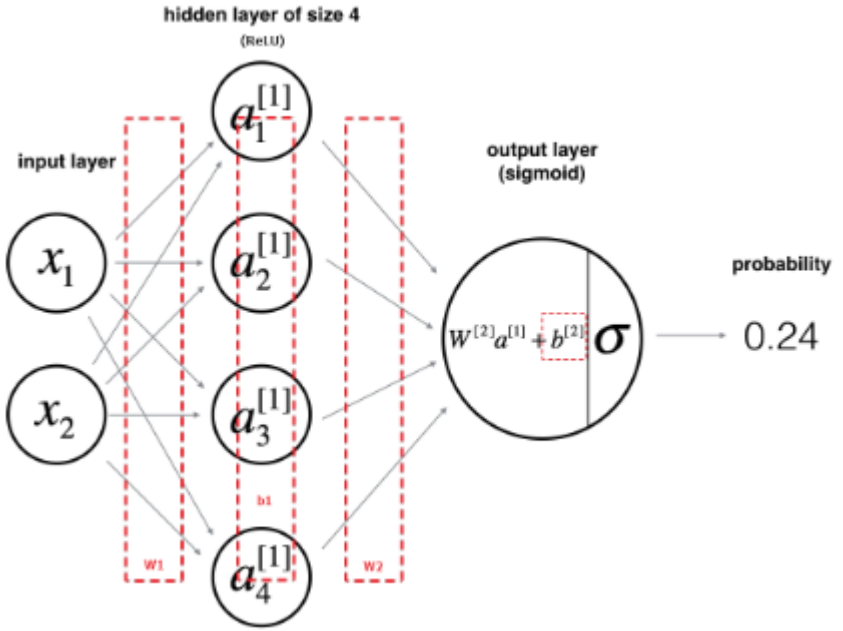
2) 만들어진 Network을 학습시켜 XOR Gate 문제를 해결하라.

XOR Gate 문제



?

2 Layers DNN

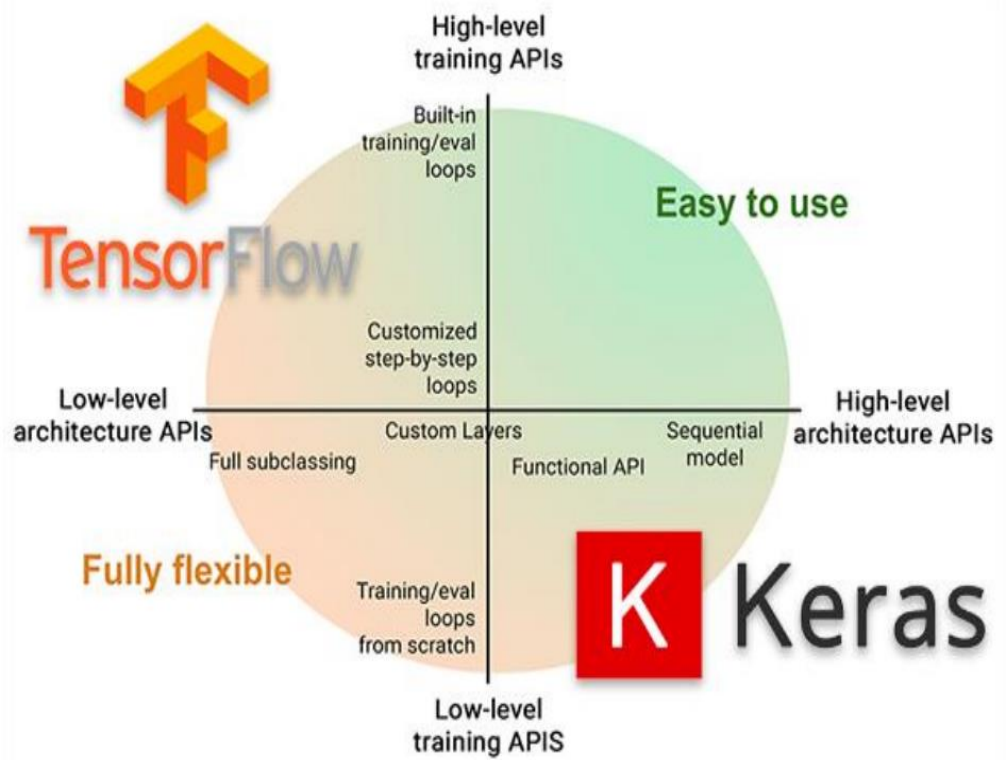


Keras

- (Short) Intro
  - Hig-level Neural Networks API로 문법이 간결함
  - Tensorflow, Theano 엔진 지원 (keras == tf.keras)

- API – Layer to Layers
  - 완전연결층 → tf.keras.layers.Dense(unints, kernel_initial …)
  - 시퀀스계층 → tf.keras.layers.RNN( ), LSTM( ), GRU( )
  - 이미지분류 → tf.keras.layers.Conv2D( )

- API – Sequential/Model
  - model = tf.keras.Sequential(layers)
  - model.add(layer)
  - model.compile(optimization, loss, metrics …)
  - model.fit(X, Y, batch_size, epochs, verbose, callbacks …)
  - model.save(filepath …)
  - Model = tf.moels.load_model(filepath)

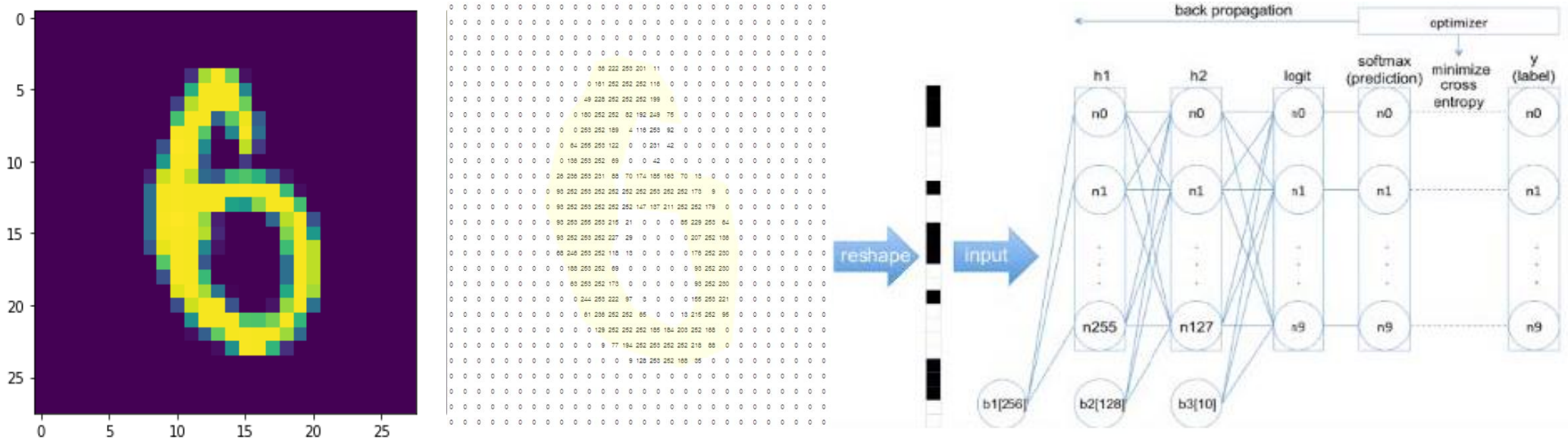


Keras for Image Classification Problem

- Image → Neural Net의 입력으로 변환 (Flattening & Floating)

  - Image Size = 25 x 25 = 625

    ☞ "이미지 분류 문제는 수백 또는 수천 차원의 회귀분석 방정식이다!"



Alpha of a Dot = [X, Y]
= 0~255
= 0~ 1.0

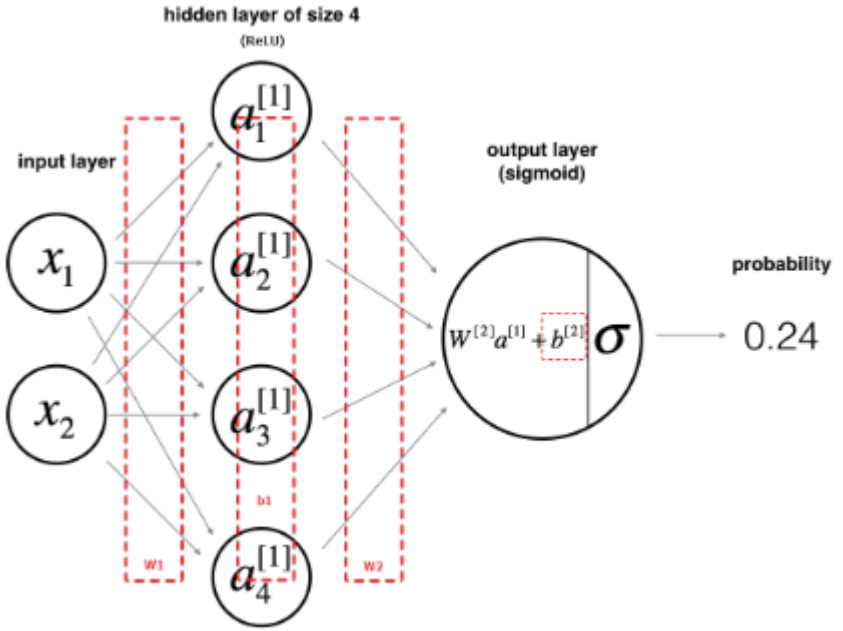
Exercise #2. (Keras) 2 Layers DNN (★)

1) Keras를 이용하여 주어진 2L DNN을 만들어라.

2) 만들어진 Network을 학습시켜 주어진 Image 데이터에서 고양이가 아닌 것을 선별하라.

고양이 Data

2 Layers DNN

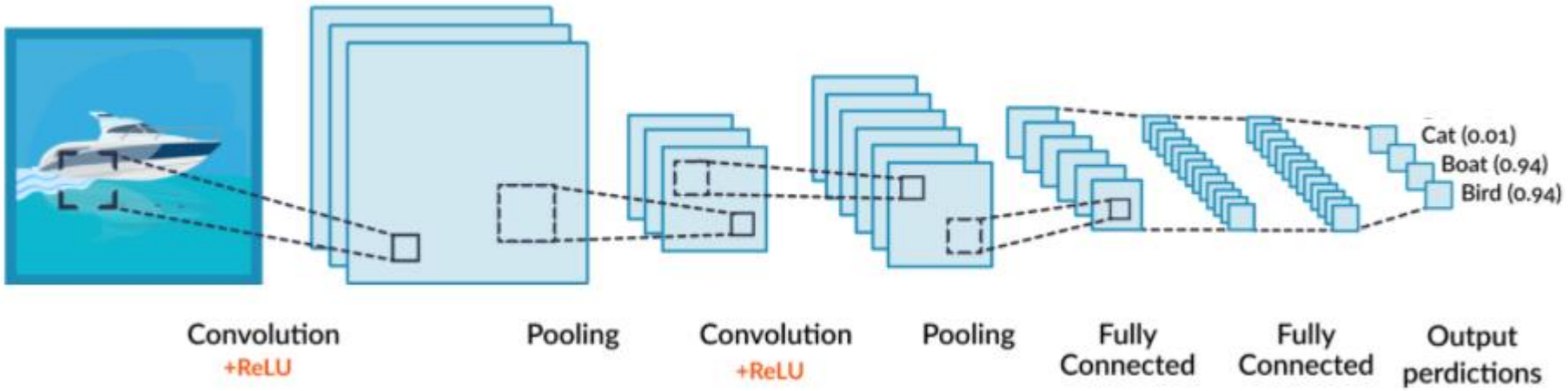


Convolutional neural network 3 Layers

- Convolution / Polling Layer
  - Feature를 추출, 의미 없는 특징은 Zero화 (0으로 제거), 특징 개수 축소, 중요한 특징만 유지

- Fully-Connected Layer
  - 일반적인 Neural Network로써 수집된 최종 Feature들로만 학습

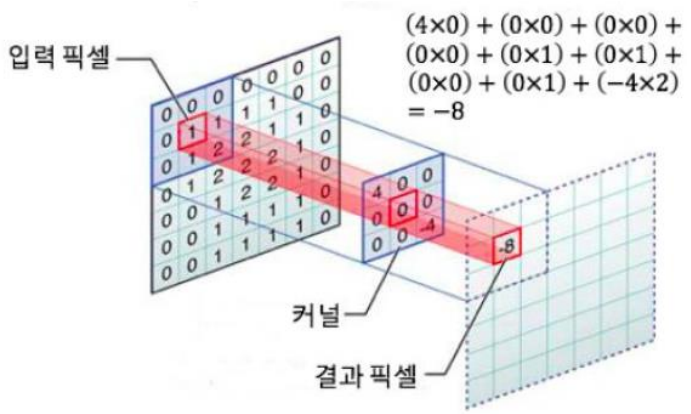
CNN Process



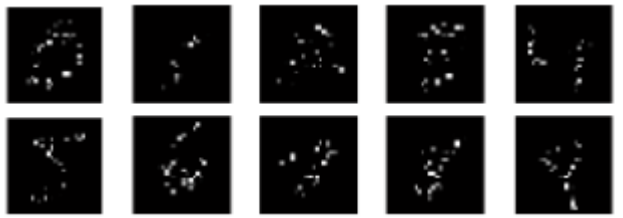
Convolutional Layer

- Convolution 연산

  - Kernel의 각 요소와 대응하는 입력 픽셀 값을 곱해서 모두 합함 (차원이 축소됨)



Original MNIST image



Kernel Filtered MNIST image (Feature Map)

  - CNN의 목표는 Kernel을 학습시키는 것 (Kernel 內 하나의 Dot = 하나의 Weight)

$P=W+B$

$W=C\times K^2\times N$

$B=N$

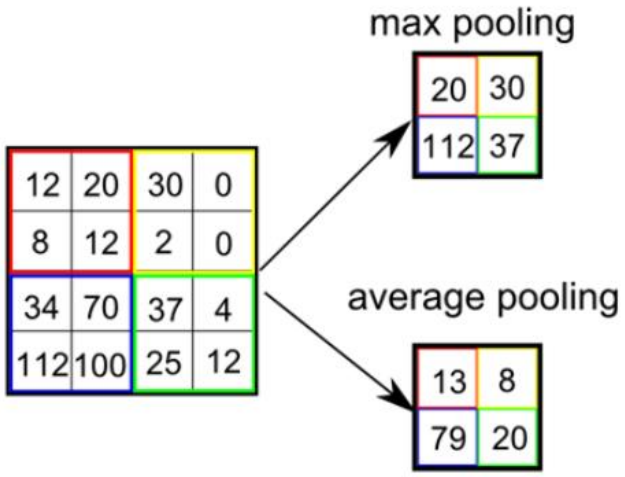
W : 가중치(Weight) 갯수

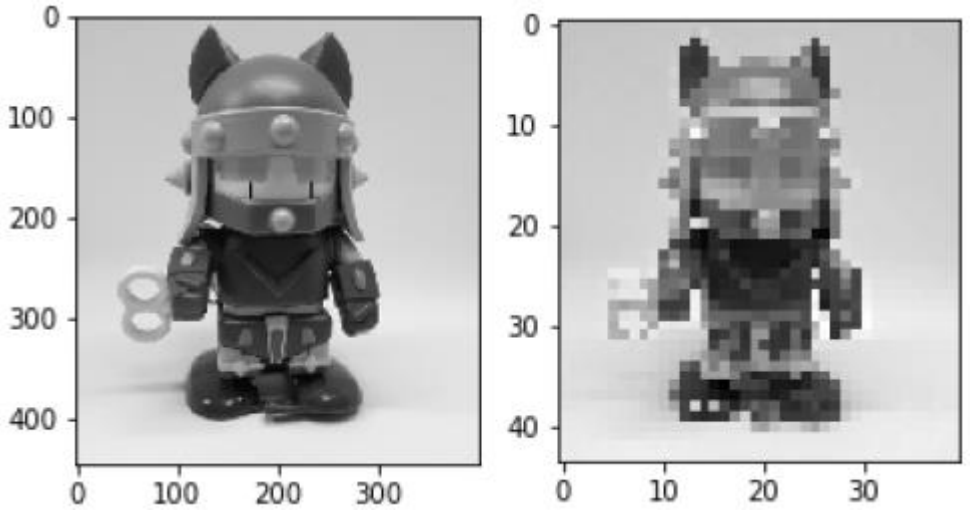
C : 입력 이미지 채널 수 K : 커널 사이즈 N : 커널(출력) 갯수

B : 편향(bias) 갯수

Pooling Layer

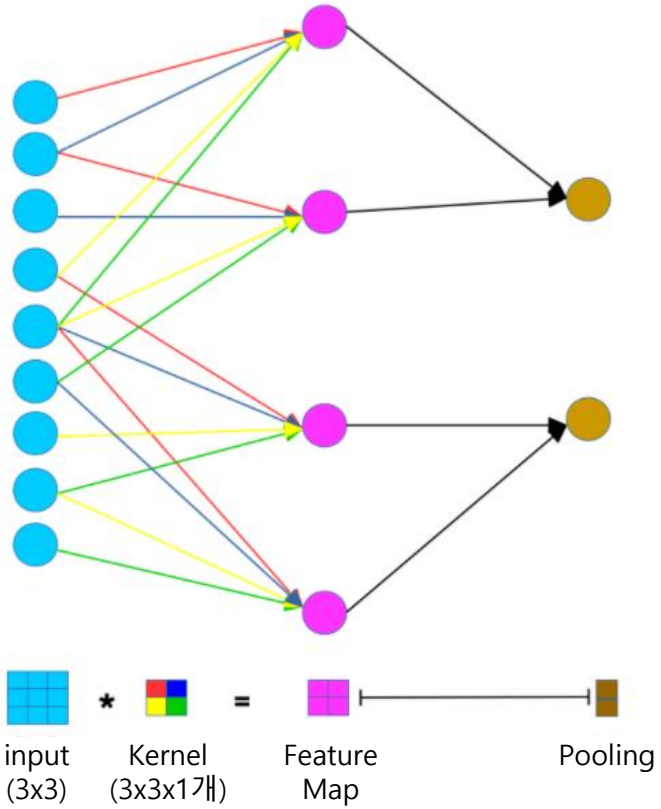
- Feature Map조차도 다 볼 필요가 없음 → Down Sampling
  - N x N 크기의 Window안에 값 하나만을 선택
    - ① Mean Pooling
    - ② Max Pooling (일반적으로 가장 성능이 우수한 것으로 알려져 있음)
    - ③ Average Pooling



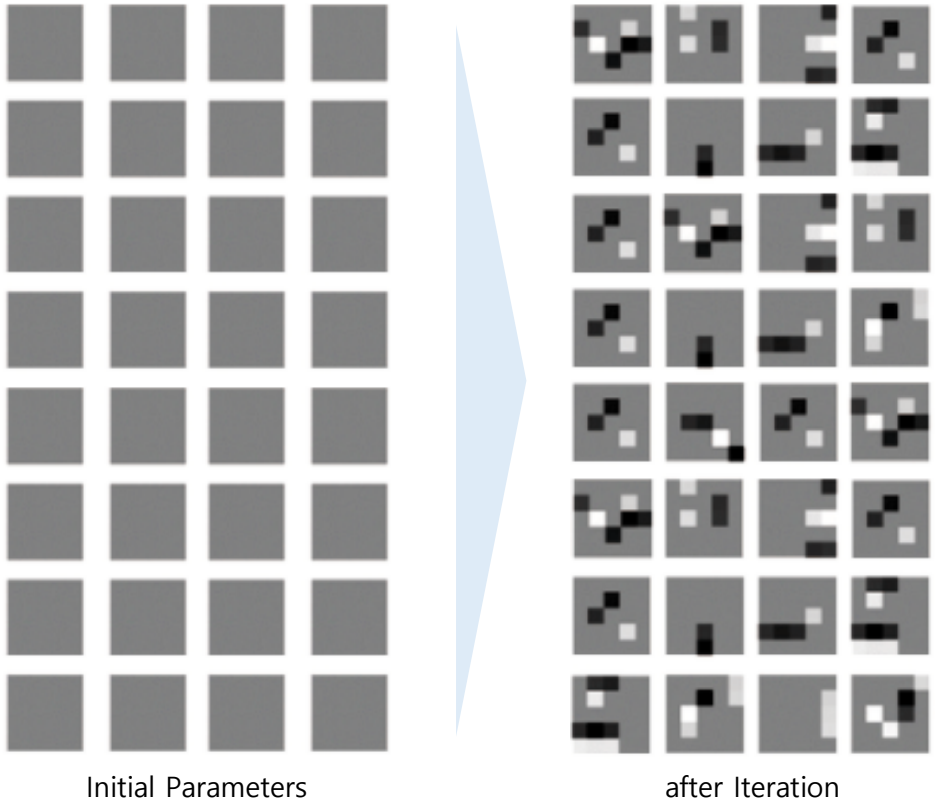


3. Convolutional Neural Network

- Forward: Convolution & Pooling
  - Kernel = 2 x 2 x 1개

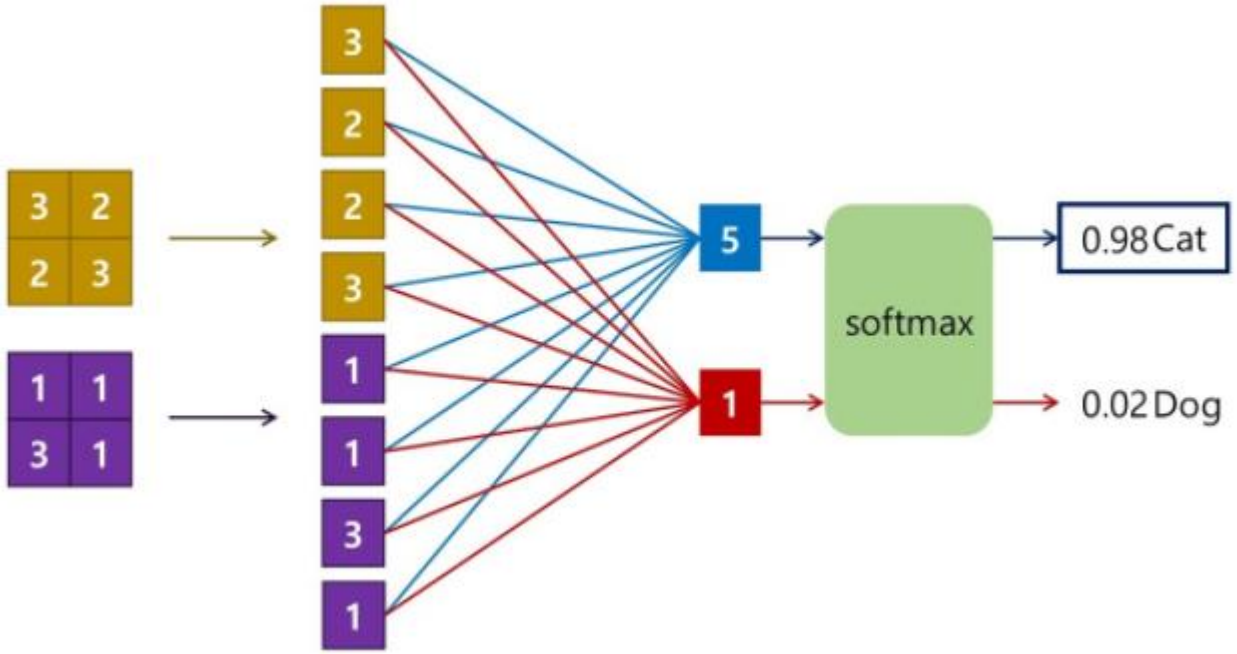


- Backward: Update Kernels
  - Kernel = 2 x 2 x 1개



Fully-Connected Layer

- 마지막에 flatten으로 일렬로 쭉 펴서 dense layer로 처리.
- Convolution Layer에서 뽑은 특징으로 Image 분류



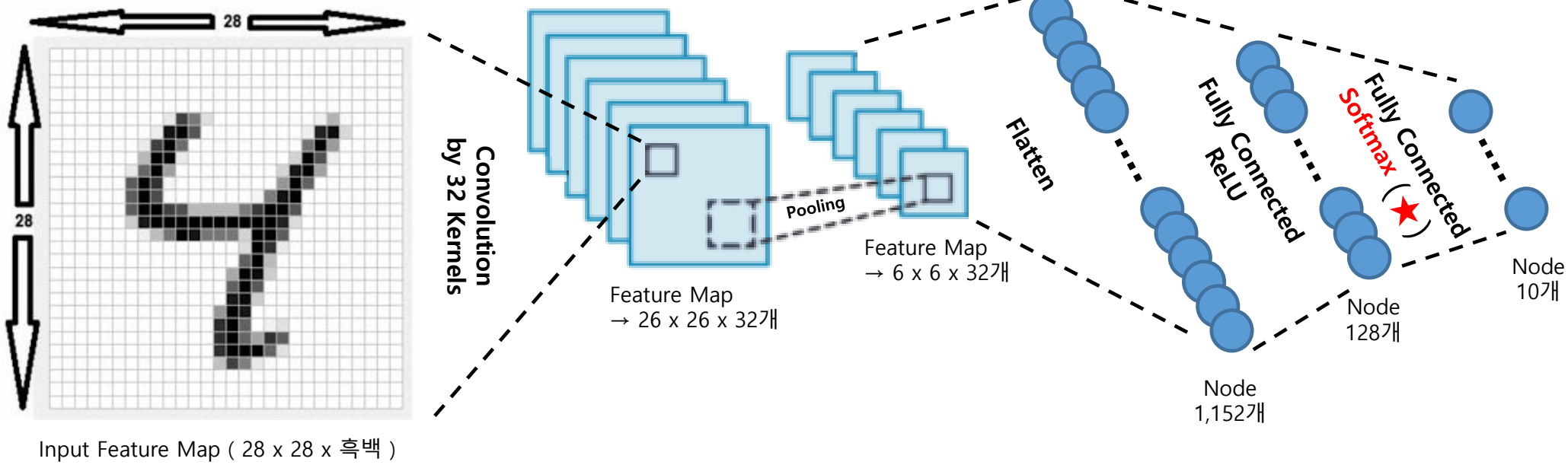
이항분류 → Sigmoid

다항분류 → Softmax (★)

Exercise #3. (CNN) MNIST Classification

1) Keras를 이용하여 주어진 CNN을 만들어라.

2) 만들어진 Network을 학습시켜 MNIST 데이터를 분류하라.

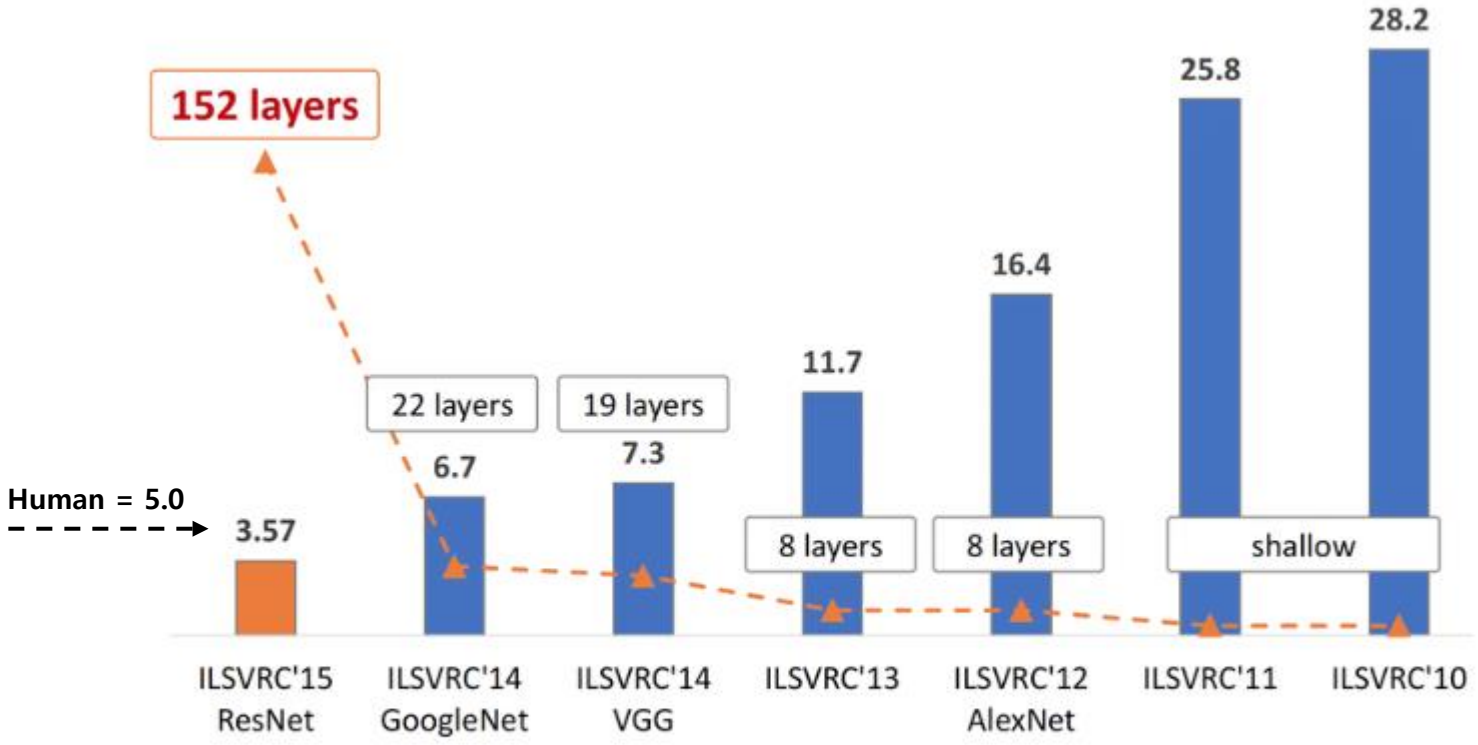


다항분류

0 → 14%
1 → 20%
2 → 11%
3 → 66%
4 → 97%
5 → 41%
6 → 12%
7 → 45%
8 → 14%
9 → 32%

Famous CNN Models

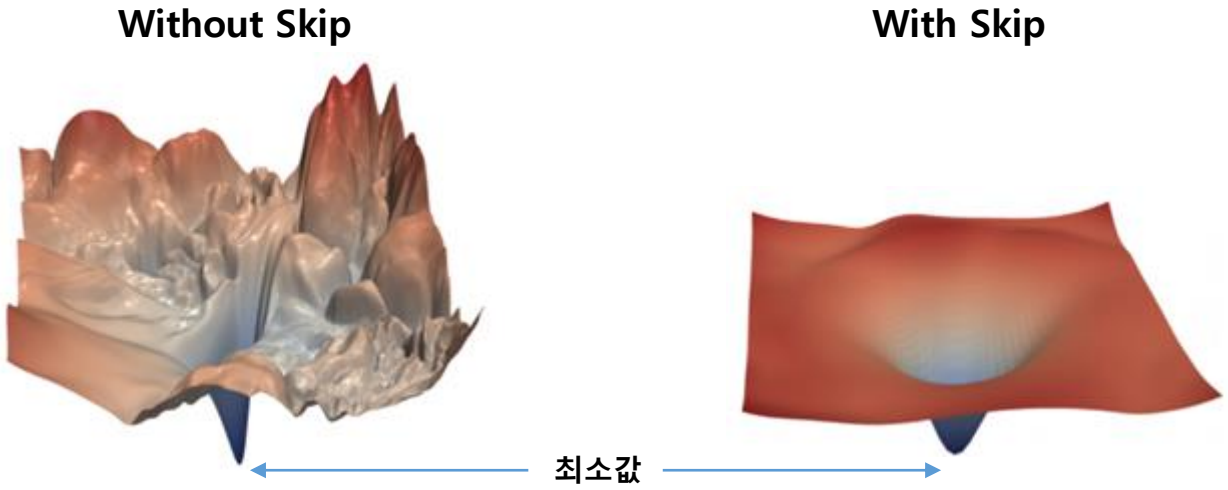
- LeNet
- AlexNet, 2012
- GoogleNet, 2014
- VGGNet, 2014
- Inception-v3, 2015
- ResNet, 2015
- Xception, 2016
- DenseNet, 2016
- SqueezNet, 2016
- MobileNet, 2017

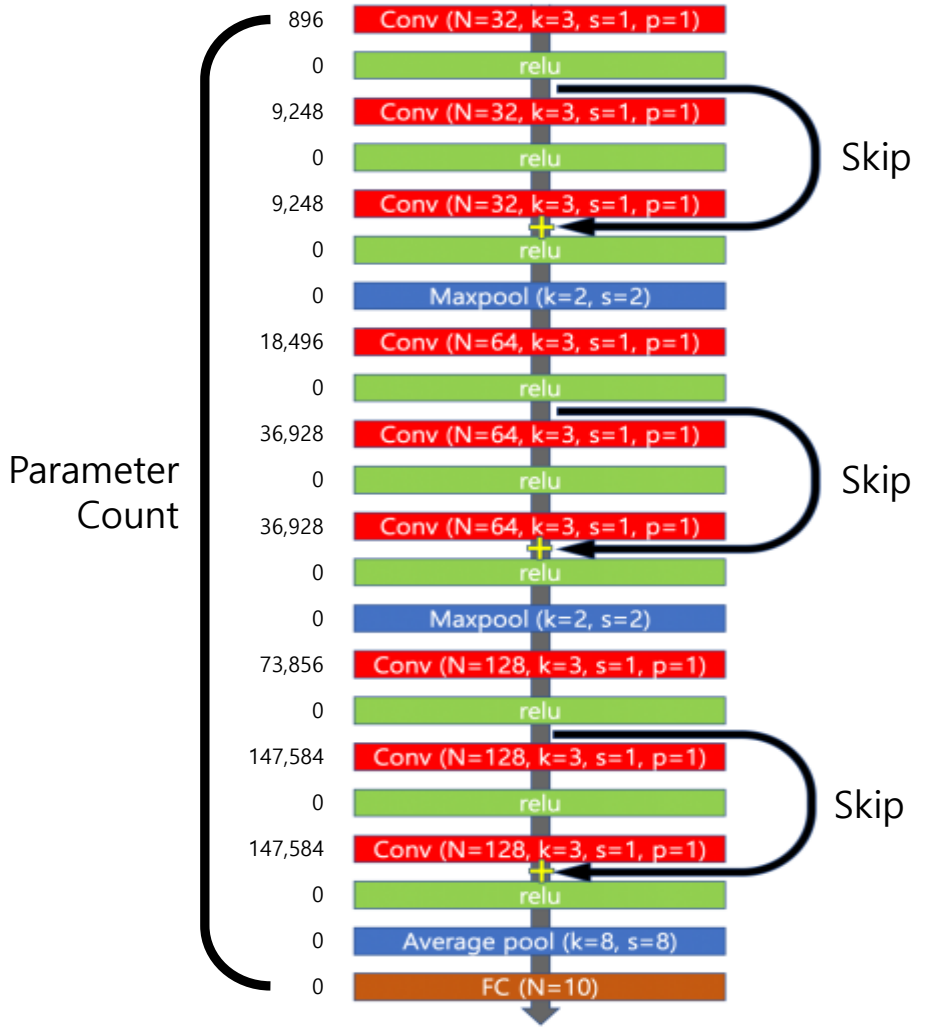


The evolution of the winning entries on the ImageNet Large Scale Visual Recognition Challenge from 2010 to 2015. Since 2012, CNNs have outperformed hand-crafted descriptors and shallow networks by a large margin. Image re-printed with permission [36].

Residual Network

- Features
  - → Parameter의 개수는 483,846개
  - → Skip 기능으로 Vanishing Gradient 문제 해결 (★)
  - → ILSVC에서 최초로 인간의 능력을 능가함

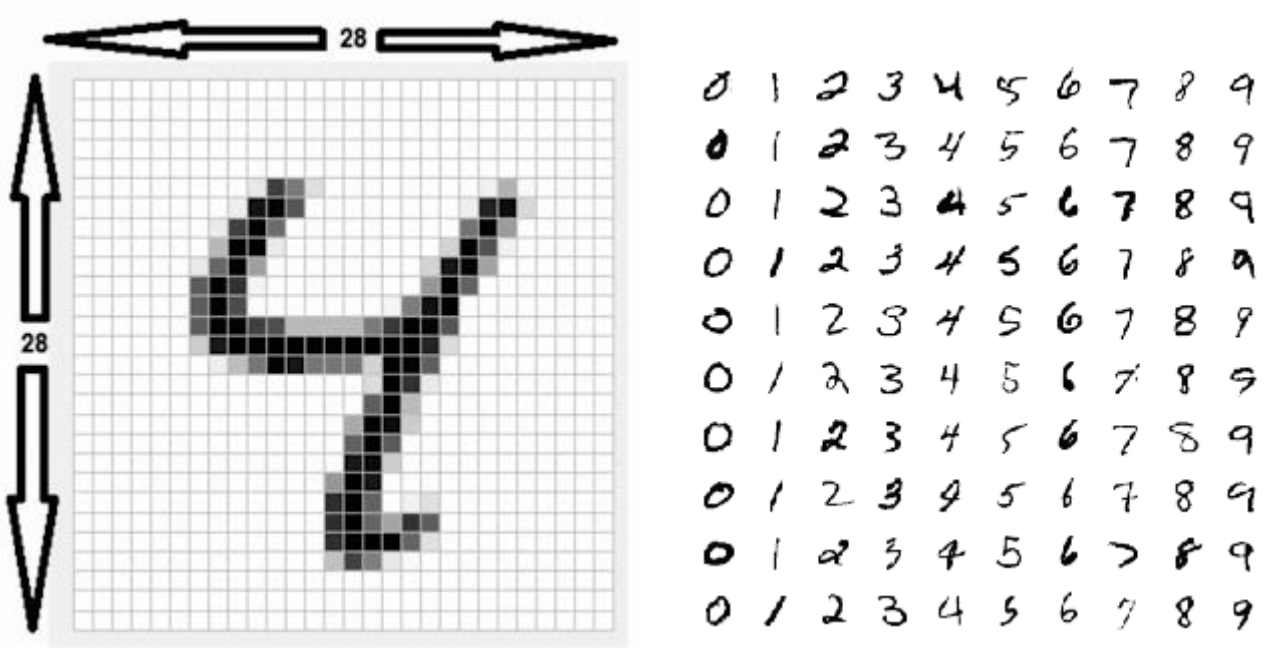




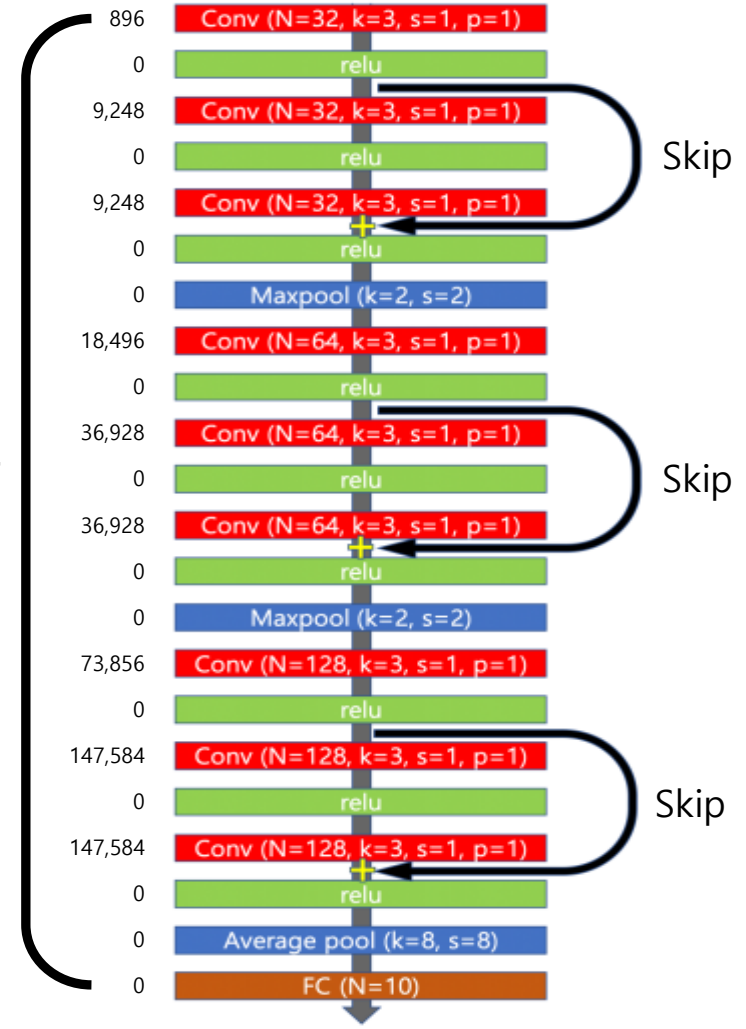
Exercise #4. (ResNet) MNIST Classification

1) Keras를 이용하여 ResNet을 만들어라.

2) 만들어진 Network을 학습시켜 MNIST 데이터를 분류하라.



Input Feature Map (28 x 28 x 흑백)

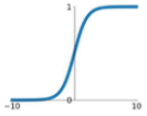


Batch Normalization

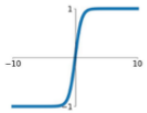
- ReLU 함수의 문제점
  - Sigmoid 또는 TanH를 쓸 경우, Backpropagation 동작 안됨
  - ReLU를 쓰면 Vanishing 문제는 해결되나

  $\sum_i w_i x_i + b$ 의 값이 Node별로 현저한 차이가 있을 수 있음

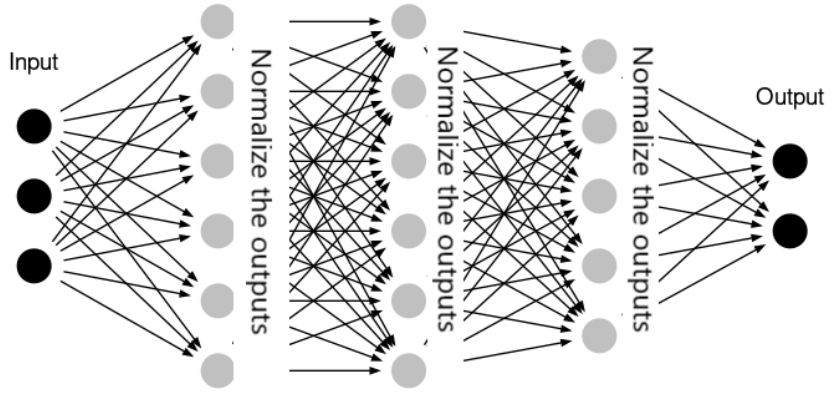
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



tanh
$\tanh(x)$



- Output의 정규화
  - 이전 Layer의 Output이 다음으로 전달되기 전에 정규화 실행
  - 정규화를 위해서는 분산과 평균이 필요, 전체 Dataset은 부하가 심하므로 Batch의 평균과 분산을 사용



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

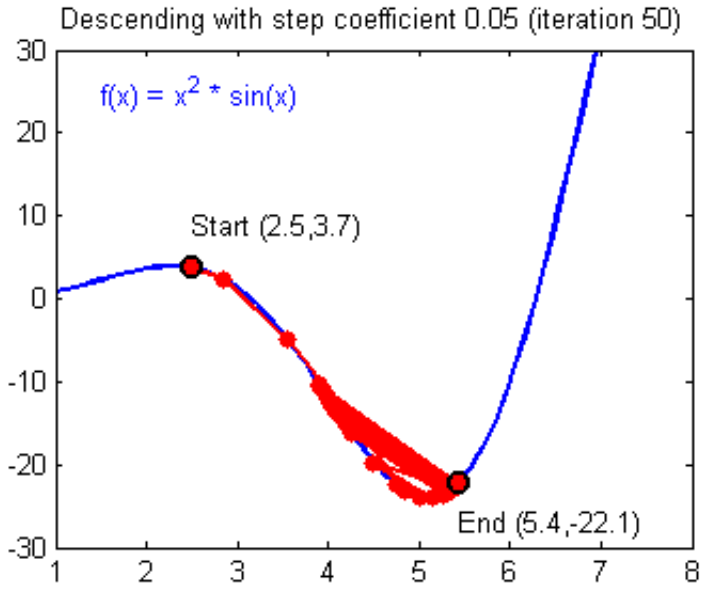
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

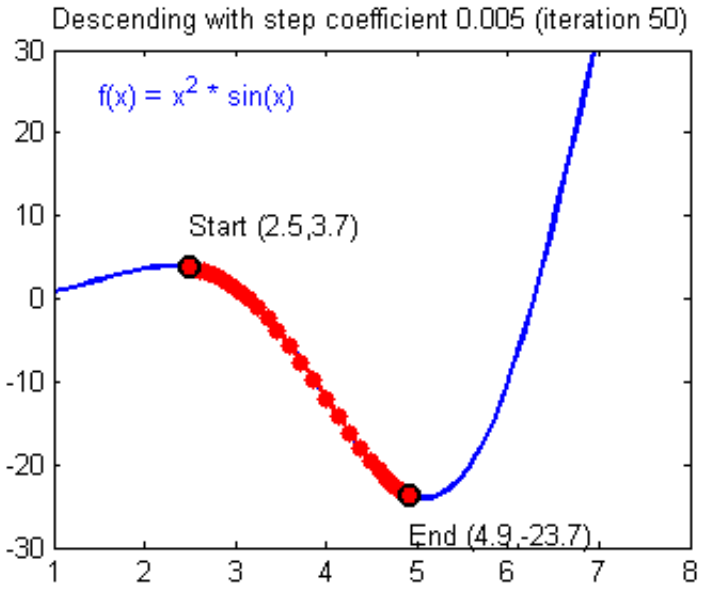
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Learning Rate Schedule

- 계단식 구간에서 일정 수준Learning Rate를 감소함
- 좀 더 빠르게 최저점을 다갈 수 있음 → Exploding 문제를 방지함



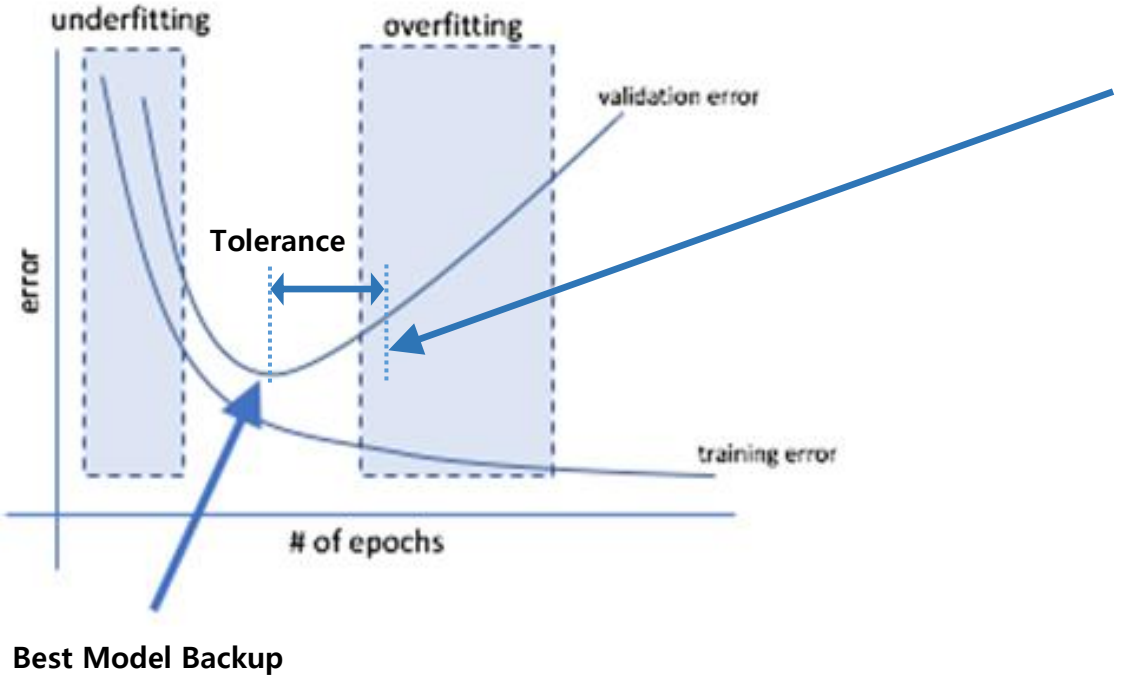
Learning Rate가 클 때



Learning Rate가 작을 때

Early Stopping

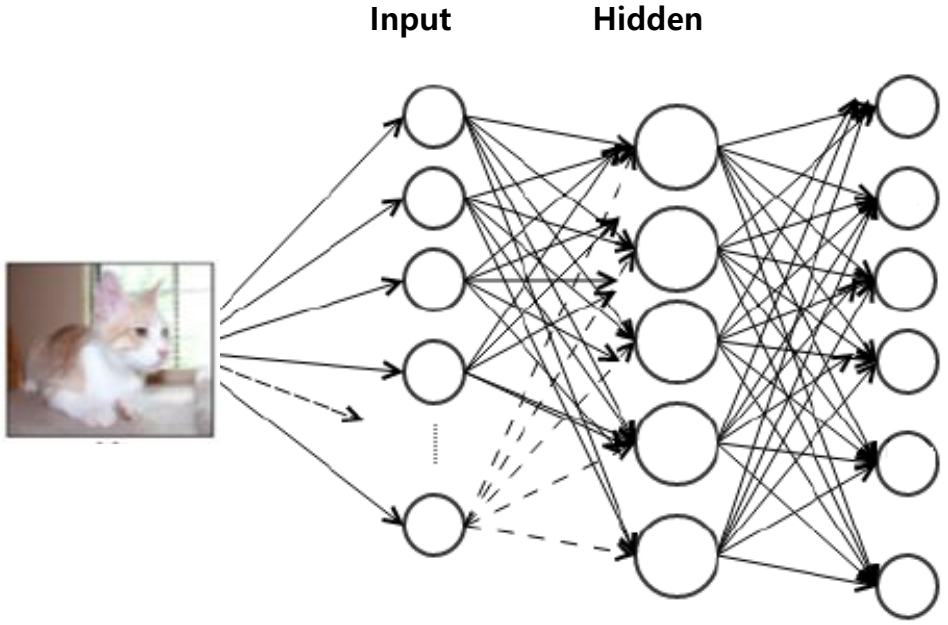
- Loss가 떨어질 때마다 Model을 Backup 해 둠
- Validation Loss의 값이 일정기간 더 감소하지 않으면 학습 중단



Stop Here !
Early finished !

Ensemble

- Train set을 Random shuffle하여 10회 수행
- 10개의 각기 다른 Model 획득 (Augmentation 효과도 병행)



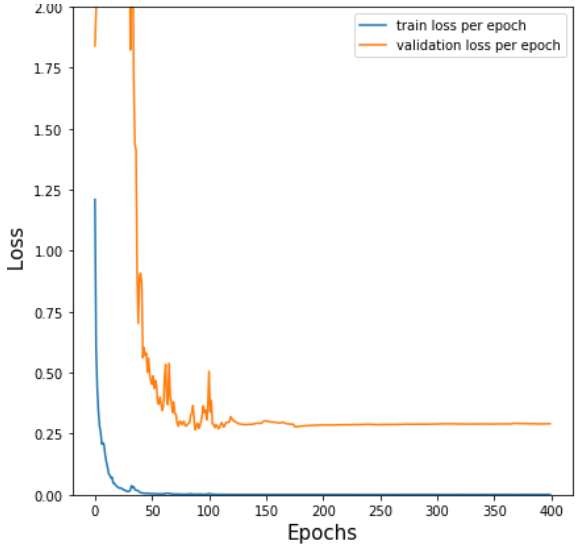
	Train (1)	Train (2)	Train (3)	…	Train (9)	Train (10)	SUM
Dog	0.12	0.11	0.11		0.11	0.12	12
Cat	0.85	0.72	0.72		0.72	0.91	91
Camera	0.23	0.21	0.21		0.21	0.18	18
Horse	0.41	0.18	0.18	…	0.18	0.22	22
Gun	0.14	0.10	0.10		0.10	0.09	9
Car	0.11	0.01	0.01		0.01	0.41	4

Exercise #5. BN, LRS, Early Stopping

1) Exercise #4에서 실습한 Network에 Optimization을 수행하라.

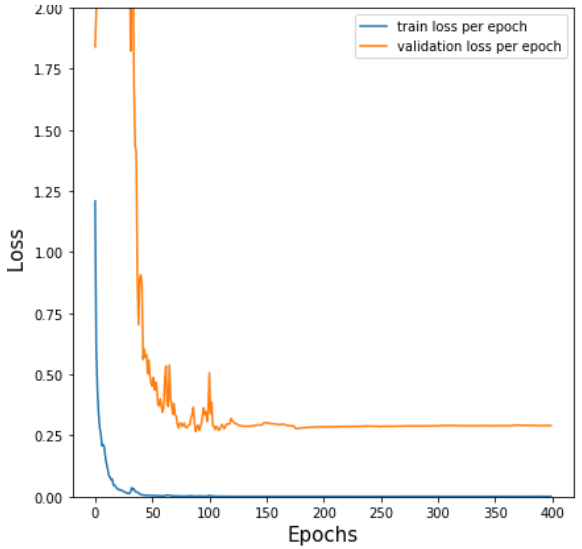
2) 성능이 나아졌음을 Graph로 표현하라.

Batch Normalization



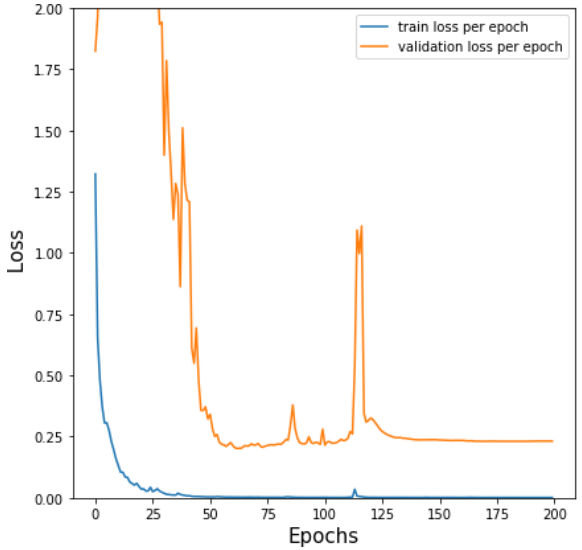
Valid: 0.9435
Test: 0.9169

+ Learning Rate Schedule



Valid: 0.9369
Test: 0.9203

+ Model Backup
+ Early stopping

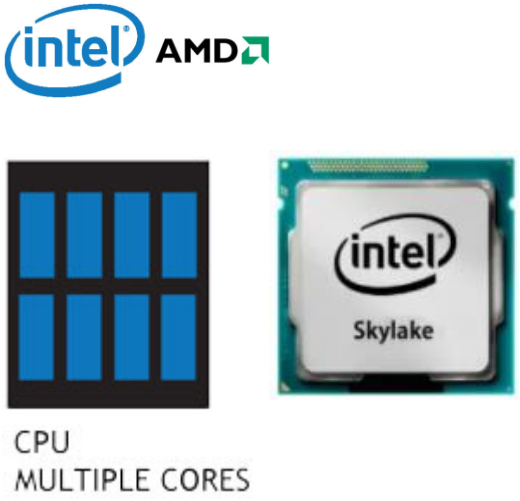


Valid: 0.9468
Test: 0.9402

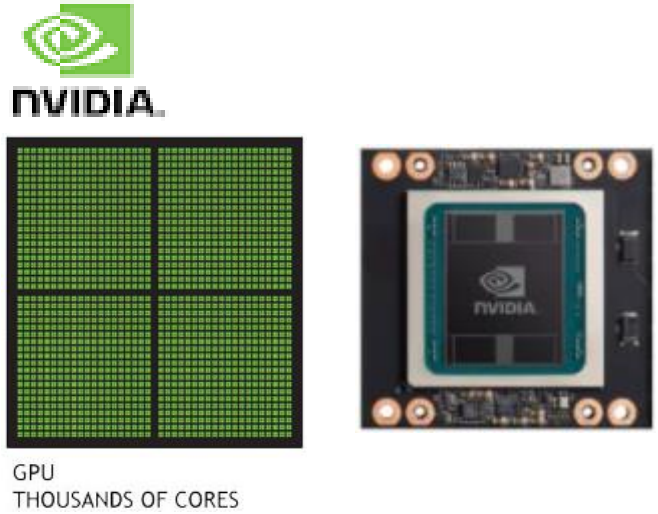
Parallel Processing using GPU

- GPU, 원래 실시간 그래픽 처리를 위해 개발 되었음
- CUDA가 개발됨에 따라 데이터 병렬 처리를 위한 보조 프로세서로서 사용됨

  (CPU = Multi-Tasking, GPU = Multi-Processing)



CPU



GPU

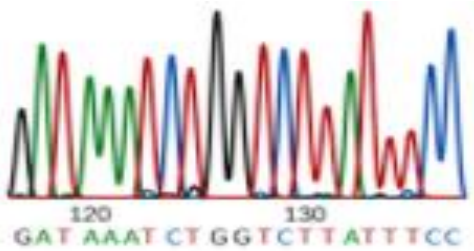
Exercise #6. Parallel Processing test using GPU

1) Python 3.7 그리고 Tensorflow 2.1을 지원하는 GPU 전용 가상환경을 생성하라.

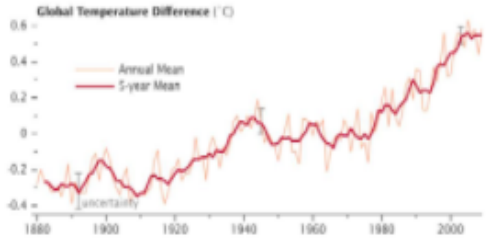
2) Exercise #5의 Code를 실행시켜보고 실행속도가 얼마나 차이가 나는지 측정하라.

Sequential Data

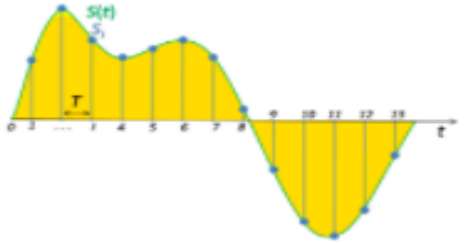
- 순서가 의미가 있으며 순서가 달라질 경우 의미가 손상되는 데이터



DNA



Stock/Share



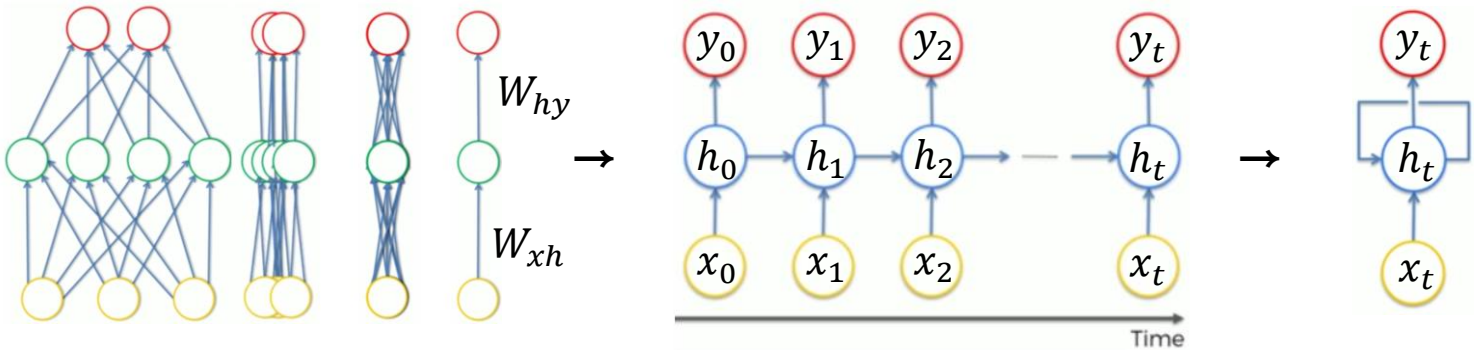
Sound

How old are you

How old you are

Language
(한국어 제외)

- RNN Architect



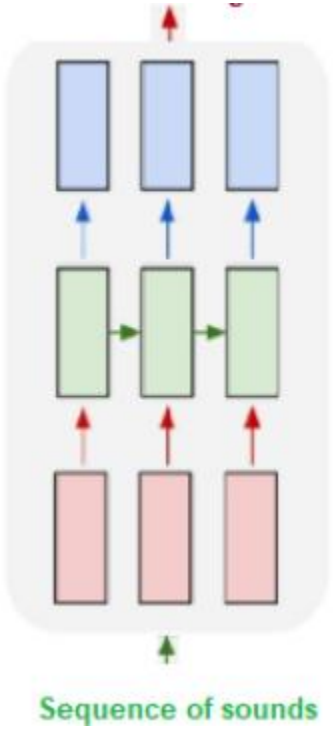
$$y_t = W_{hy} h_t$$

$$h_t = f_W(h_{t-1}, x_t)$$
$$= \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

(※ Vanilla RNN)

Many-to-Many, Many-to-One, One-to-Many

- Many-to-Many (full-sync)



예) 외국어 번역기

순번	입력
1	I
2	have
3	a
4	pen.

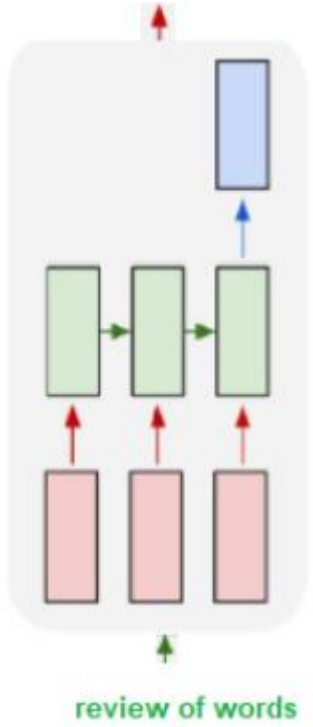
⇨ Sequence Data Processor ⇨

순번	출력
1	나는
2	펜을
3	가지고 있다.

번역기는 다중 입력에 대해 다중 출력을 해 주는 모델이다. 물론 입력과 출력의 길이는 다를 수 있다.

Many-to-Many, Many-to-One, One-to-Many

- Many-to-One



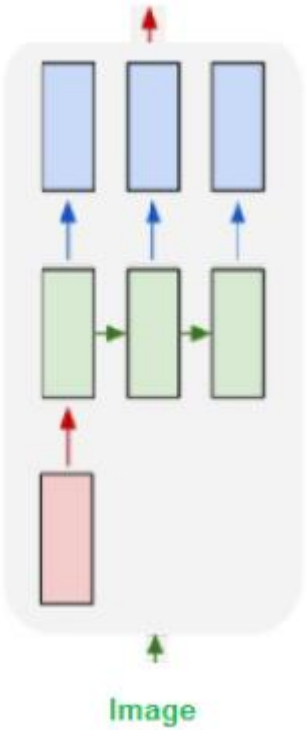
예) 개인 비서 서비스

순번	입력
1	What
2	time
3	is
4	it?

개인 비서 서비스는 다중 입력에 대해 단일 출력을 하는 모델이다.

Many-to-Many, Many-to-One, One-to-Many

- One-to-Many



예) Image Captioning

Sequence Data Processor

순번	출력
1	Cat
2	is
3	eating.

사진을 묘사하는 장면 이해 알고리즘은 단일 입력에 대해 다중 출력을 내는 모델이다.

Exercise #7. (Many-to-Many) One-hot Encoding

1) NumPy를 이용하여 다음의 주어진 Text Data를 Indexing하는 함수를 작성하라.

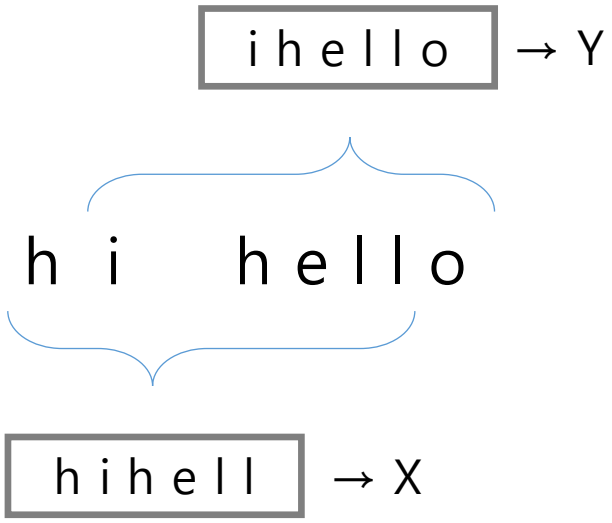
2) 또한, 주어진 데이터를 해당 요소 번째만 1을 갖고 나머지는 0을 갖는 Code로 변환하라.

(※ 학습목표: Text Data를 ANN에 입력할 수 있게 변형하는 것)

Index

e	0	1, 0, 0, 0, 0
l	1	0, 1, 0, 0, 0
i	2	0, 0, 1, 0, 0
o	3	0, 0, 0, 1, 0
h	4	0, 0, 0, 0, 1

검색어 자동완성 Data



One-hot Code

→

```
y_enc: [[[0. 0. 1. 0. 0.]
         [0. 0. 0. 0. 1.]
         [1. 0. 0. 0. 0.]
         [0. 1. 0. 0. 0.]
         [0. 1. 0. 0. 0.]
         [0. 0. 0. 1. 0.]]]
```

→

```
x_enc: [[[0. 0. 0. 0. 1.]
         [0. 0. 1. 0. 0.]
         [0. 0. 0. 0. 1.]
         [1. 0. 0. 0. 0.]
         [0. 1. 0. 0. 0.]
         [0. 1. 0. 0. 0.]]]
```

Exercise #8. (Many-to-One) Token Embedding

1) Keras Embedding Layer를 이용하여 다음의 주어진 Text Data의 긍정/부정 여부를 예측하라.

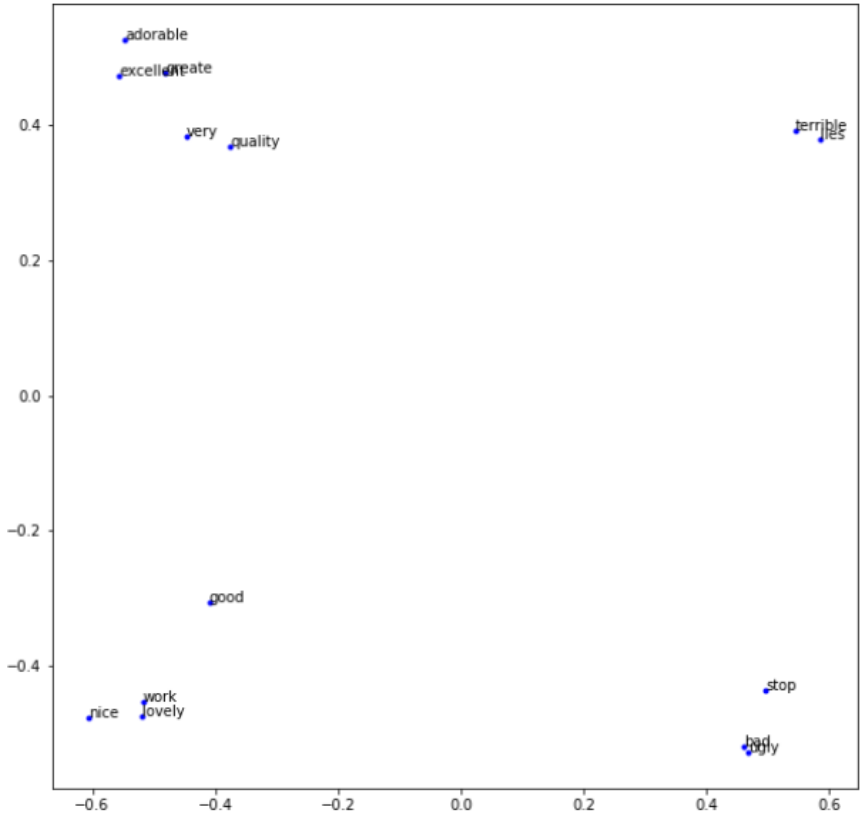
2) 결과를 그래프로 표현하라.

Label

X	Y
Very good nice quality	1
Stop lies	0
Ugly terrible	0
Excellent work	1
Adorable lovely	1
Bad	0
Great nice	1

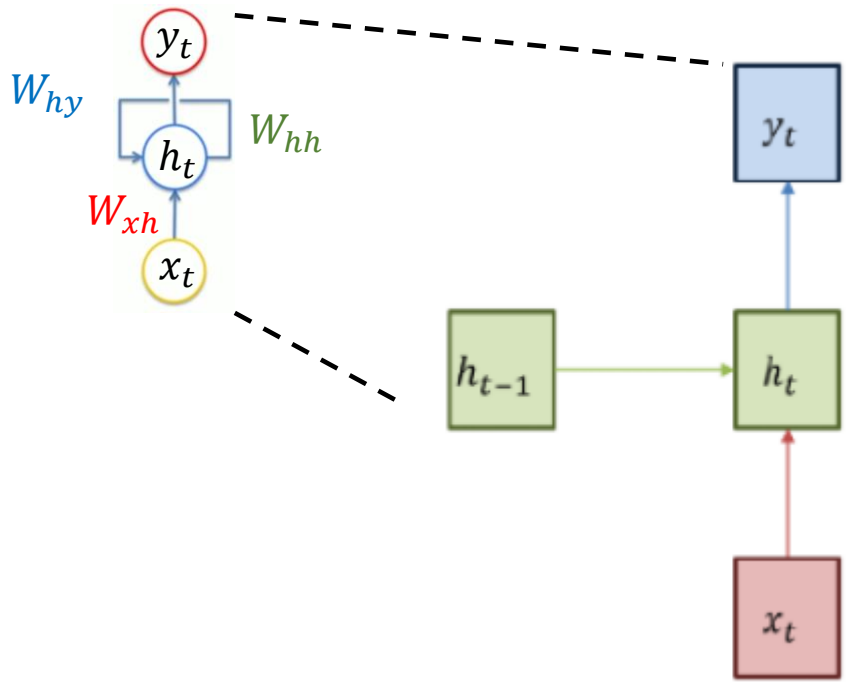
Embedded Vector

```
Terrible [[ 0.46805128  0.45514685]
Nice      [-0.6060485  -0.47585234]
Excellent [-0.44584236  0.38378528]
          [-0.4085907  -0.3049673 ]
Verry     [-0.37545377  0.3680631 ]
Qualty    [ 0.49695083 -0.43553331]
Stop      [ 0.5856042   0.37878647]
          [ 0.46793577 -0.5277508 ]
Bad       [ 0.5443586   0.39237097]
Ugly      [-0.55632174  0.4738349 ]
Terrible  [-0.5174578  -0.45235133]
Lies      [-0.5468264   0.5264467 ]
          [-0.5186092  -0.47502556]
Adrable   [ 0.46115583 -0.51825726]
great     [-0.4813648   0.47686896]]
```



Vanilla RNN

- 아무것도 첨가하지 않은 Vanilla 아이스크림 같은 RNN 이란 뜻.
- Architect



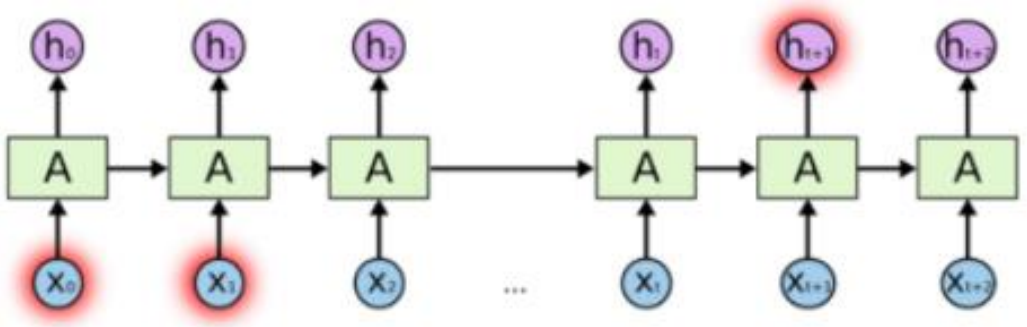
$$y_t = W_{hy} h_t + b_y$$

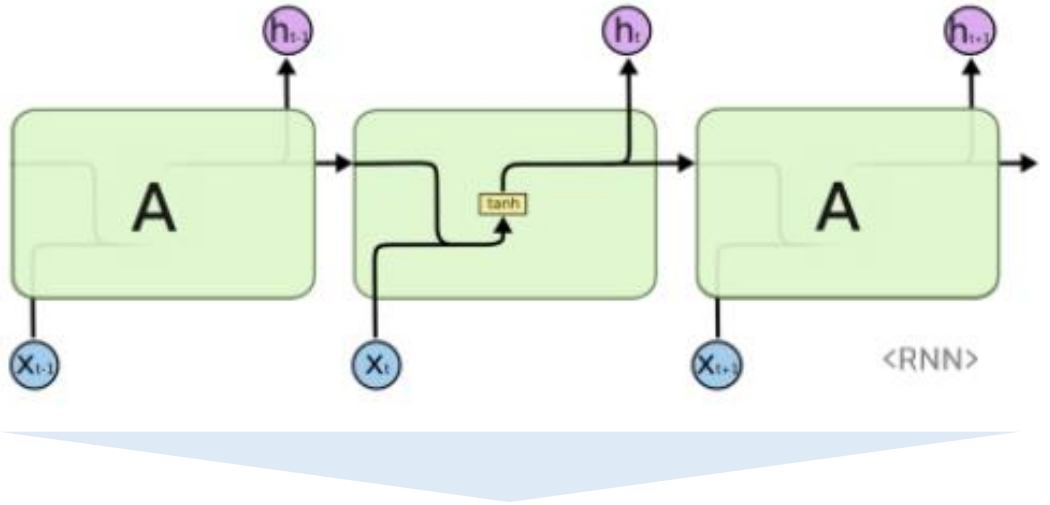
$$h_t = f_W(h_{t-1}, x_t)$$

$$= \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

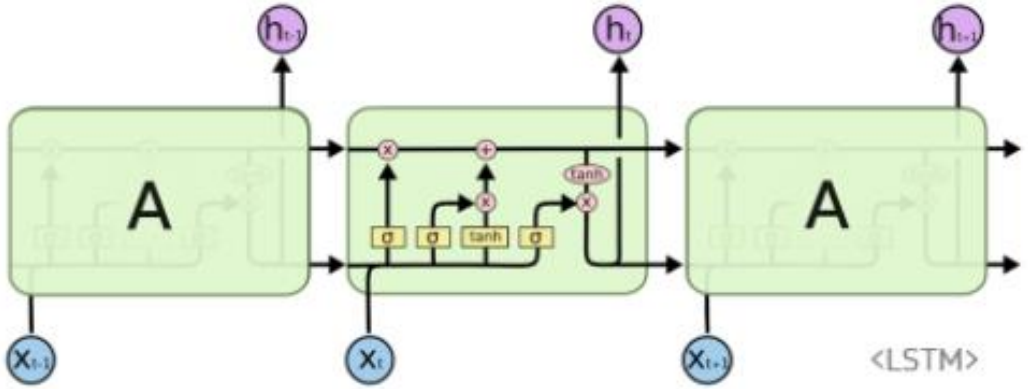
LSTM

- Vanishing Gradient (★) Problem of Vanilla RNN



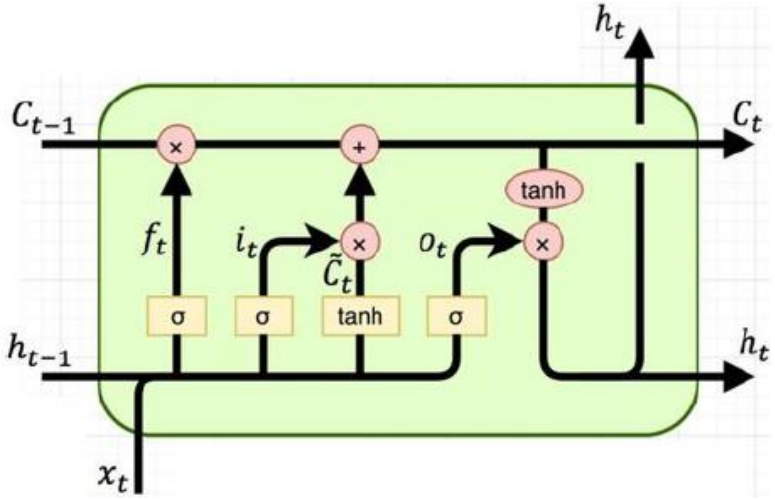


- LSTM, Hidden State에 Cell State를 추가
  - ① Long Short-Term Memory Unit
  - ② 셉 호흐라이터, 유르겐 슈미트 후버 1997년 제안
  - ③ RNNdml 장기의존성 (Longterm Dependency) 문제보안

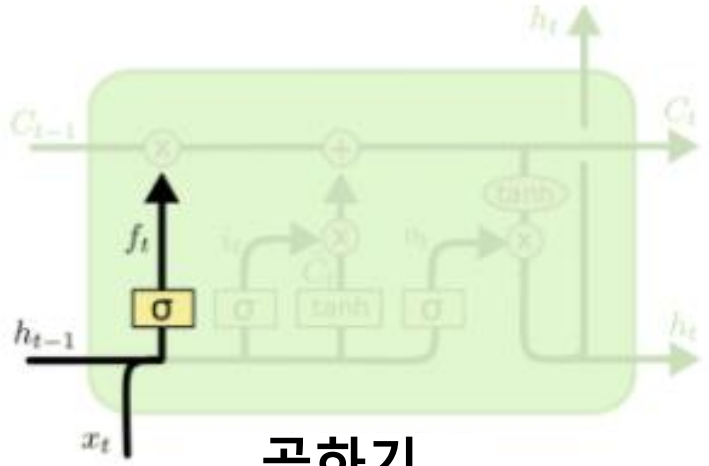


5. Recurrent Neural Network

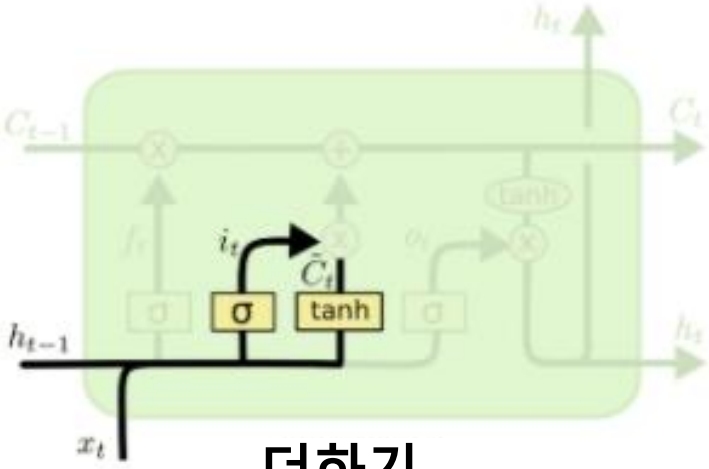
- Cell State



$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$
$$h_t = \tanh(C_t) * o_t$$



곱하기



더하기

forget gate f_t는 '과거 정보를 잊기'를 위한 게이트입니다. h_{t-1}과 x_t를 받아 시그모이드를 취해준 값이 바로 forget gate가 내보내는 값이 됩니다. 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에 그 값이 0이라면 이전 상태의 정보는 잊고, 1이라면 이전 상태의 정보를 온전히 기억하게 됩니다.

input gate $i_t \odot g_t$는 '현재 정보를 기억하기' 위한 게이트입니다. h_{t-1}과 x_t를 받아 시그모이드를 취하고, 또 같은 입력으로 하이퍼볼릭탄젠트를 취해준 다음 Hadamard product 연산을 한 값이 바로 input gate가 내보내는 값이 됩니다. 개인적으로 i_t의 범위는 0~1, g_t의 범위는 -1~1이기 때문에 각각 강도와 방향을 나타낸다고 이해했습니다.

Exercise #9. LSTM

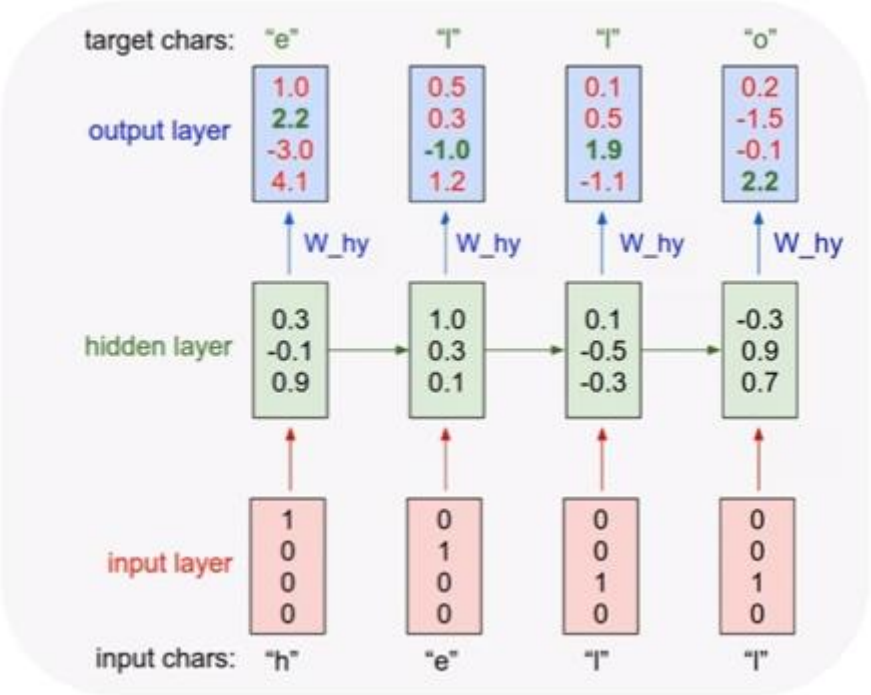
1) Keras API를 이용하여 LSTM을 생성하라.

2) Character-level language model example model을 보고 검색어 자동생성 기능을 구현하라.

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
"hello"



검색어 자동완성 Data

i h e l l o → Y

h i h e l l o

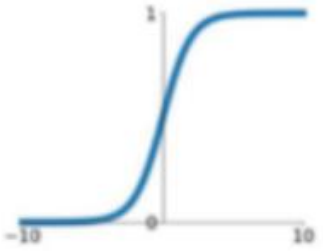
h i h e l l → X

End of Document

Appendix #1. Activate Functions

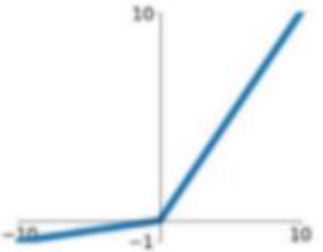
Sigmoid

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



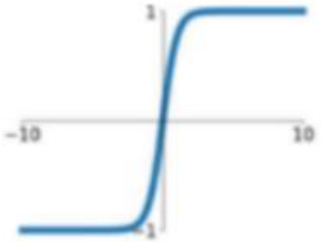
Leaky ReLU

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



tanh

$$\tanh(x)$$

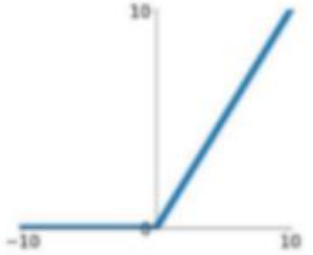


Maxout

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

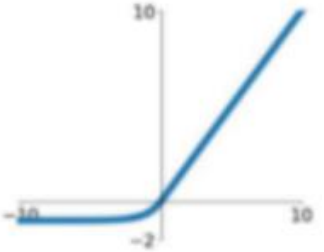
ReLU

$$\max(0, x)$$



ELU

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



Appendix #2. GPU Support

재배포 절대금지

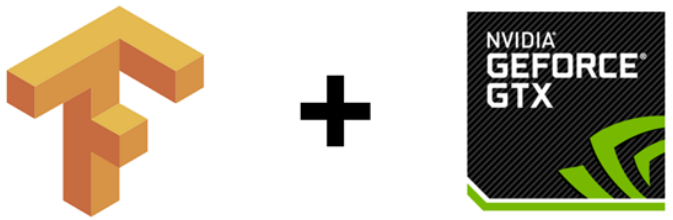
GeForce and TITAN Products

GPU	Compute Capability
GeForce RTX 3090	8.6
GeForce RTX 3080	8.6
GeForce RTX 3070	8.6
NVIDIA TITAN RTX	7.5
Geforce RTX 2080 Ti	7.5
Geforce RTX 2080	7.5
Geforce RTX 2070	7.5
Geforce RTX 2060	7.5
NVIDIA TITAN V	7.0
NVIDIA TITAN Xp	6.1
NVIDIA TITAN X	6.1
GeForce GTX 1080 Ti	6.1
GeForce GTX 1080	6.1
GeForce GTX 1070 Ti	6.1
GeForce GTX 1070	6.1
GeForce GTX 1060	6.1
GeForce GTX 1050	6.1
GeForce GTX TITAN X	5.2
GeForce GTX TITAN Z	3.5
GeForce GTX TITAN Black	3.5
GeForce GTX TITAN	3.5

GeForce Notebook Products

GPU	Compute Capability
Geforce RTX 2080	7.5
Geforce RTX 2070	7.5
Geforce RTX 2060	7.5
GeForce GTX 1080	6.1
GeForce GTX 1070	6.1
GeForce GTX 1060	6.1
GeForce GTX 980	5.2
GeForce GTX 980M	5.2
GeForce GTX 970M	5.2
GeForce GTX 965M	5.2
GeForce GTX 960M	5.0
GeForce GTX 950M	5.0
GeForce 940M	5.0
GeForce 930M	5.0
GeForce 920M	3.5
GeForce 910M	5.2
GeForce GTX 880M	3.0
GeForce GTX 870M	3.0
GeForce GTX 860M	3.0/5.0(**)
GeForce GTX 850M	5.0
GeForce 840M	5.0

GTX 1060 (6GB)



Appendix #3. Cross Entropy

- 크로스 엔트로피(Cross Entropy)
  - P(x)와 Q(x)가 서로 교차해서 곱한다는 의미
    - 실제 확률을 모르기 때문에 KLD값을 최소화 하기위해서는 Q(x)를 최소화해야하기 때문
    - $H(P,Q) = E_{X\sim P}[-logQ(x)] = -\sum xP(x)logQ(x) = H(P) + D_{KL}\,(P||Q)$
  - 크로스 엔트로피는 딥러닝 모델의 손실함수로 자주 쓰임
    - 딥러닝 모델을 학습 할 때 크로스 엔트로피를 최소화하는 방향으로 가중치를 업데이트
    - 크로스 엔트로피를 최소화한다는 의미는 KLD를 최소화하는 것과 의미가 같음
      - 데이터 P(x)는 변화하지 않기 때문

Appendix #4. 지도학습 vs 비지도학습 vs 강화학습

Appendix #5. Image Augmentation