Artificial Intelligence

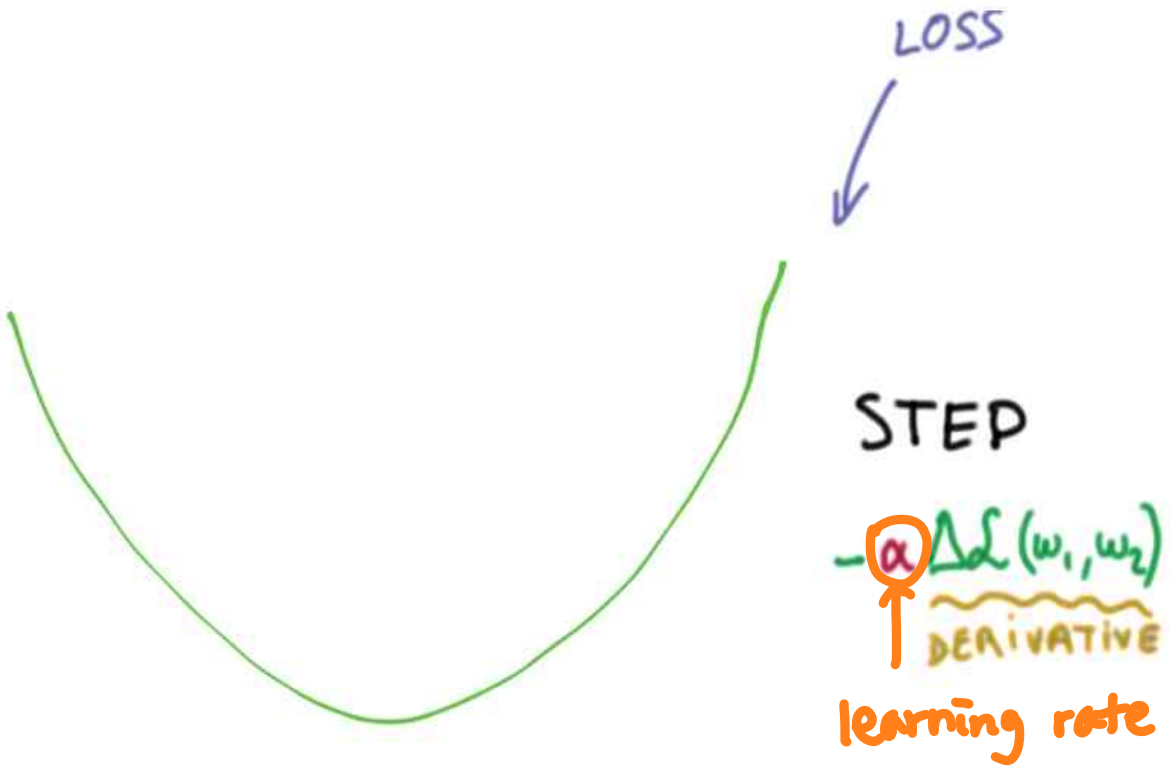
Yukyung Choi

Agenda

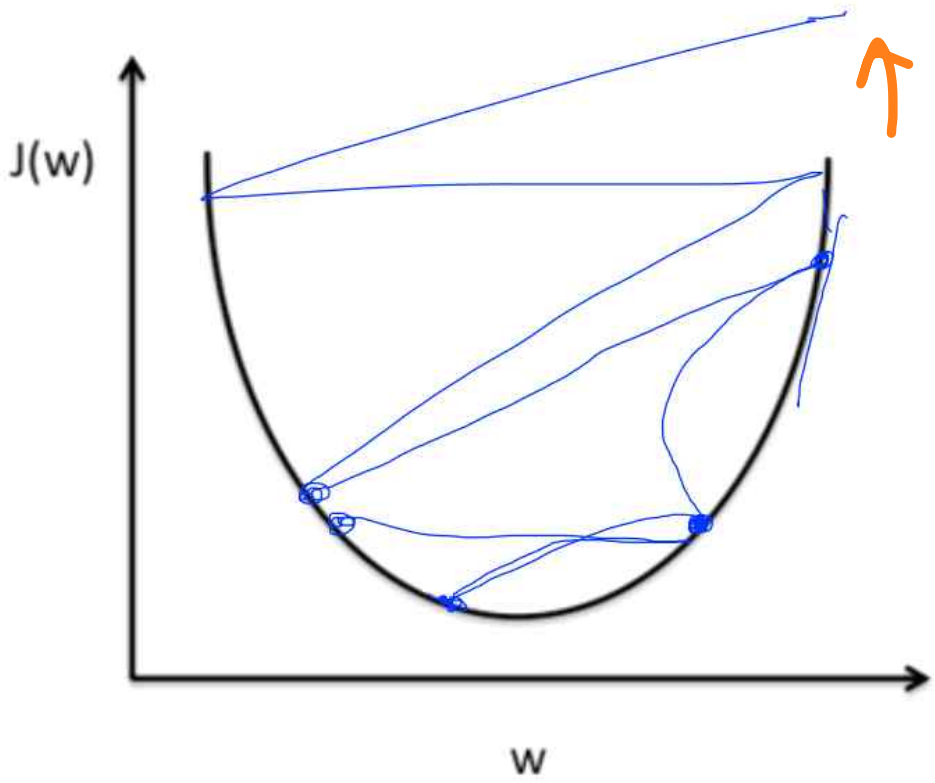
- Learning rate
- Data preprocessing
- Avoid overfitting
  - More training data
  - Regularization
- Performance evaluation

What is a Learning rate

Loss function



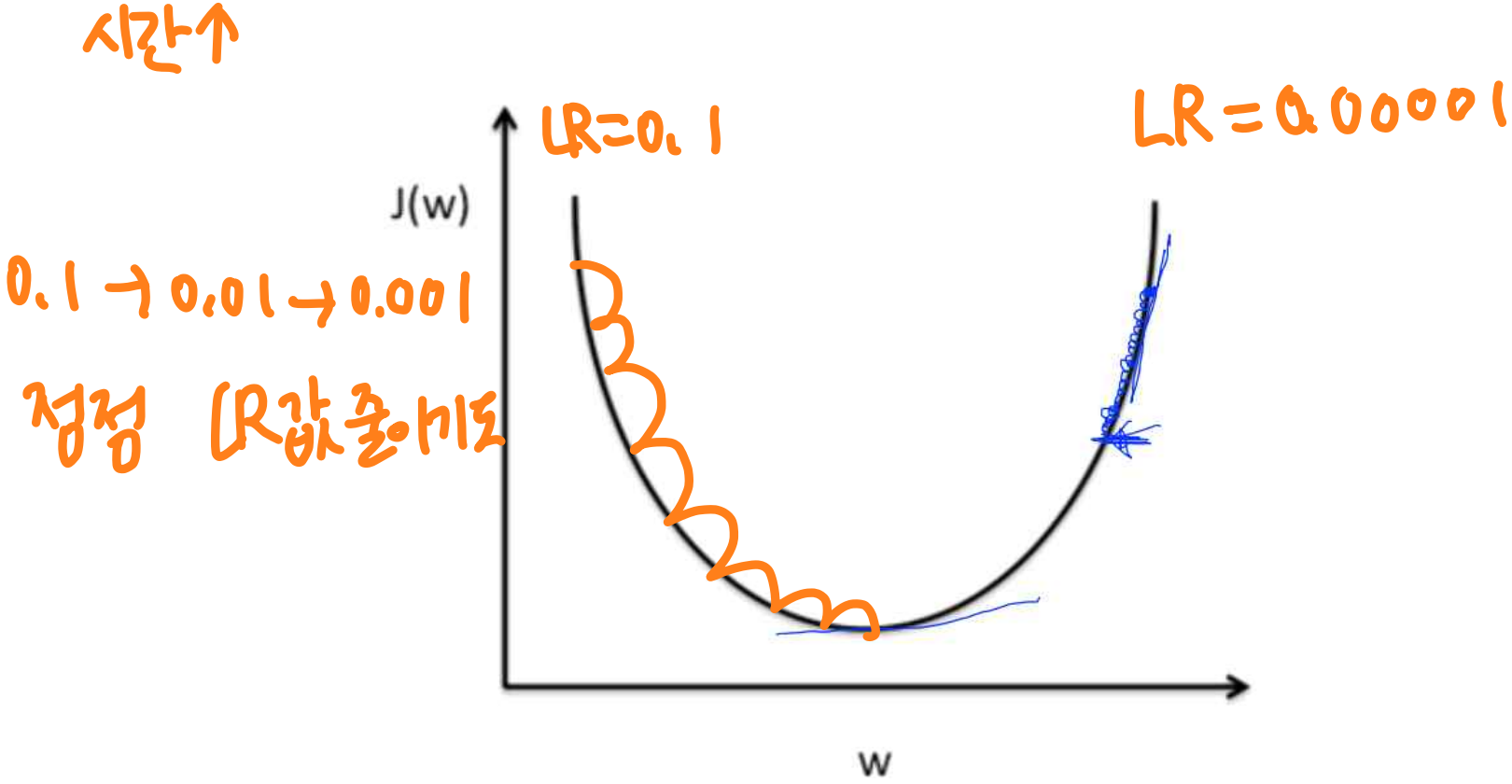
Large learning rate: overshooting



⇒ Loss : Nan
(발산 INF ∞)

Small learning rate

- Takes too long, stops at local minimum



Try several learning rates

- Observe the cost function

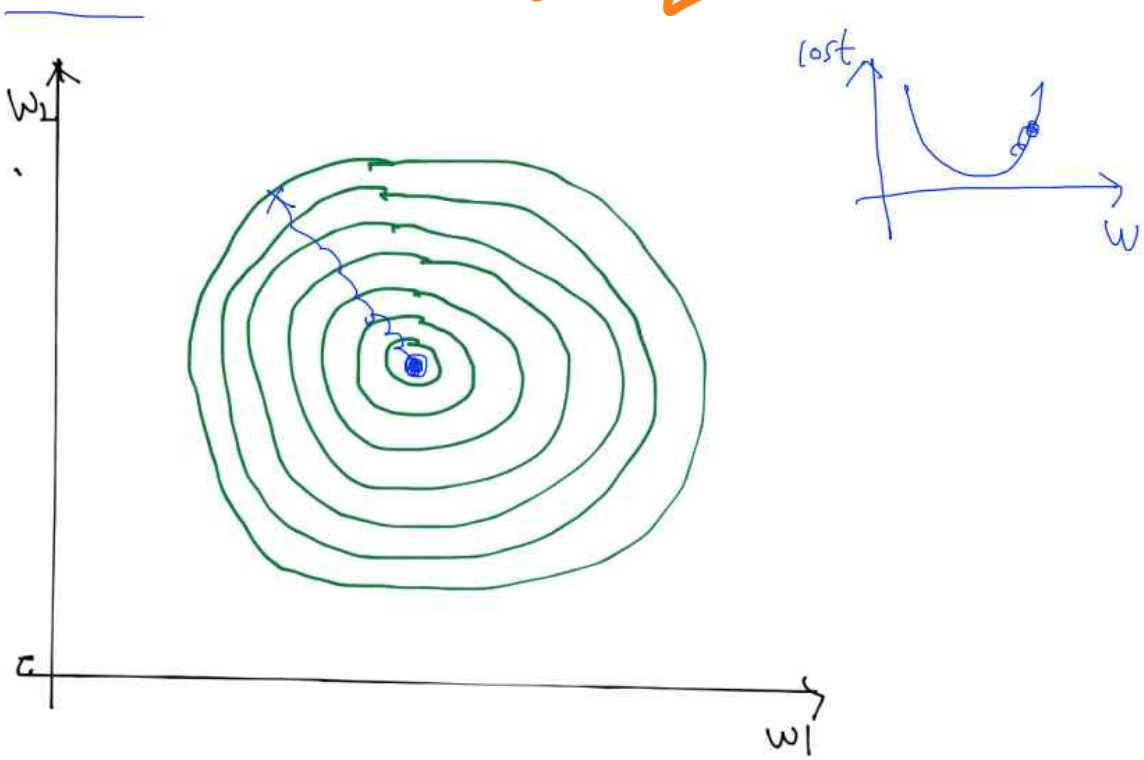
- Check it goes down in a reasonable rate

최적값 정해져있지 X
실험적

Data Preprocessing for gradient descent

데이터정규화

⇊정방향

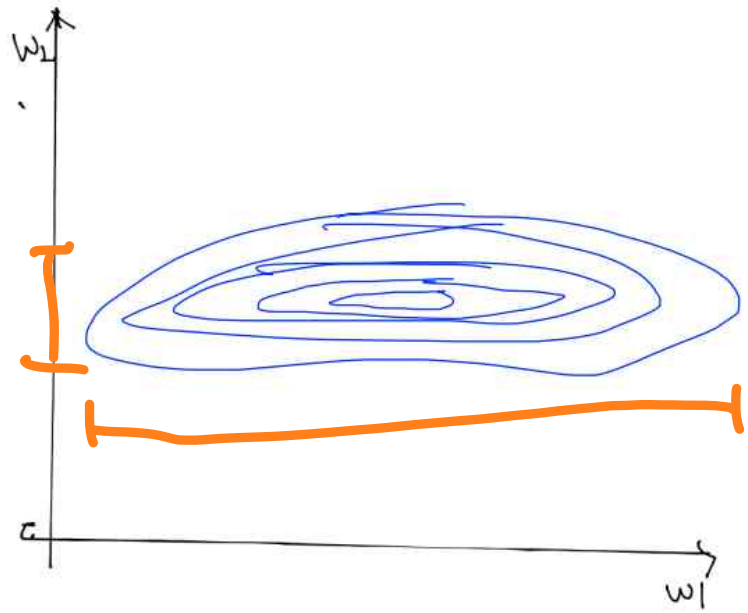


Data Preprocessing for gradient descent

x1	x2	y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C

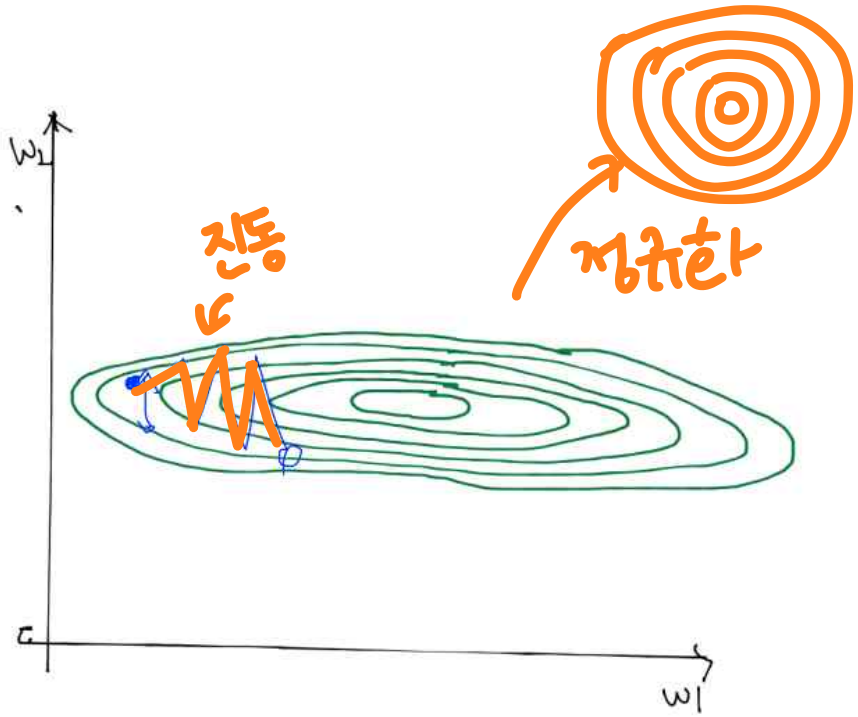
0 0 1 1 2

scale이 다름 → 너무

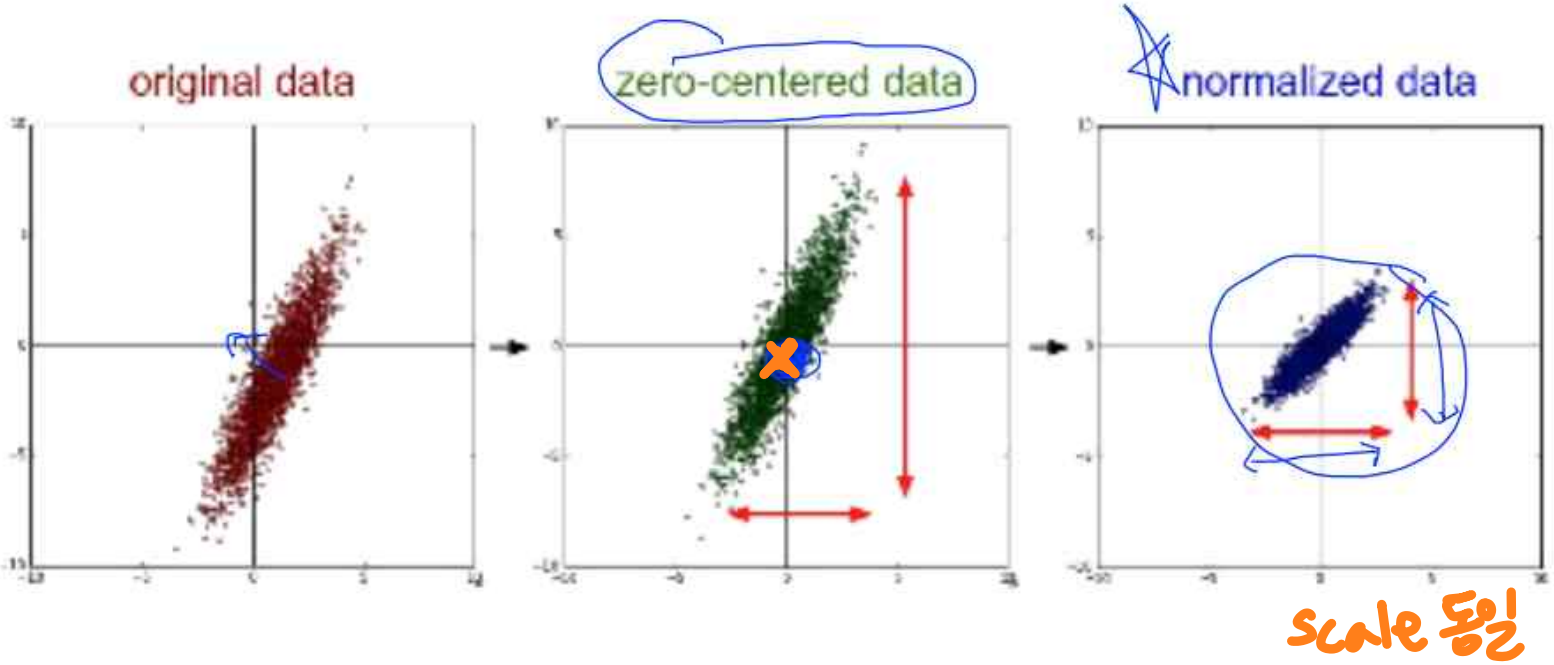


Data Preprocessing for gradient descent

x1	x2	y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C



Data Preprocessing for gradient descent



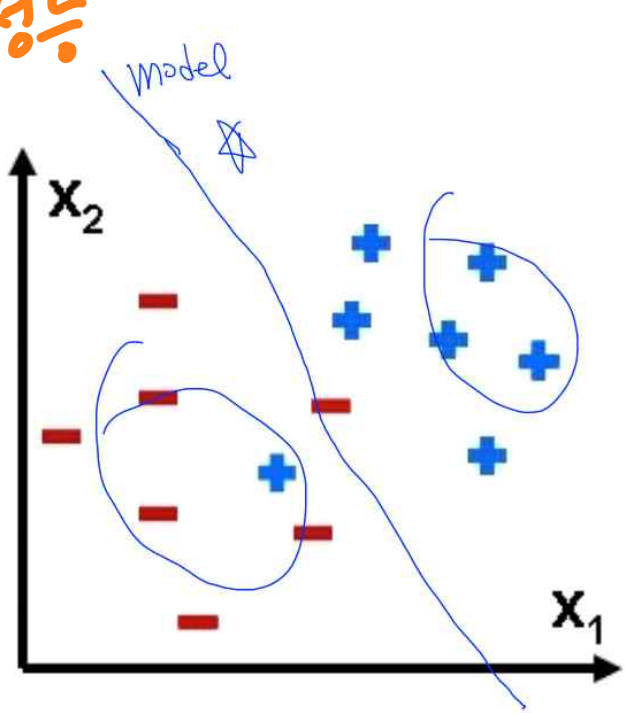
Standardization: $$x_j' = \frac{x_j - \mu_j}{\sigma_j}$$

What is a overfitting

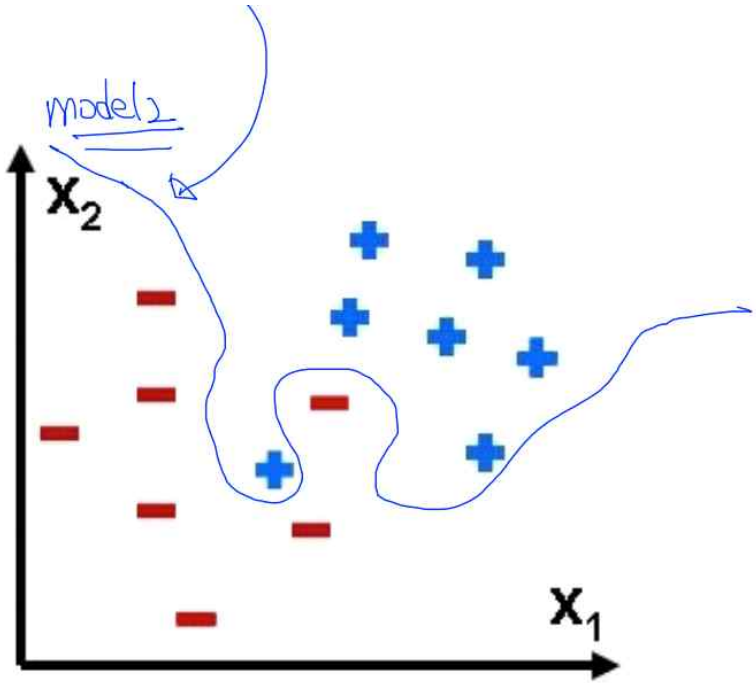
- Our model is very good with training data set(with memorization)
- Not good at test dataset or in real use

학습 성능 ↑ 유사 ↓
테스트성능

Good model



Overfitting model



Solutions for overfitting

- More training data
- Reduce the number of features
- **Regularization**

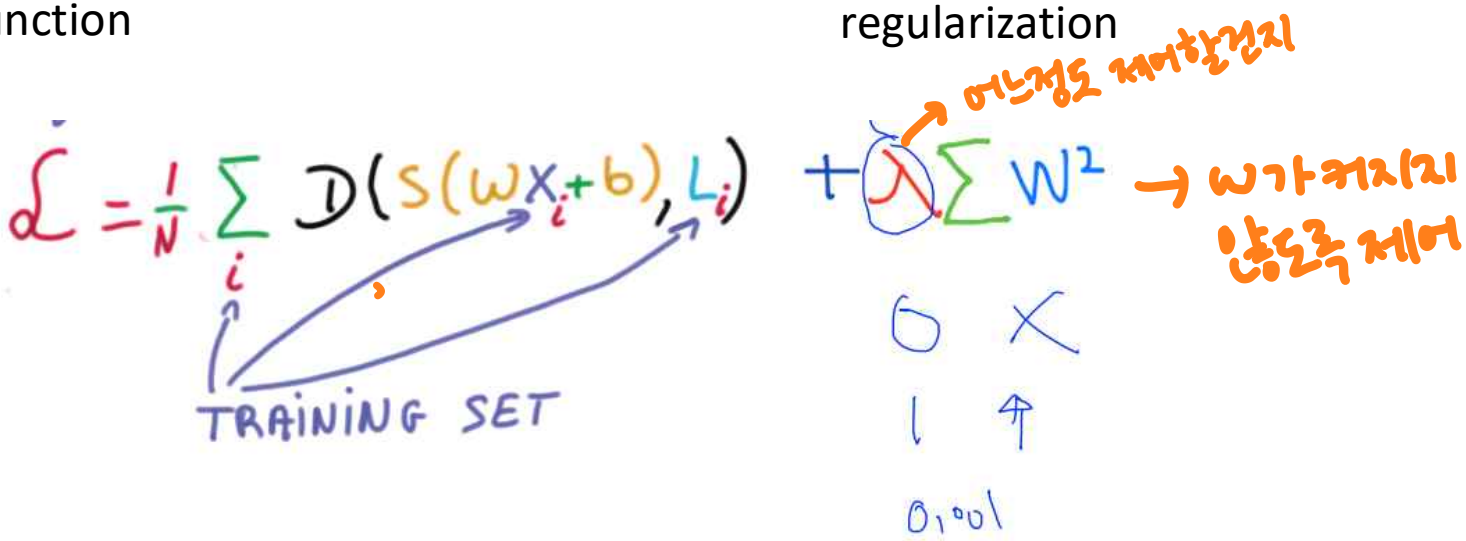
정규화

Regularization

- Let's not have too big numbers in the weight

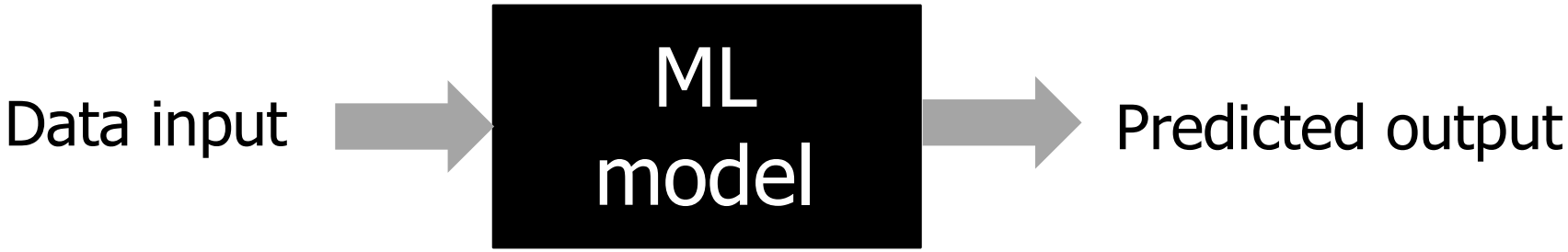
Loss function

regularization



Performance evaluation

- Use the learned parameter (W,b)
- Use the unseen data
  - Do not use training data



Evaluation using training set

- Just memorization instead of understanding

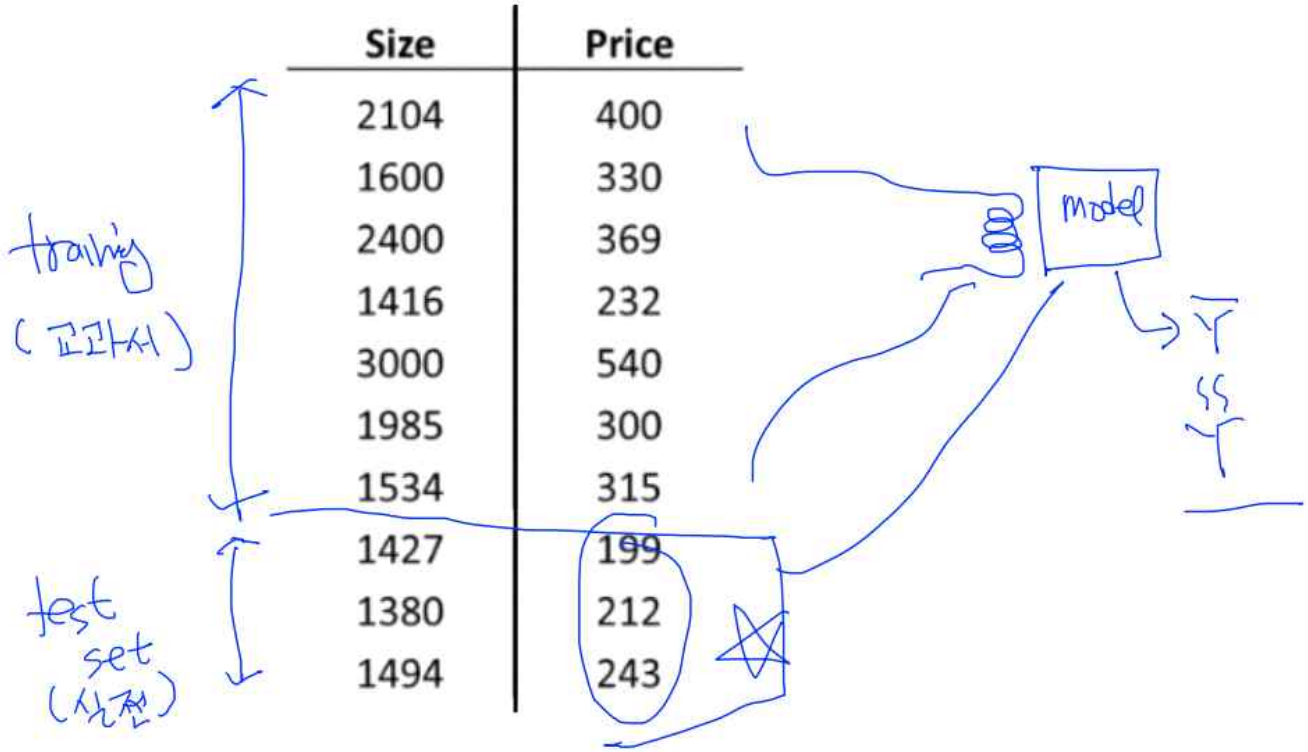
- For example) Examination

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

Training data

100% correct (accuracy)

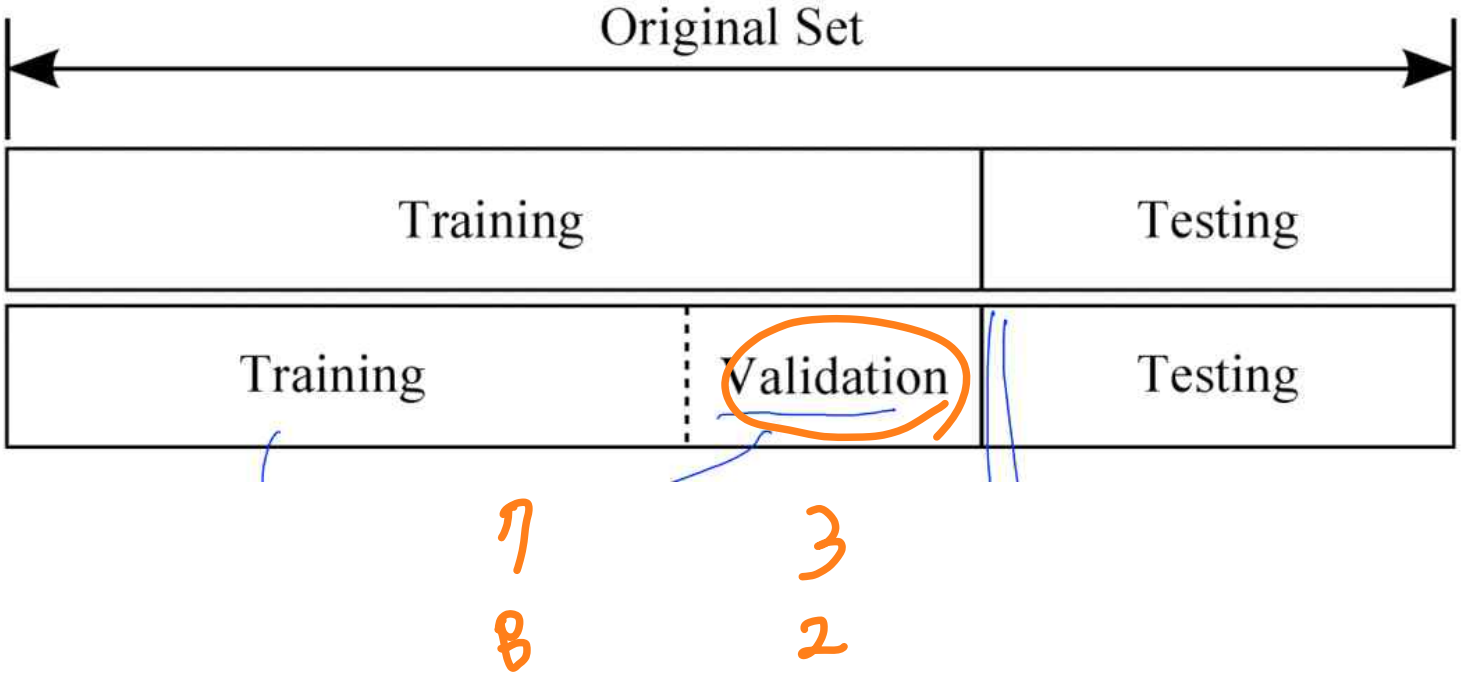
Evaluation using disjointed train & test sets



Training, validation and test sets

↑ 학습하는 과정 중간중간 모델 평가하고 싶다.

↳ 정답이 주어지지 X



Summary

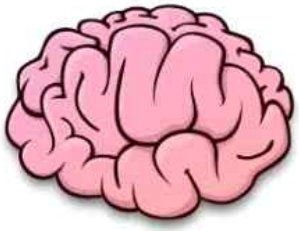
- Learning rate
- Data preprocessing
- Avoid overfitting
  - More training data
  - Regularization
- Performance evaluation

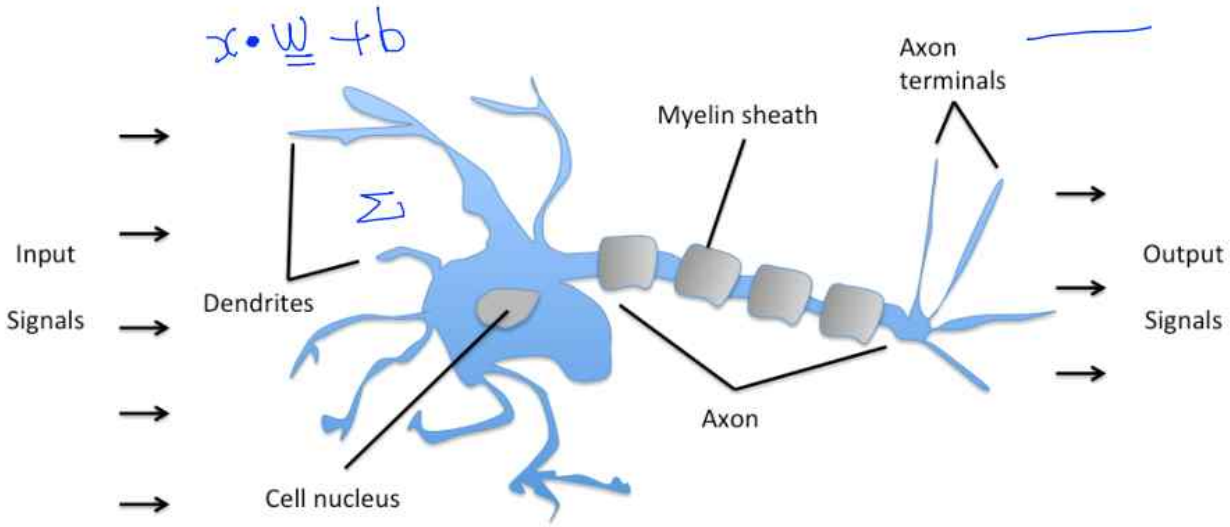
Agenda

- History of MLP

Ultimate dream: thinking machine

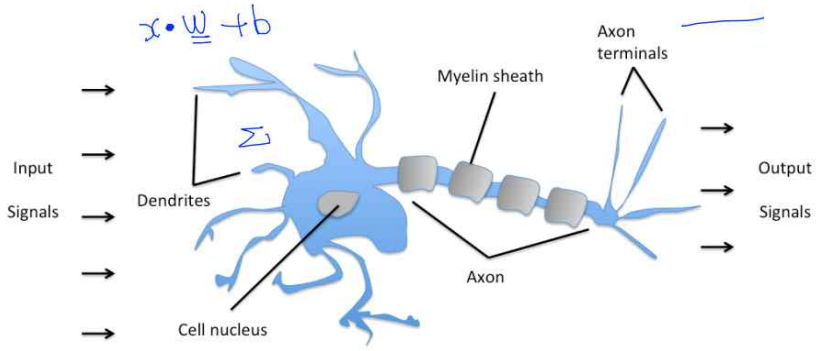
NN: 신경망



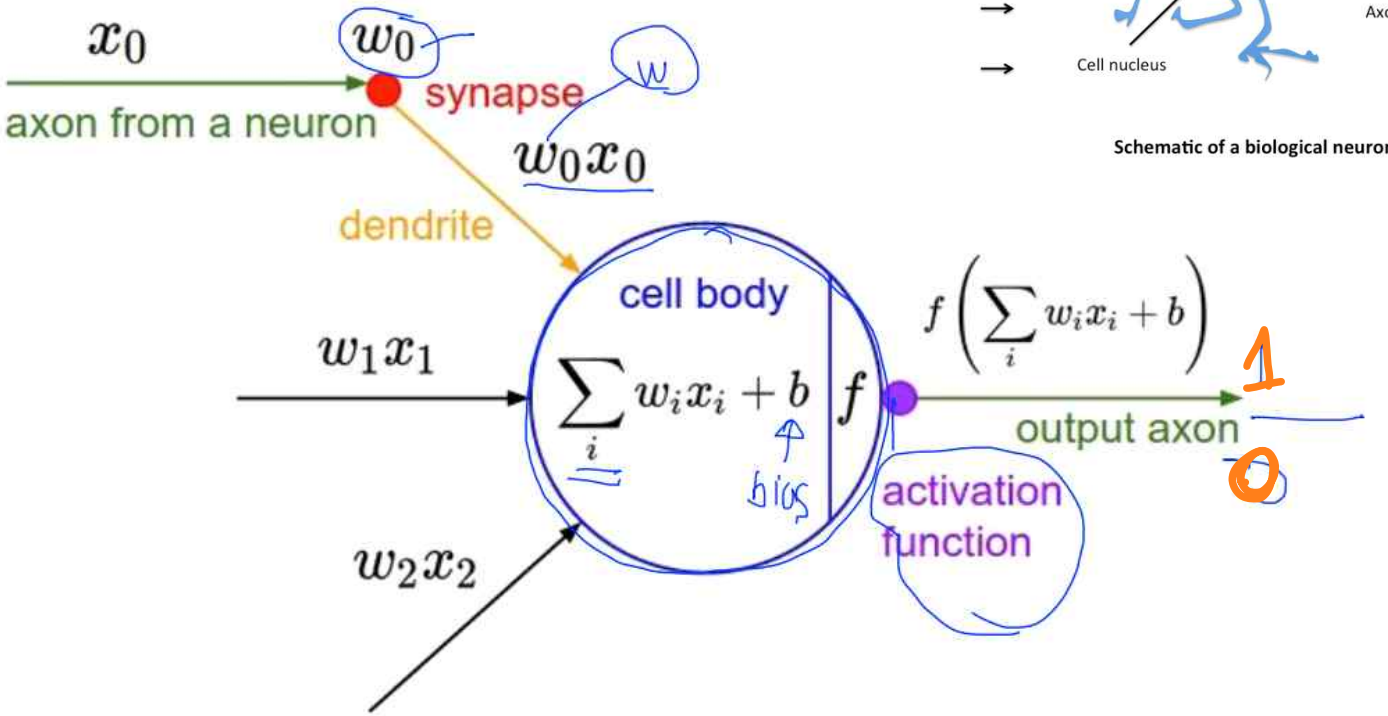


Schematic of a biological neuron.

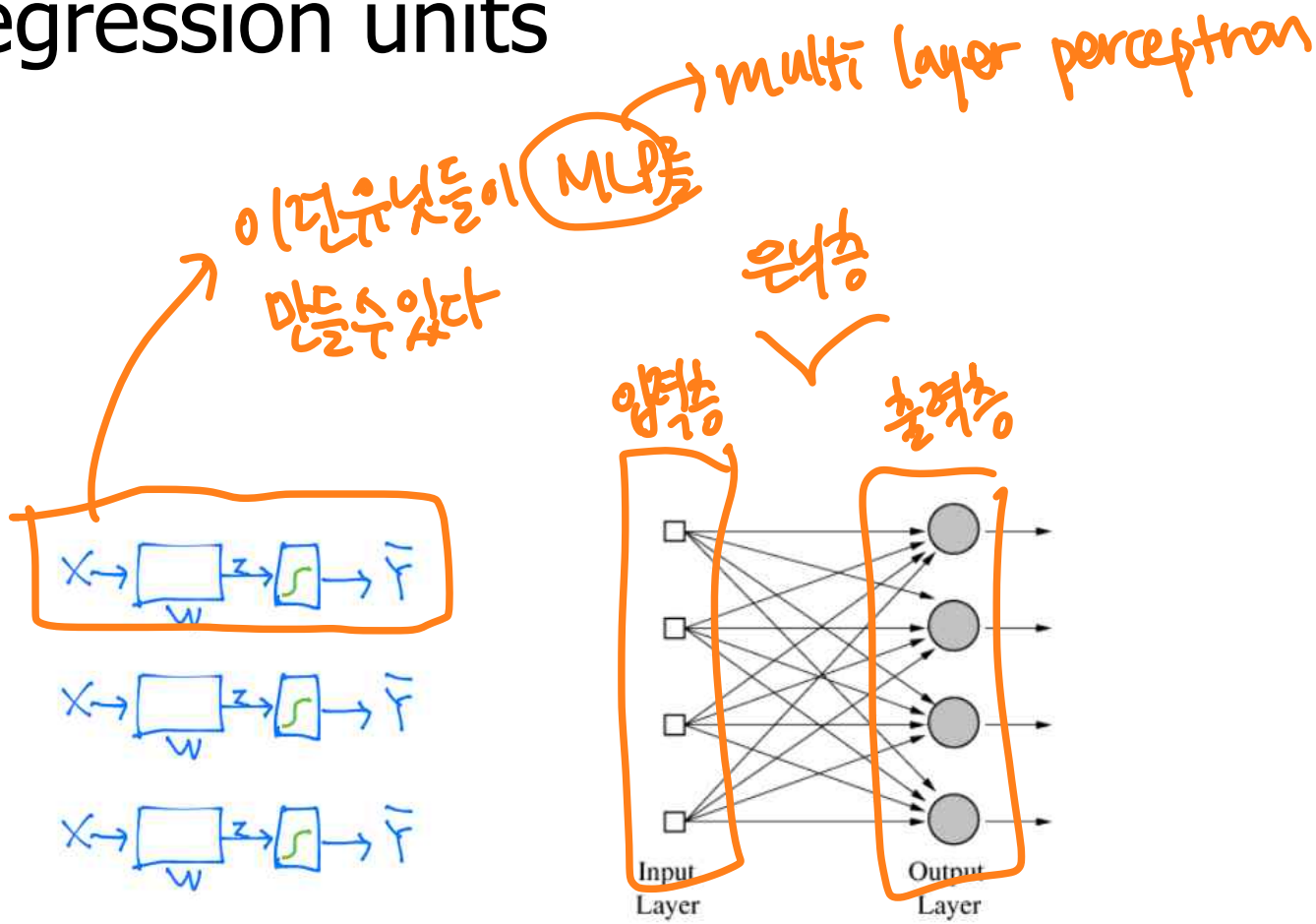
Activation function



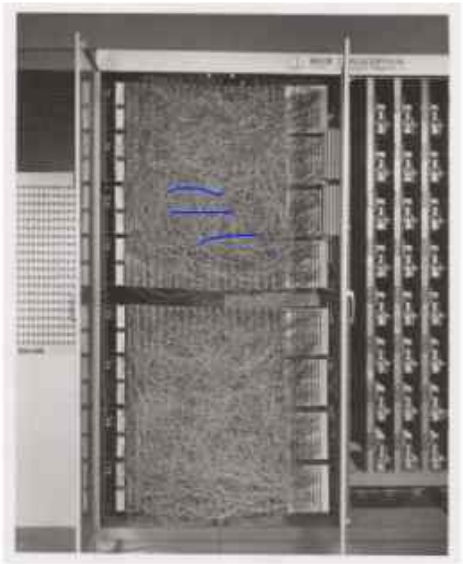
Schematic of a biological neuron.



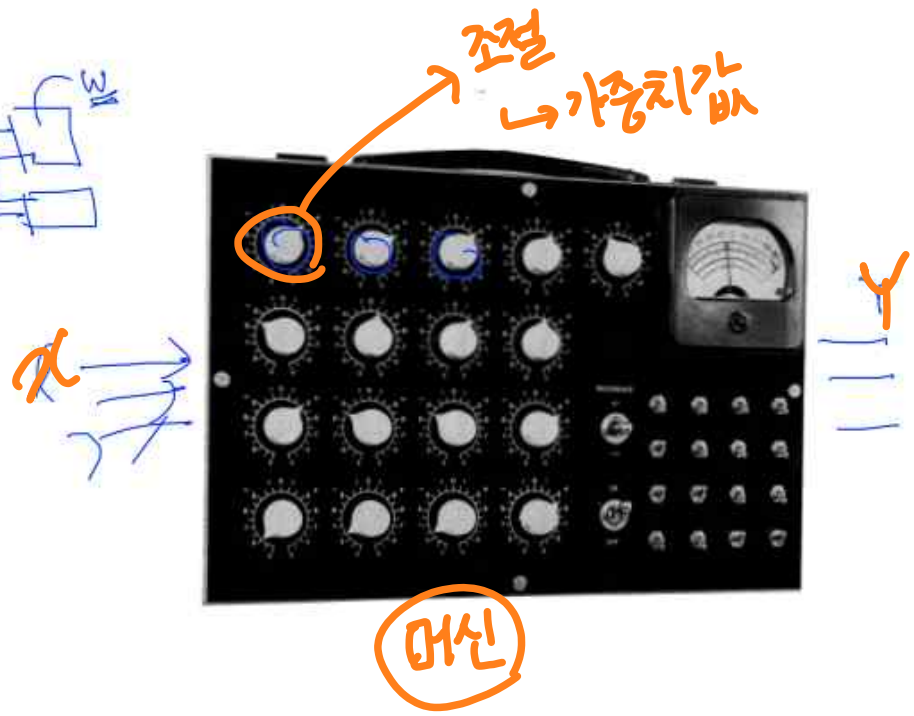
Logistic regression units



Hardware implementations



Frank Rosenblatt, ~1957: Perceptron



Widrow and Hoff, ~1960: Adaline/Madaline

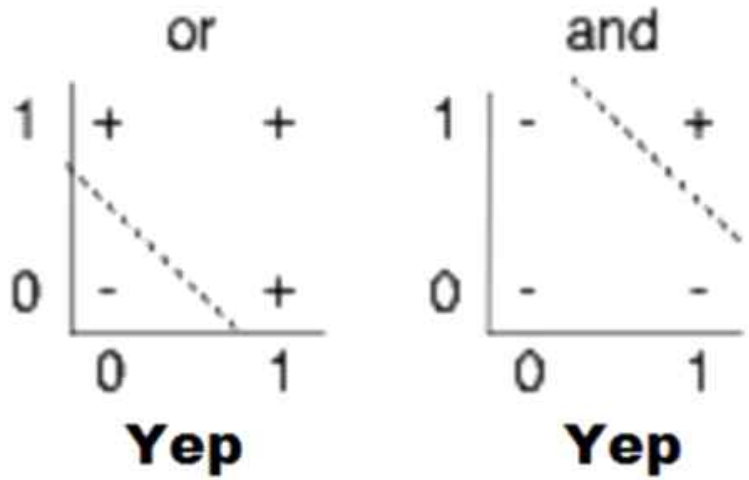
False Promises

"The Navy revealed the embryo of an electronic computer today that *it expects will be able to walk, talk, see, write, reproduce itself* an be conscious of its existence … Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers" The New York Times July 08, 1958

(Simple) AND/OR Problem: linearly separable

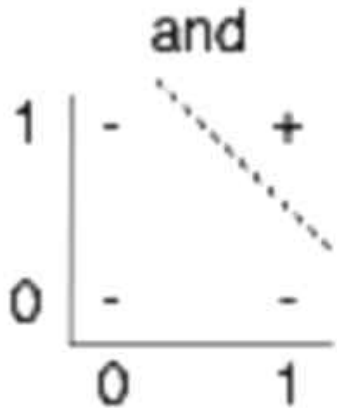
컴퓨터에 필요한 3개 : And / or / xor
논리연산자

선형모델로 해결가능 ⇒

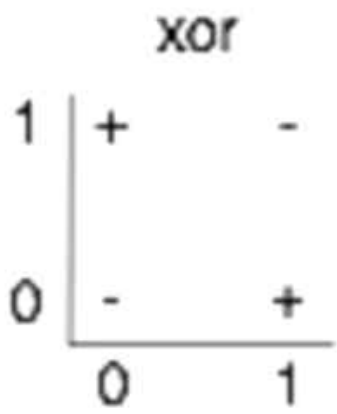


(Simple) XOR Problem: linearly separable?

Yep



Yep

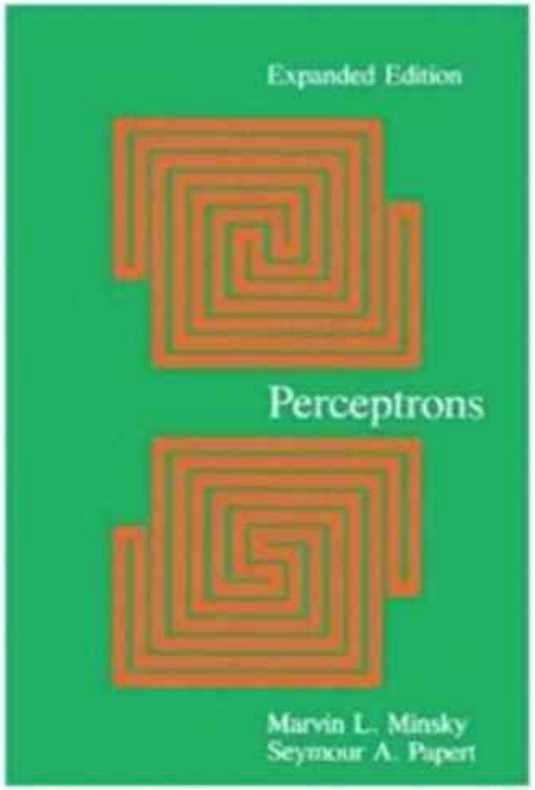


Nope

x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0

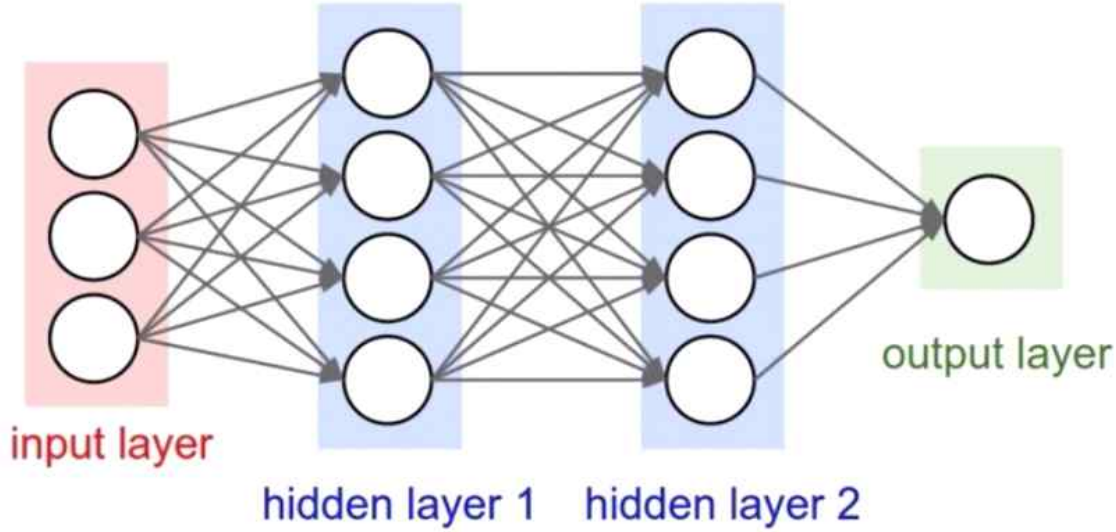
XOR Problem is

- Perceptron(1969) by Marvin Minsky, founder of the MIT AI Lab



- We need to use MLP, multiplayer perceptron (multilayer neural nets)
- No on on earth had found a viable way to train MLPs good enough to learn such simple functions.

No one on earth had found a viable way to train*



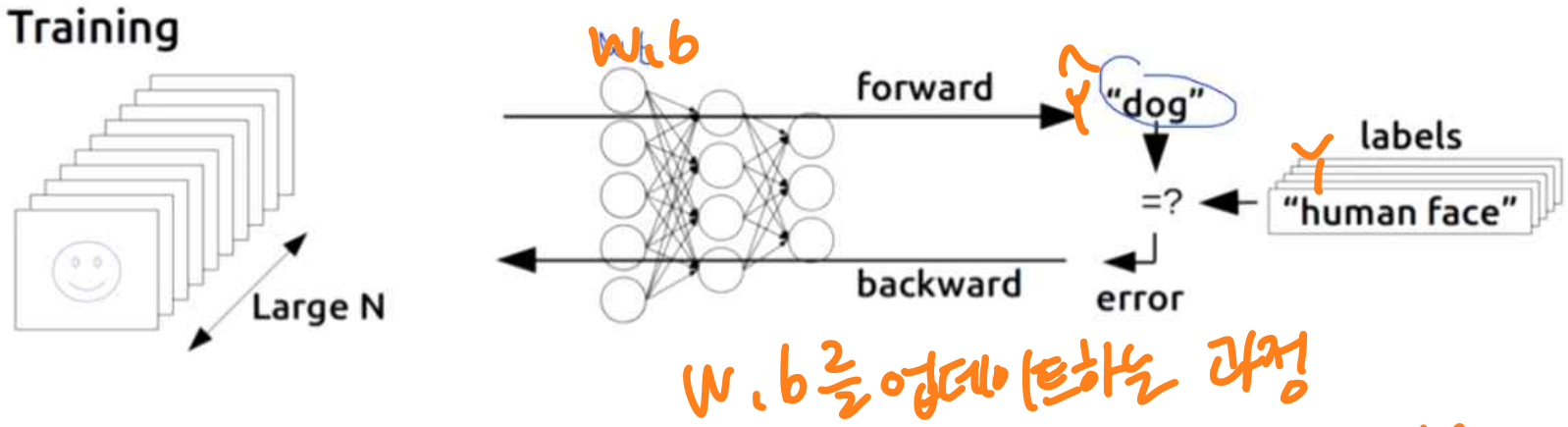
Marvin Minsky, 1969

History of MLP training

역전파

- Backpropagation (1974, 1982 by Paul Werbos, 1986 by Hinton)

역방향으로 오차를 전파시키면서 각층의 가중치를 업데이트하고
최적의 학습결과를 찾아가는 방법 (W,b)



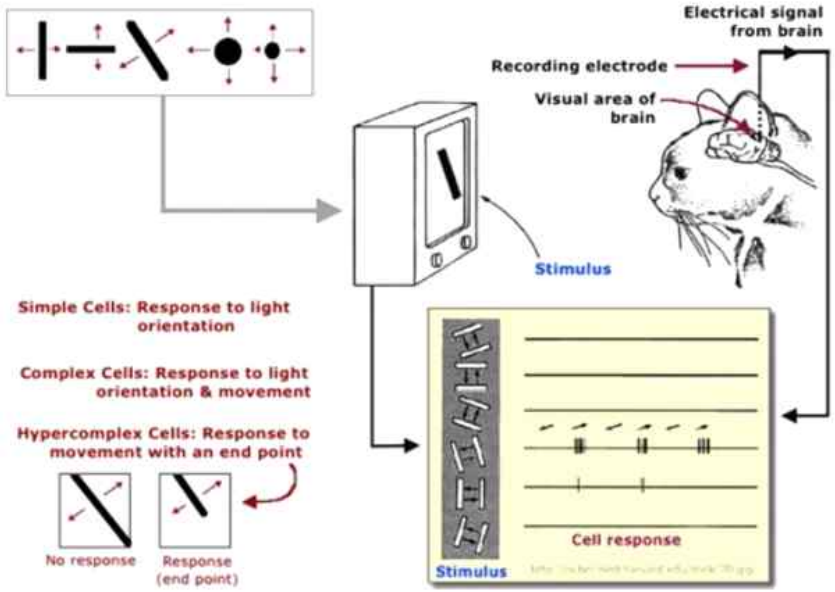
W, b을 업데이트하는 과정

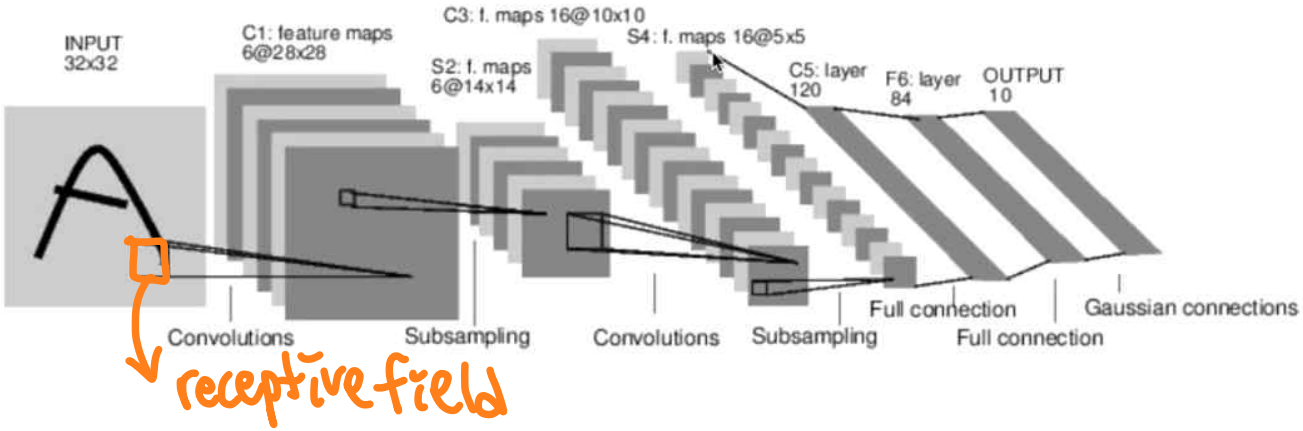
$error = |\hat{Y} - Y|$

Re-discovery: Backpropagation by Hinton

이후로, XOR보다 더 복잡한 문제를 풀기 시작함

Convolutional Neural Networks CNN





LeNet-5, Lecun 1980

Popularity of Neural Network

- Neural Network based Autonomous Car
- NavLab 1984-1994, CMU
  - https://www.youtube.com/watch?v=5-acCtyKf7E
  - Load Detection & Wheel Control

Popularity of Neural Network

- Terminator 2

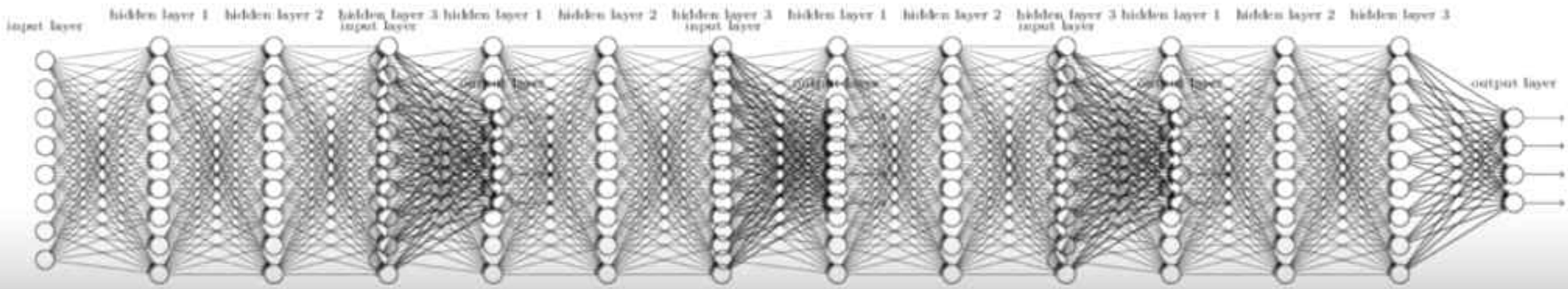
- https://www.youtube.com/watch?v=xcgVztdMrX4

"My CPU is a neural-net processor….a learning computer."

A Big problem

- Backpropagation just did not work well for normal neural nets with many layers ⇒ Deep Neural Net(DNN)
- Other rising machine learning algorithms: SVM, Random Forest, etc.
- 1995 "Comparison of Learning algorithms for Handwritten Digit Recognition" by LeCun et al. found that this new approach worked better.

Gradient Vanishing 문제



Breakthrough

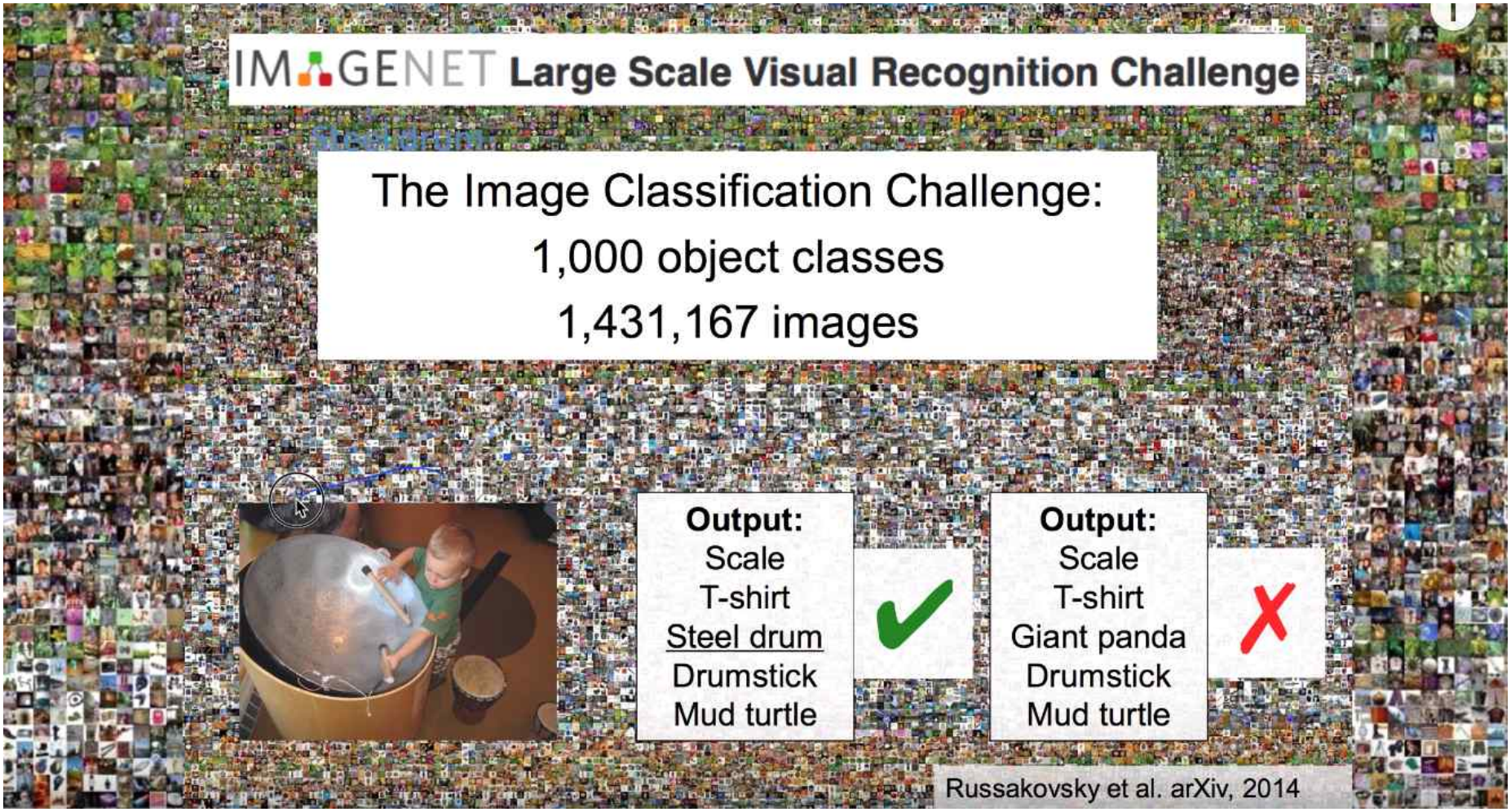
- In 2006 and 2007 by Hinton and Bengio

- (2006) Neural networks with may layers really could be trained well, if the weights are initialized in a clever way rather than randomly.

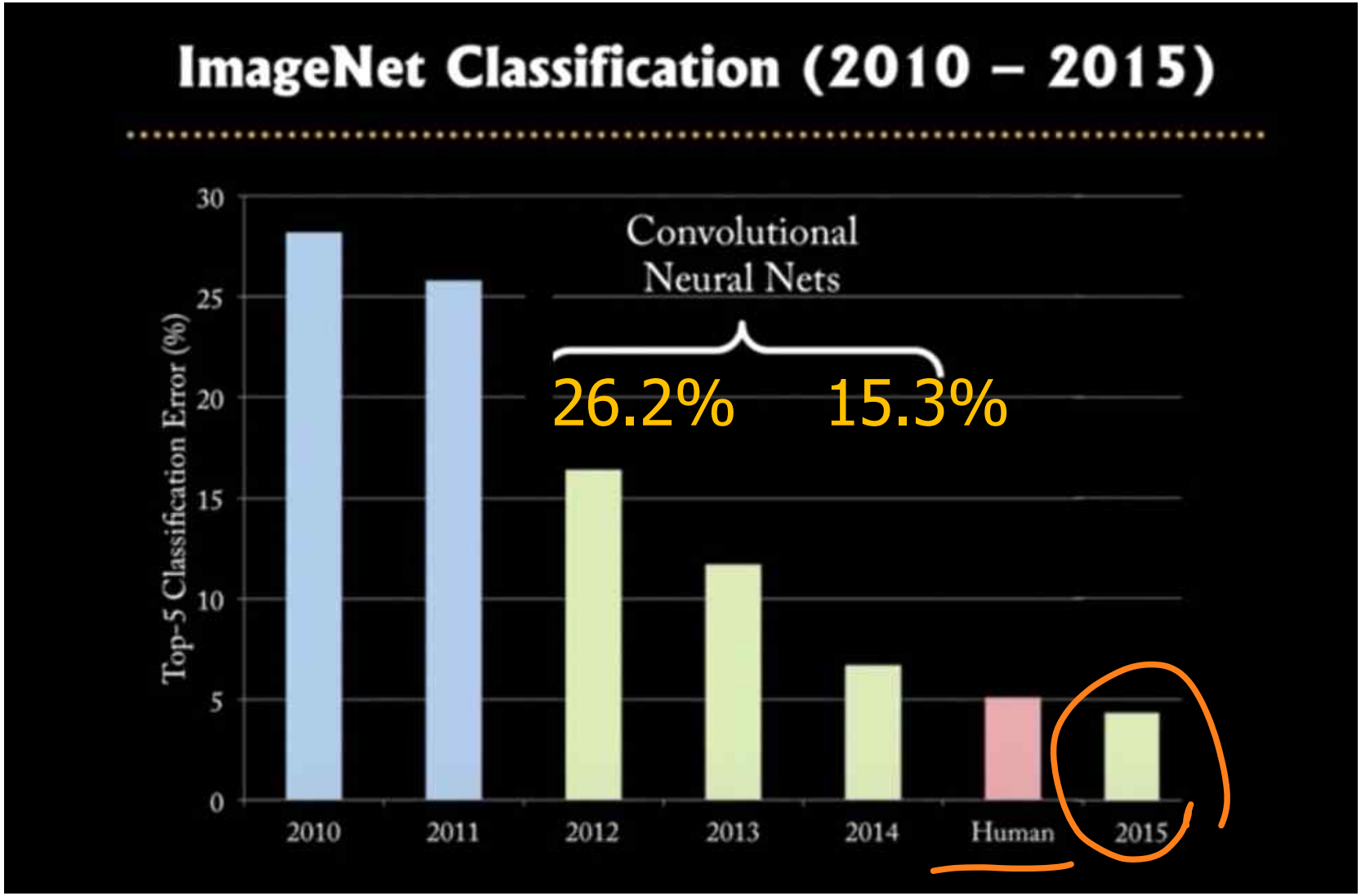
- (2007)Deep machine learning methods are more efficient for difficult problems than shallow methods

- Rebranding to **Deep Nets, Deep Learning**

Breakthrough

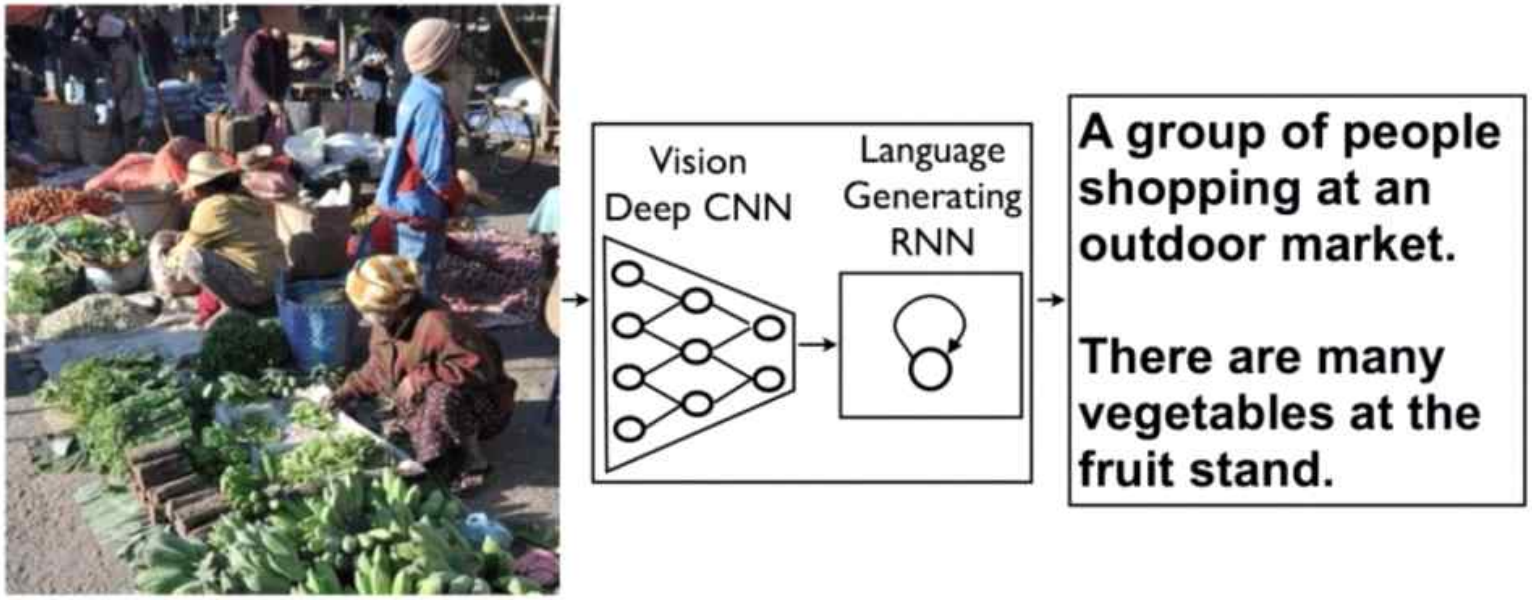


Breakthrough



After Breakthrough

- Neural Networks that can explain Photos



After Breakthrough

- Deep API Learning

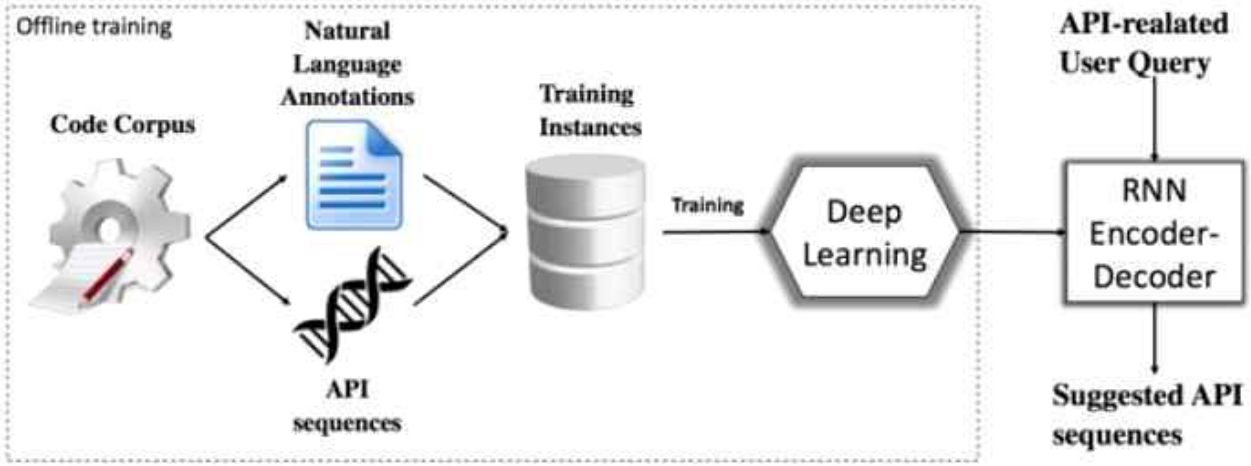


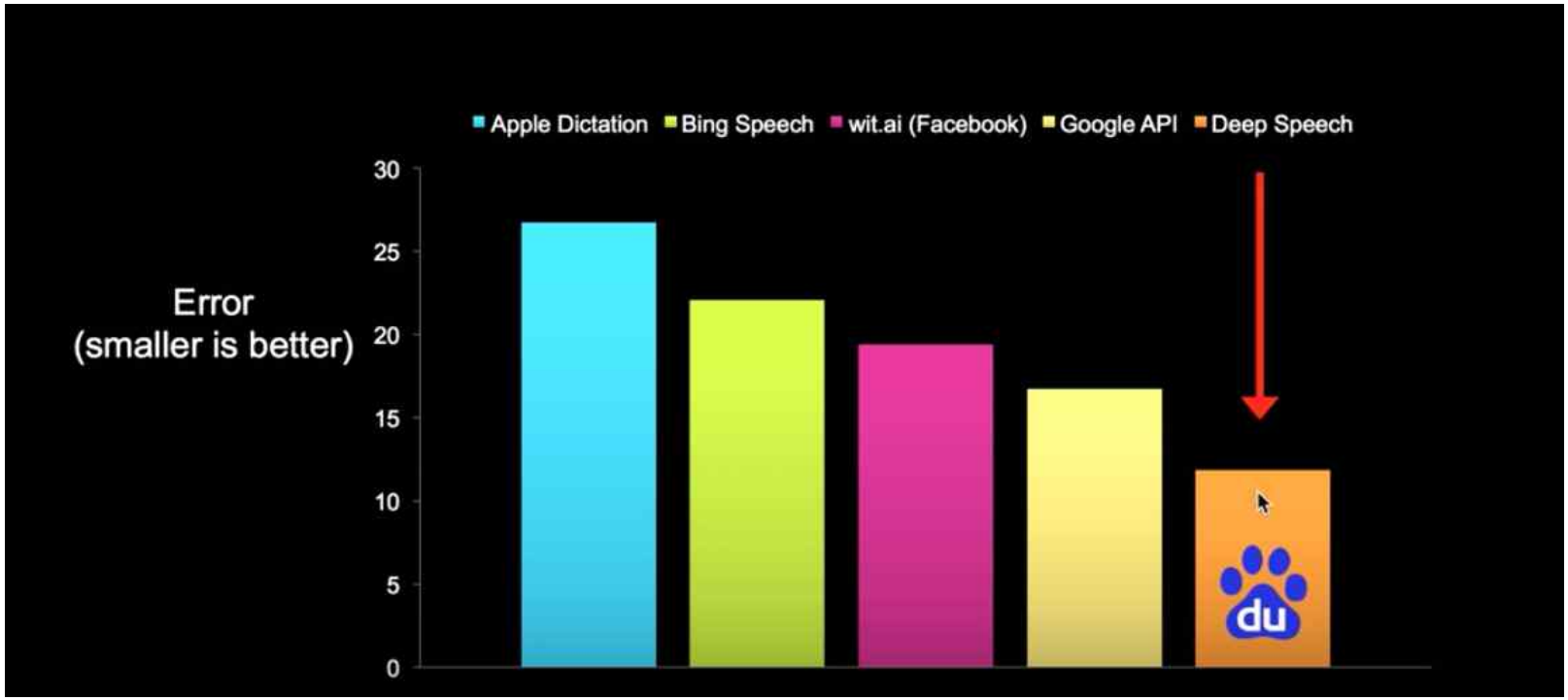
Figure 3: The Overall Workflow of DEEPAPI

"Copy a file and save it to your destination path"

20% 60%

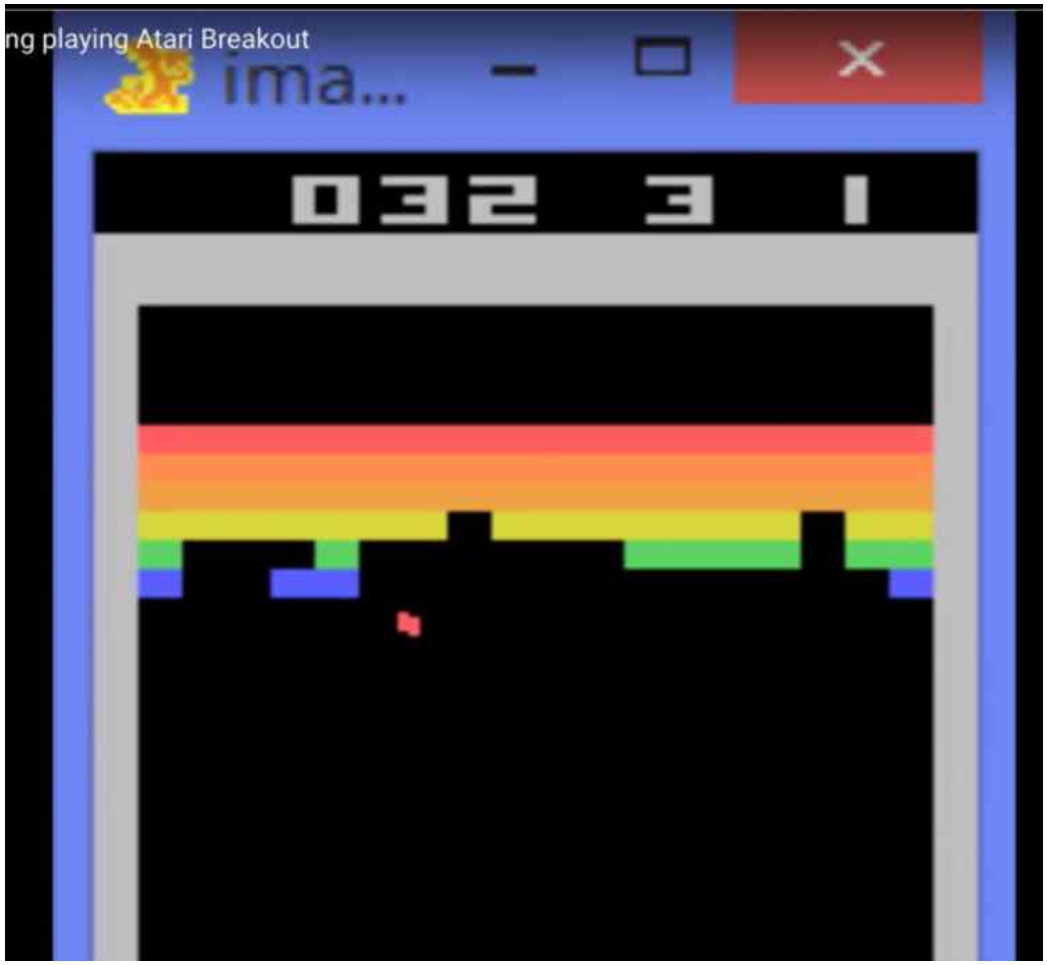
After Breakthrough

- Speech recognition errors



After Breakthrough

- Altari



Geoffrey Hinton's summary of findings up to today

정답

- Our labeled datasets were thousands of times too small
- Our computers were millions of times too slow
- We initialized the weights in a stupid way
- We used the wrong type of non-linearity

Artificial Intelligence

Yukyung Choi

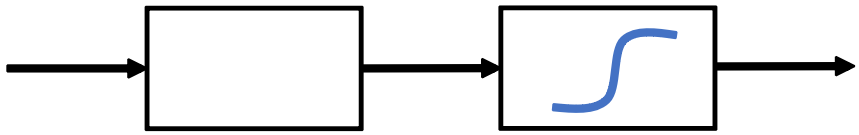
Agenda

- History of MLP

- Neural Net(NN)
  - Perceptron
  - Multi-Layer Perceptron (MLP)
  - Backpropagation

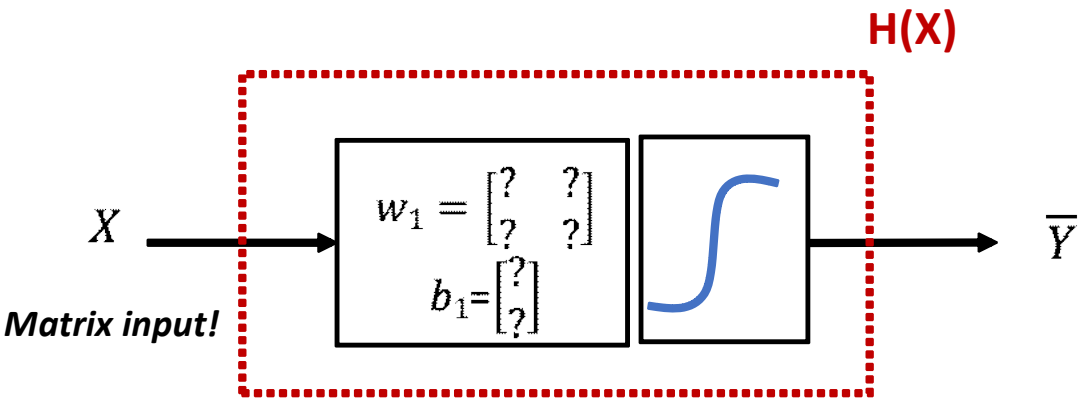
Problem: Perceptron → 입력층과 출력층 제외 은닉층이 1개

- One logistic regression unit cannot separate XOR



X w z $\overline{Y}$

[실습] Perceptron for XOR



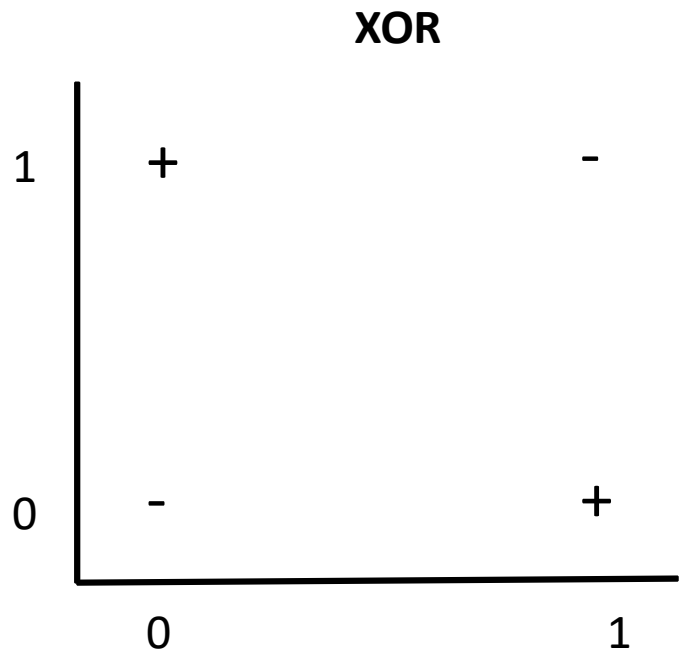
$$\overline{Y} = H(X) = \text{sigmoid}(X \cdot w_1 + b_1)$$

X_1	X_2	$\overline{Y}$
0	0	0
0	1	1
1	0	1
1	1	0

Problem: Perceptron

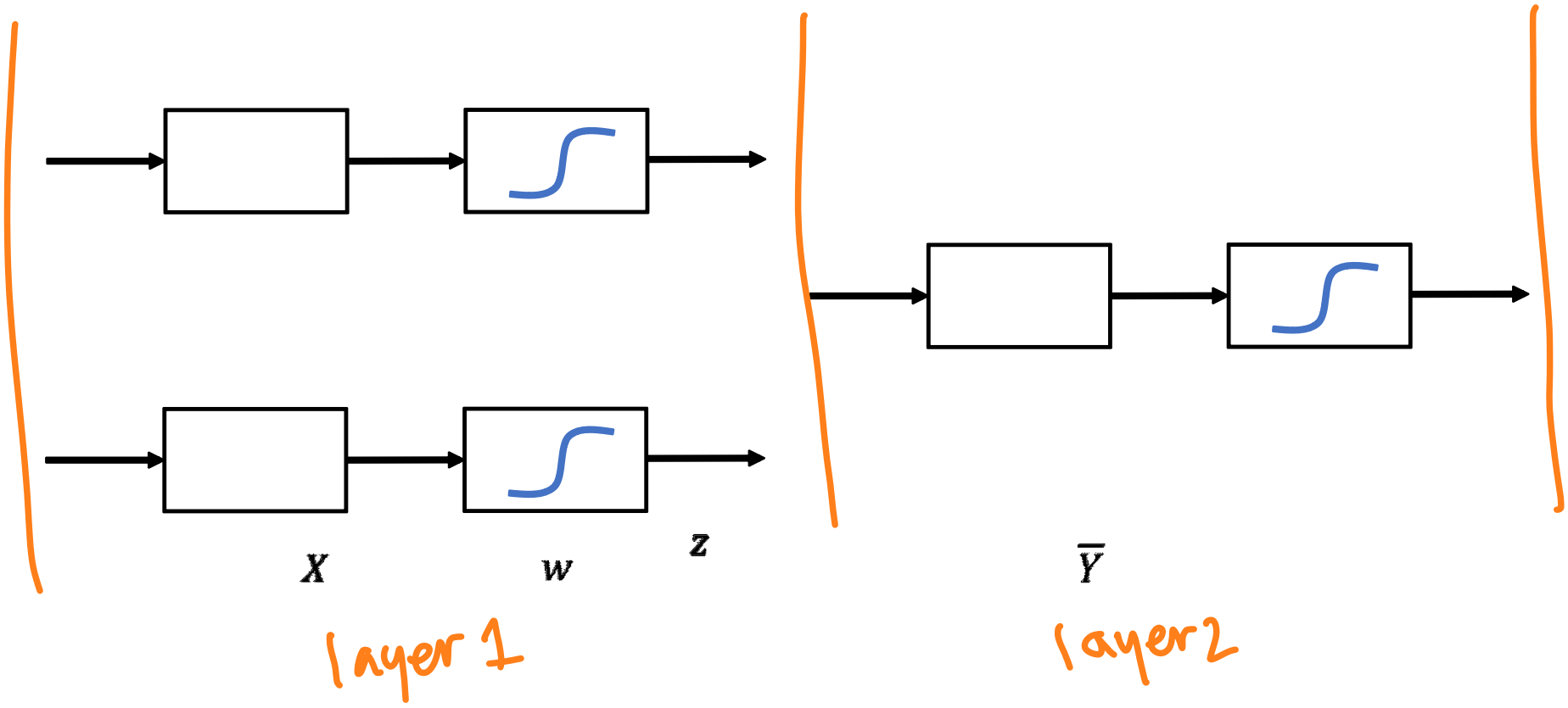
- XOR using NN

X1	X2	XOR	
0	0	0	-
0	1	1	+
1	0	1	+
1	1	0	-



Solution: MLP

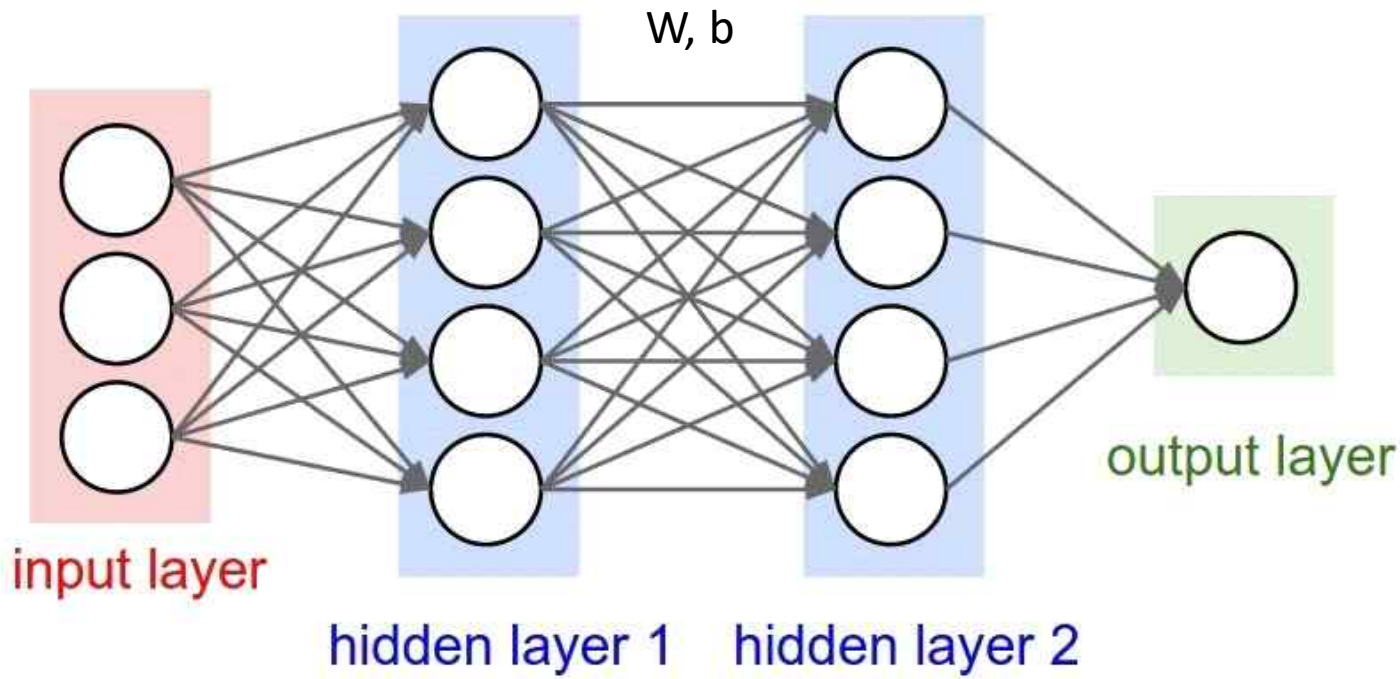
- Multiple logistic regression units can separate XOR



Solution: MLP

은닉층2개이상이면

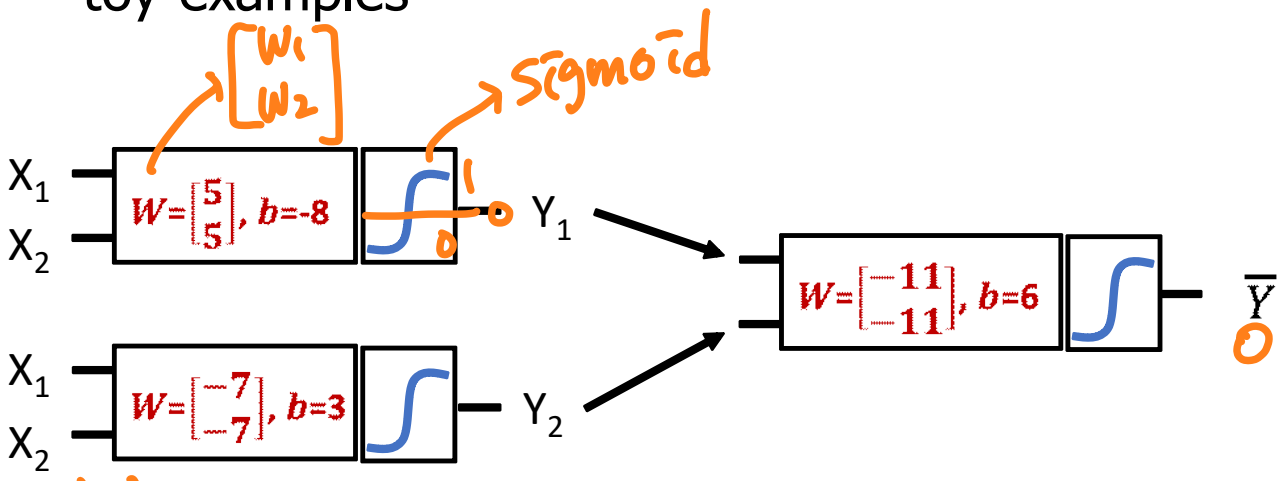
- Multiple logistic regression units can separate XOR
- But! "No one on earth had found a viable way to train"



*Marvin Minsky

Solution: MLP

- XOR using NNs
  - toy-examples



(0,0) 일 때

$Y_1 = X_1 \cdot W_1 + X_2 W_2 + b = -8 \rightarrow 0$

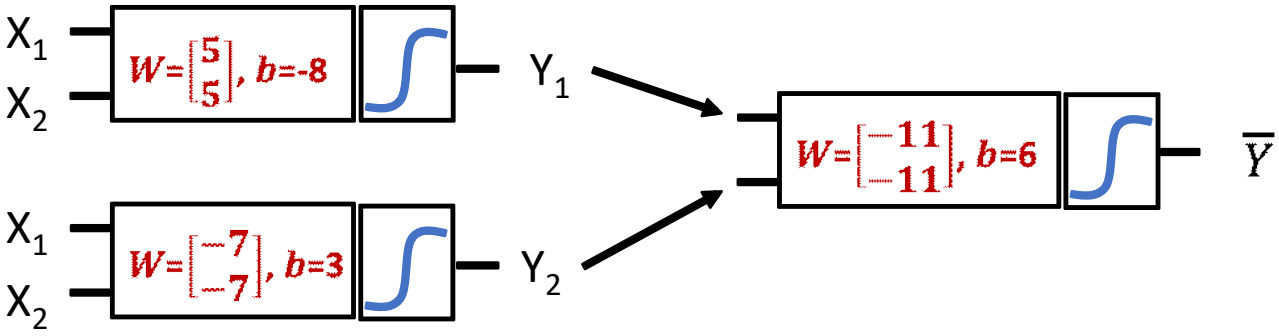
$Y_2 = X_1 \cdot W_1 + X_2 \cdot W_2 + b = 3 \rightarrow 1$

$\bar{Y} = Y_1 \cdot W_1 + Y_2 W_2 + b = -5 \rightarrow 0$

X_1	X_2	Y_1	Y_2	Y_1 Y_2 $\bar{Y}$	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0

Solution: MLP

- XOR using NNs

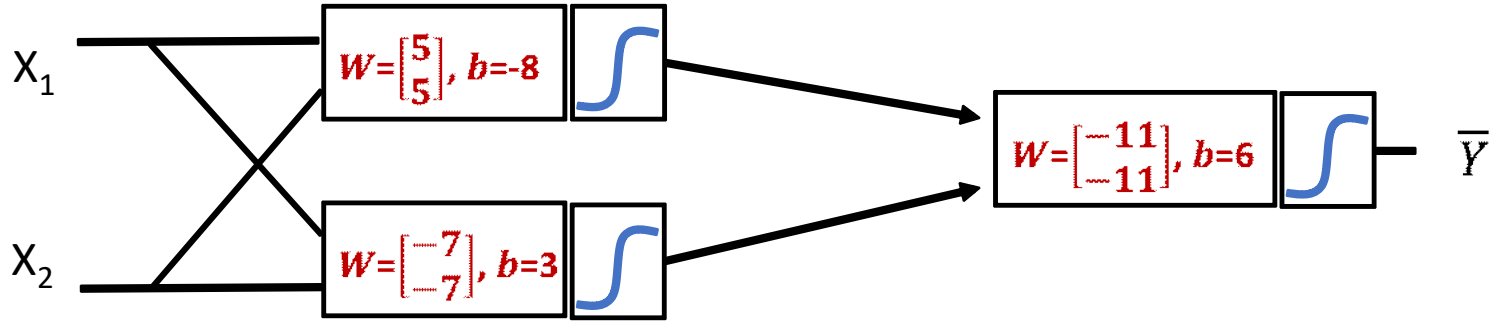


X_1	X_2	Y_1	Y_2	Ŷ (Y1 Y2: 0 1)	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0

Solution: MLP

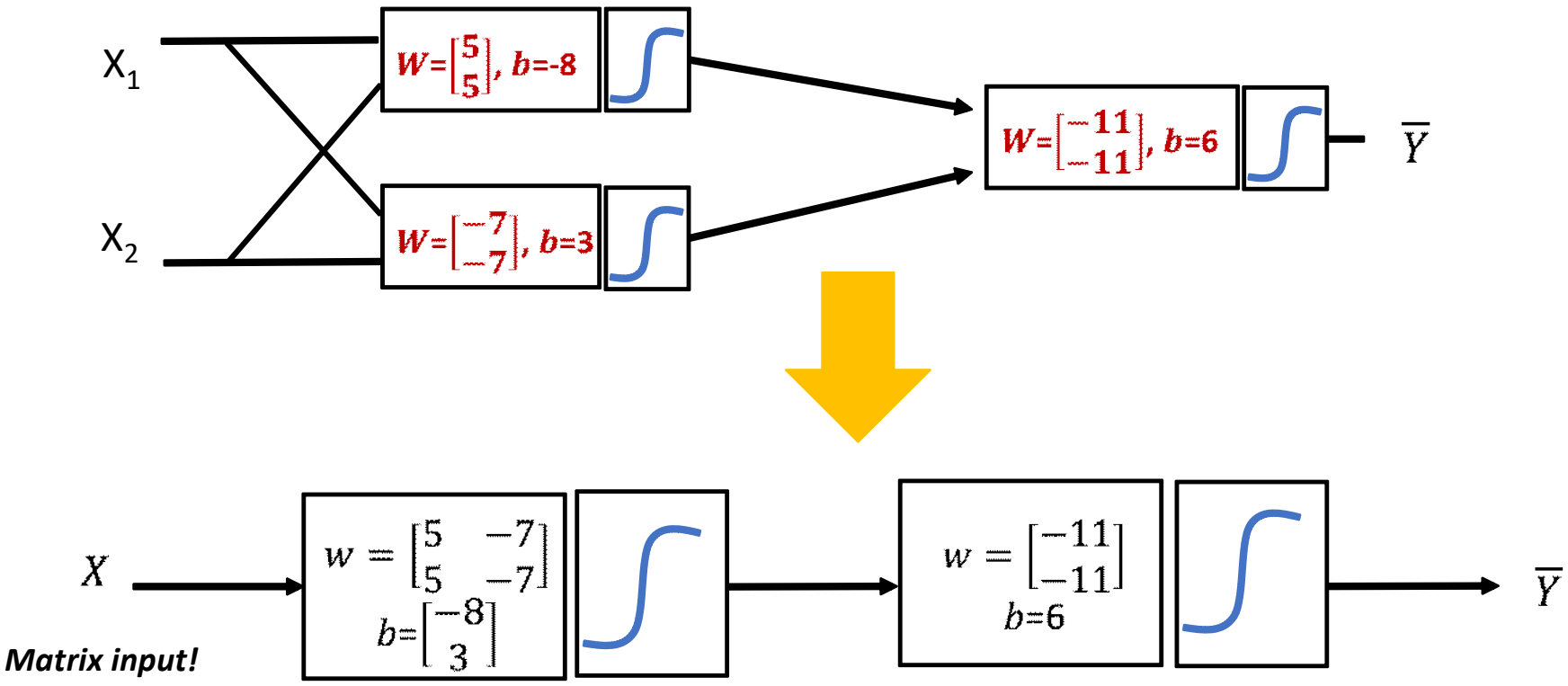
- Forward Propagation을 통한 XOR 문제 풀이 가능 검증 완료

= FeedForward

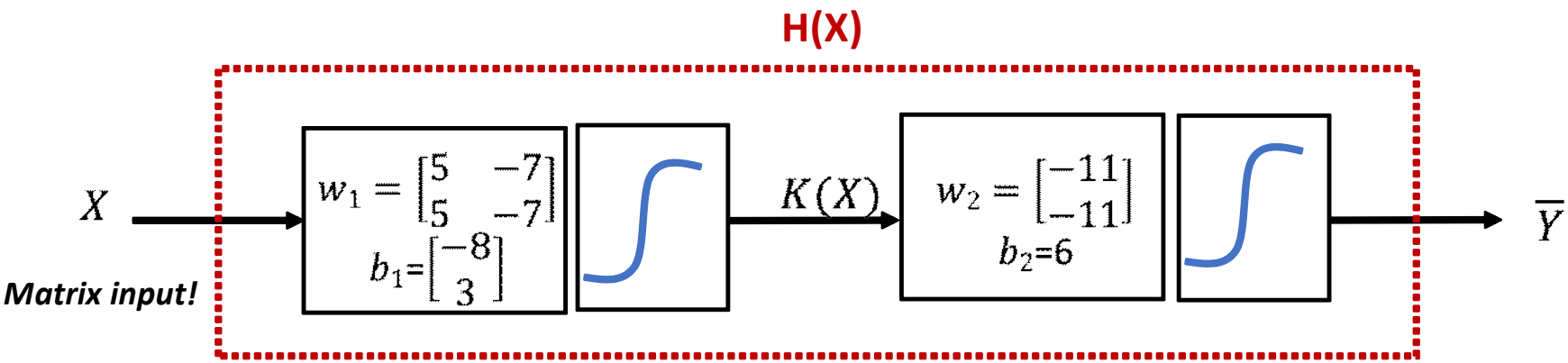


Solution: MLP

- Forward Propagation을 통한 XOR 문제 풀이 가능 검증 완료



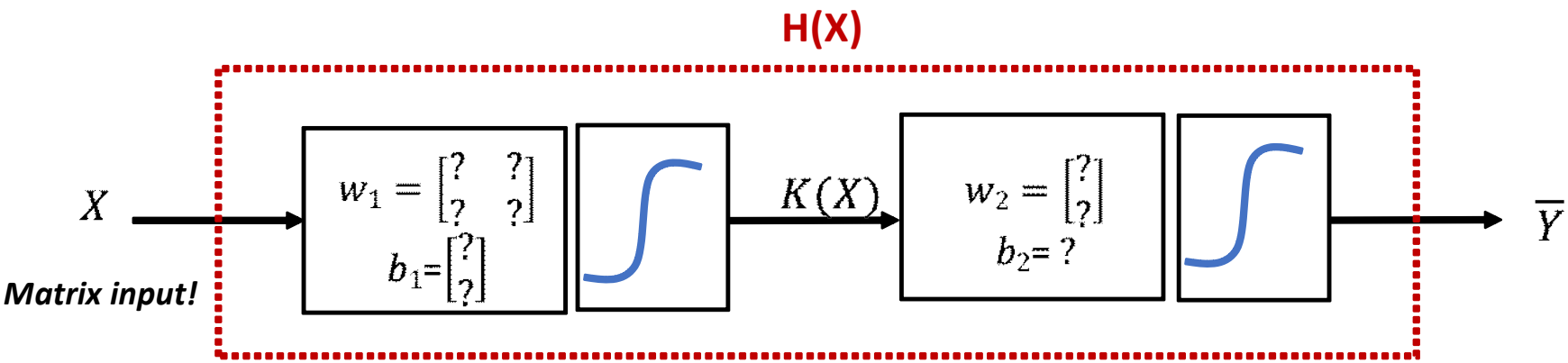
Solution: MLP for XOR



$$K(X) = \text{sigmoid}(X \cdot W_1 + b_1)$$

$$\overline{Y} = H(X) = \text{sigmoid}(K(x) \cdot w_2 + b_2)$$

How can we learn W1, W2, B1, B2 from training data?



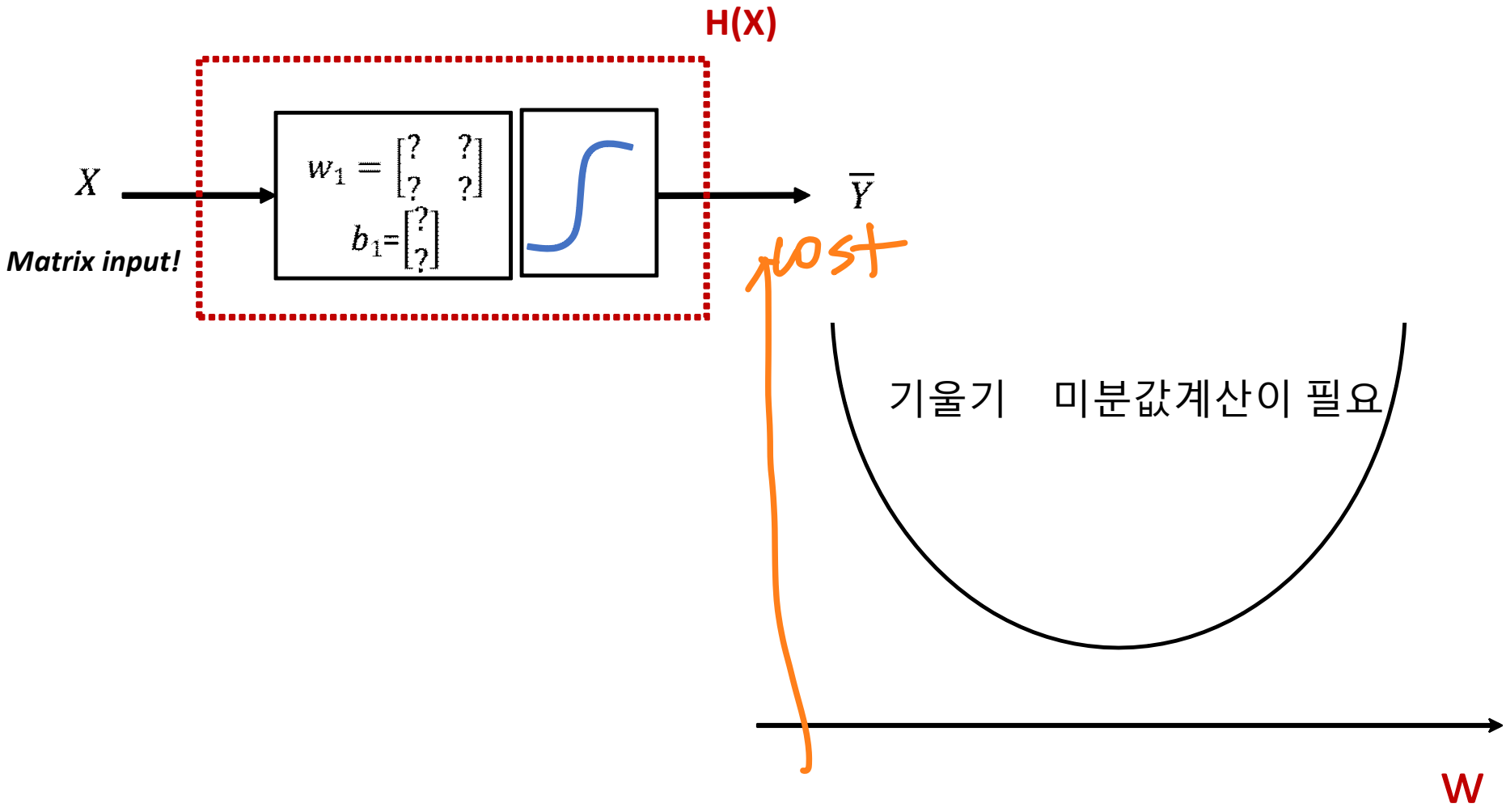
$$K(X) = \text{sigmoid}(X \cdot W_1 + b_1)$$

$$\overline{Y} = H(X) = \text{sigmoid}(K(x) \cdot w_2 + b_2)$$

X_1	X_2	$\overline{Y}$
0	0	0
0	1	1
1	0	1
1	1	0

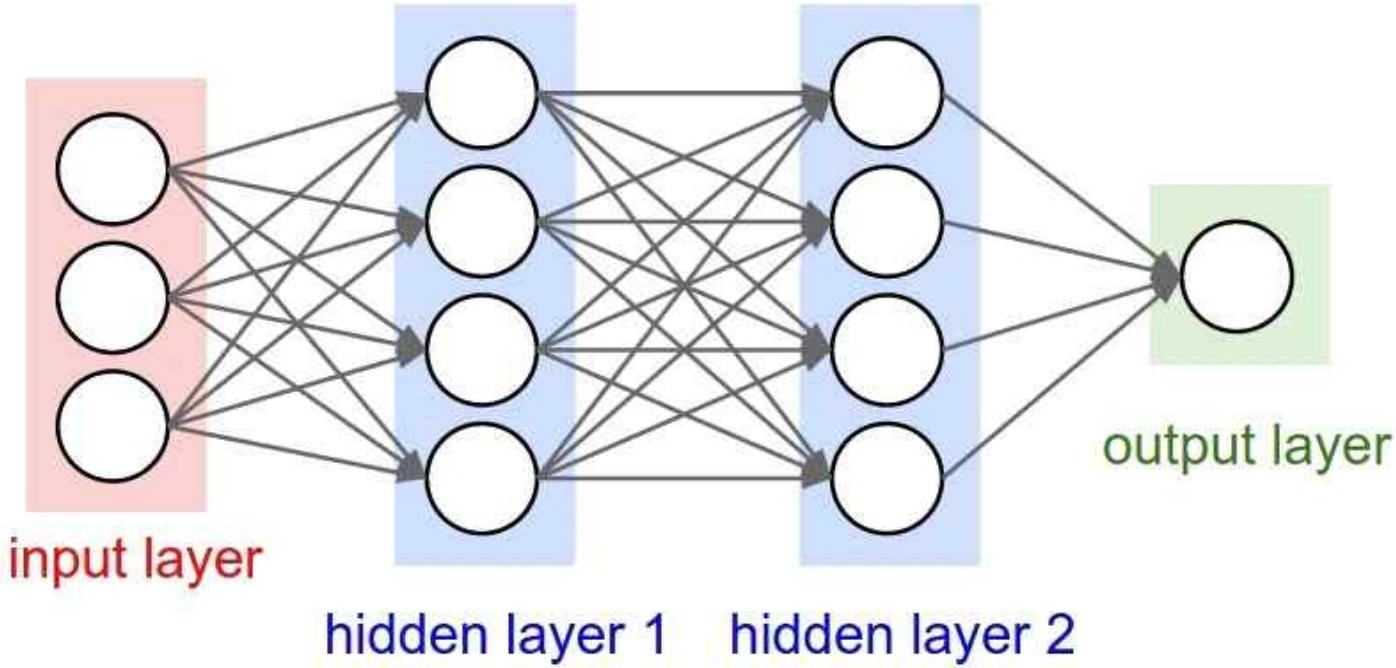
Remind

- We try to find a minimum value using a gradient descent method.



Remind

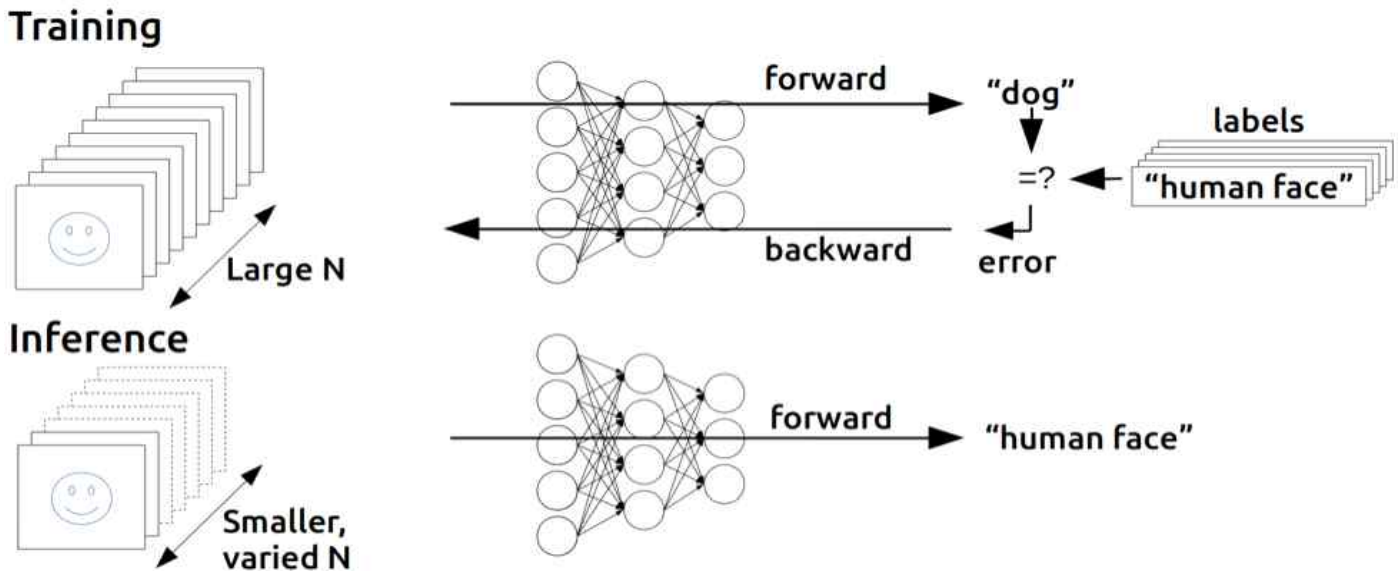
- But, **MLP** is too complex to apply gradient descent method for finding optimal parameters.



입력과 출력에 영향을 미치는 각각의 W, b 값을 구하는게 어렵다.

Remind

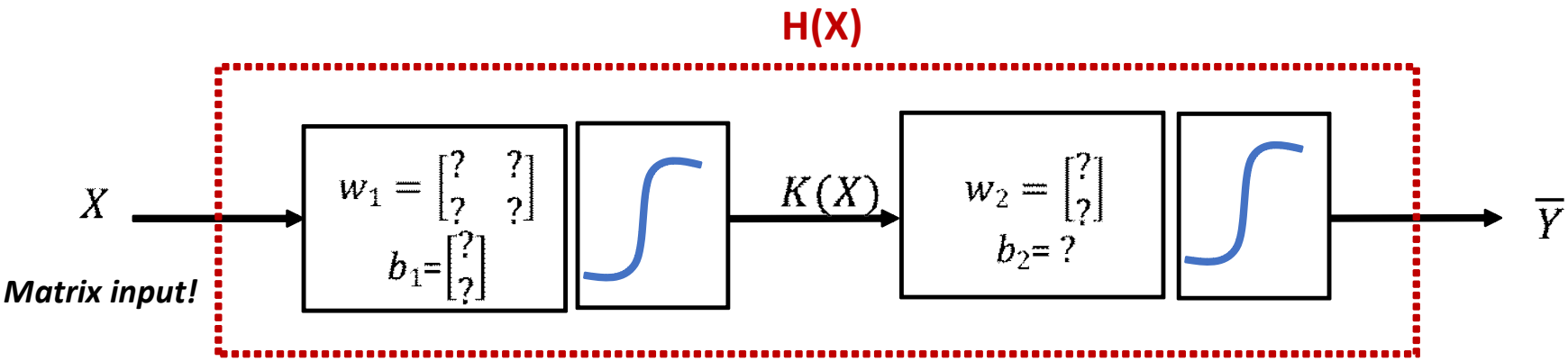
- **Backpropagation** (1974, 1982 by Paul Werbos, 1986 by Hinton)



1) W, b 초기화
2) Forward 계산 & Error 측정
3) **Error 값 을 Backward 하면서 w, b 업데이트**

[실습] MLP for XOR

How can we learn W1, W2, B1, B2 from training data?



$$K(X) = \text{sigmoid}(X \cdot W_1 + b_1)$$

$$\overline{Y} = H(X) = \text{sigmoid}(K(x) \cdot w_2 + b_2)$$

X_1	X_2	$\overline{Y}$
0	0	0
0	1	1
1	0	1
1	1	0

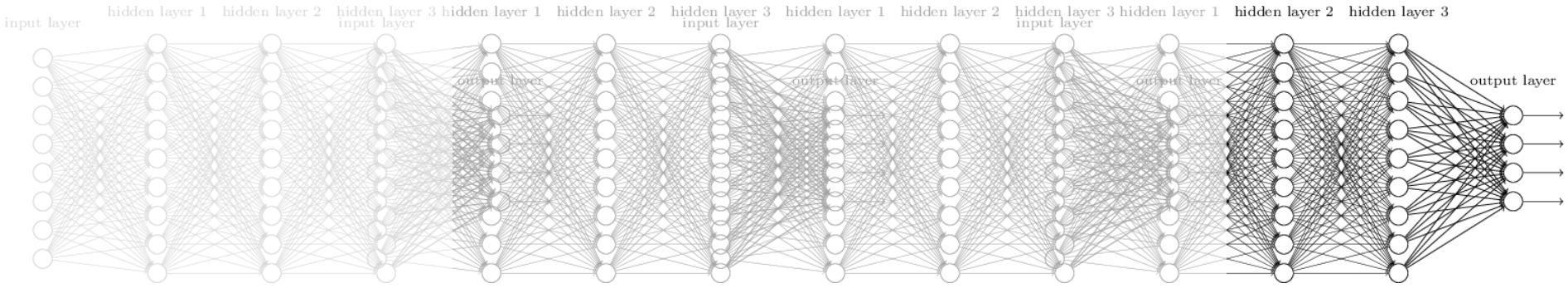
Deep Neural Network (DNN) 잘하기 편

Artificial Intelligence

Yukyung Choi

Problem

① Neural Network(NN) 을 이용한 XOR 문제 해결
② Deep Neural Network (DNN) 을 통한 어려운 문제 도전
③ DNN에서 Gradient Vanishing 문제 발생



Geoffrey Hinton's summary of findings up to today

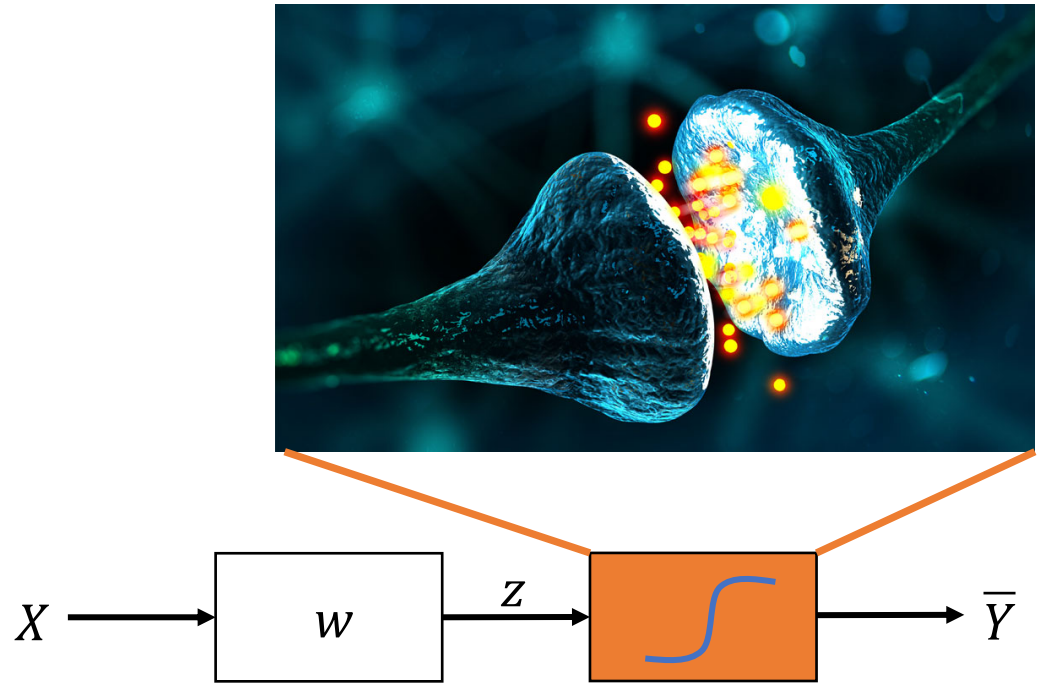
➔ DNN에서 Gradient Vanishing 문제 해결법

- Our labeled datasets were thousands of times too small
- Our computers were millions of times too slow
- We initialized the weights in a stupid way
- We used the wrong type of non-linearity

Activation Function

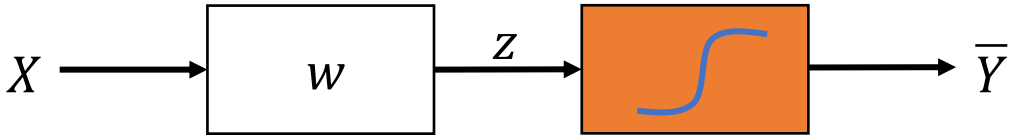
Activation Function

- 활성화 함수(Activation Function)는 신경학적으로 볼 때 뉴런 발사 (Firing of a Neuron)의 과정에 해당함
- 최종 출력 신호를 다음 뉴런으로 보내줄지 말지 결정하는 역할을 함



Activation Function

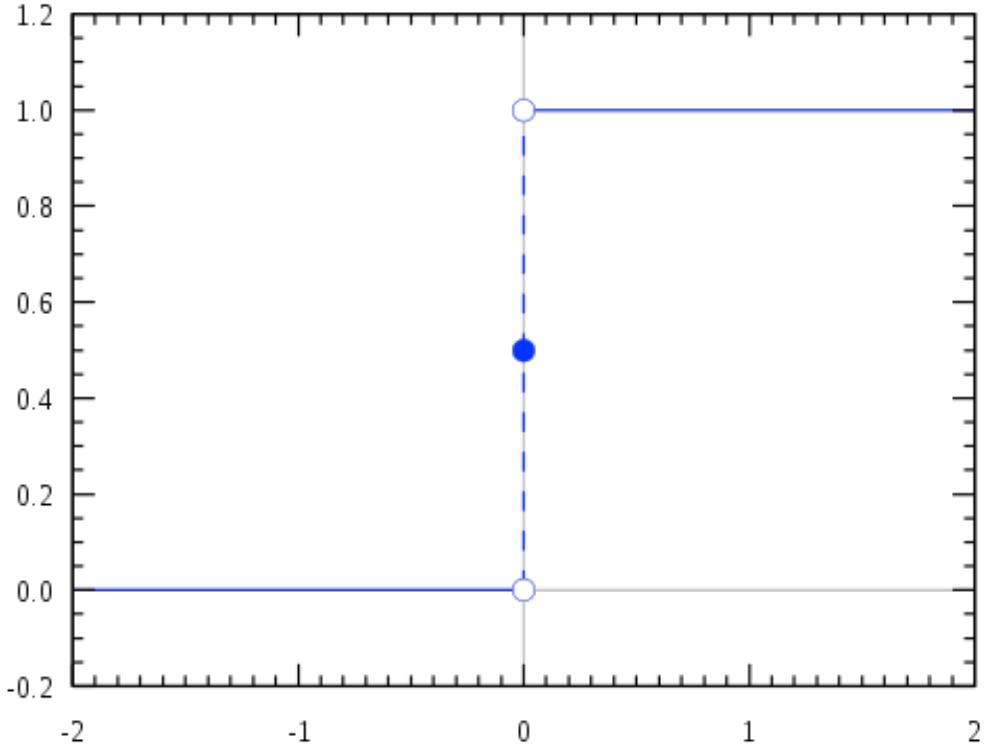
- 뉴런이 다음 뉴런으로 신호를 보낼 때 입력신호가 일정 기준 이상이면 보내고 기준에 달하지 못하면 보내지 않을 수도 있음. 즉, 활성 함수란 그 신호를 결정 해주는 것
- 많은 종류의 활성화 함수가 있고, Activation function의 결정이 결과에 크게 영향을 미침



sigmoid { 0.5↑ → 1
0.5↓ → 0

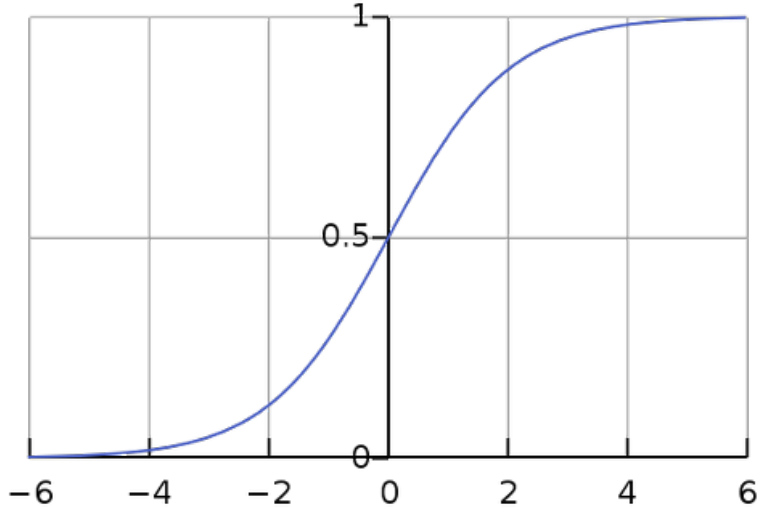
Activation Function: Step

- 입력이 양수 일 때는 1, 음수 일 때는 0 의 신호를 보내주는 이진 함수
- 미분 불가능한 함수로 모델 Optimization과정에 사용이 어려워 신경망의 활성 함수로 사용하지 않음



Activation Function: Sigmoid

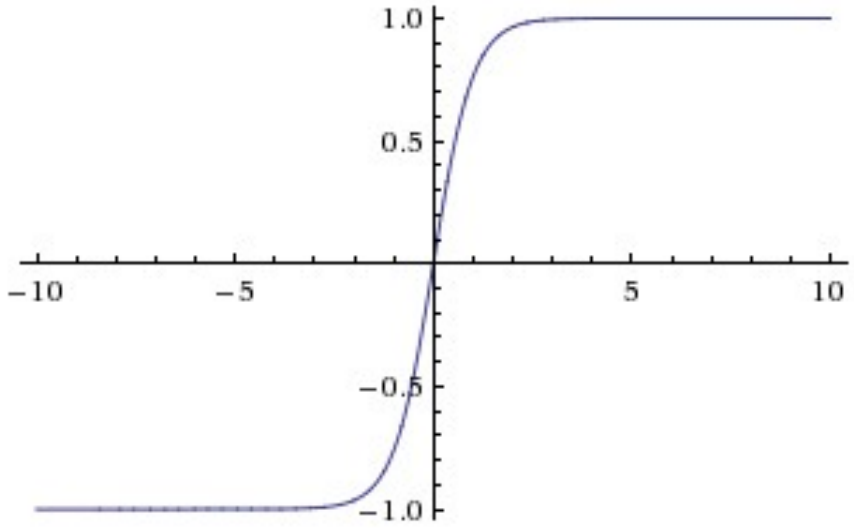
- 단일 퍼셉트론(perceptron)에서 사용했던 활성 함수
- 입력을 (0,1) 사이로 정규화(normalization) 함
- Backpropagation 단계에서 NN layer 를 거칠 때마다 작은 미분 값이 곱해져, Gradient Vanishing 을 야기함. 여러 개의 Layer를 쌓으면 신경망 학습이 잘 되지 않는 원인
- Deep Layer (3개 이상)에서 활성 함수로 사용을 권하지 않음



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

Activation Function: tanh

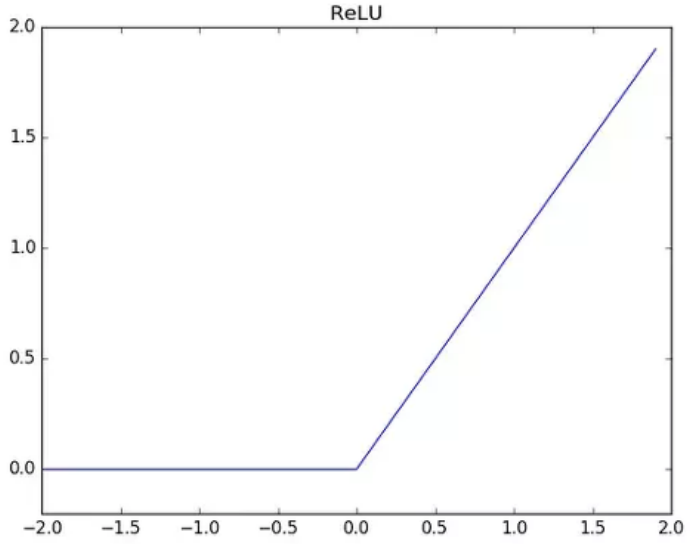
- Sigmoid를 보완하고자 제안된 활성 함수
- 입력을 (-1, 1)사이의 값으로 정규화(normalization) 함
- Sigmoid 보다 tanh 함수가 전반적으로 성능이 좋음
- 여전히 Gradient Vanishing 문제는 발생함 (Sigmoid 보다는 덜 발생함)



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

Activation Function: ReLU (Rectified Linear Unit)

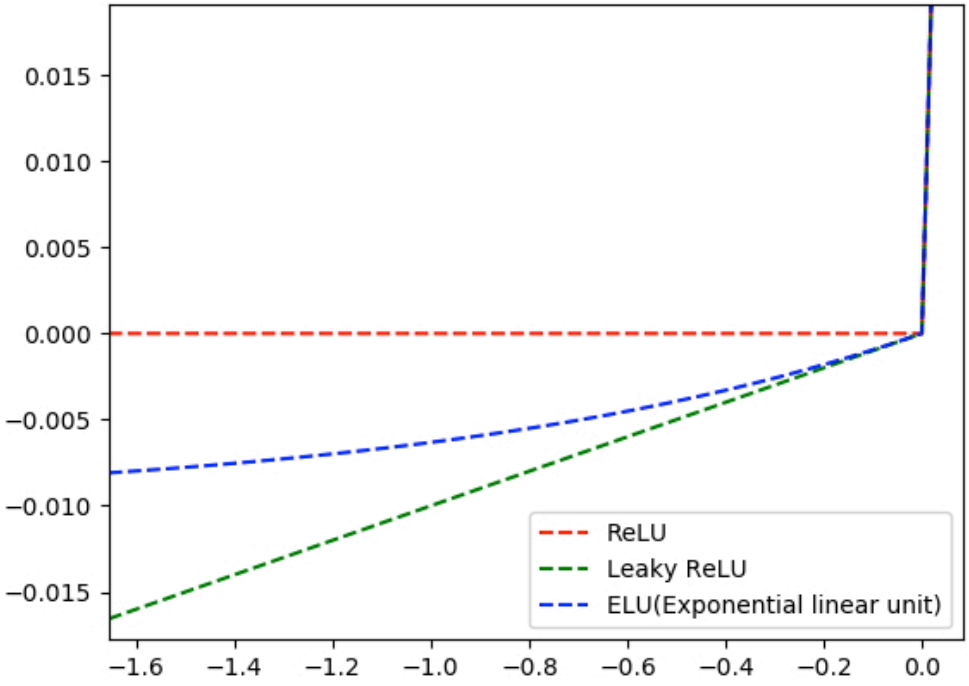
- 현재 가장 인기있는 활성화 함수
- 양수에서 Linear Function과 같으며 음수는 0을 출력하는 함수
- 미분 값을 0 또는 1의 값을 가지기 때문에 Gradient Vanishing 문제가 발생하지 않음
- Linear Function과 같은 문제는 발생하지 않으며, 엄연히 Non-Linear함수 이므로 Layer를 deep하게 쌓을 수 있음.
- exp() 함수를 실행하지 않아 sigmoid함수나 tanh함수보다 6배 정도 빠르게 학습이 진행됨



$$Relu(x)=max(0,x)$$

Activation Function: Leaky ReLU

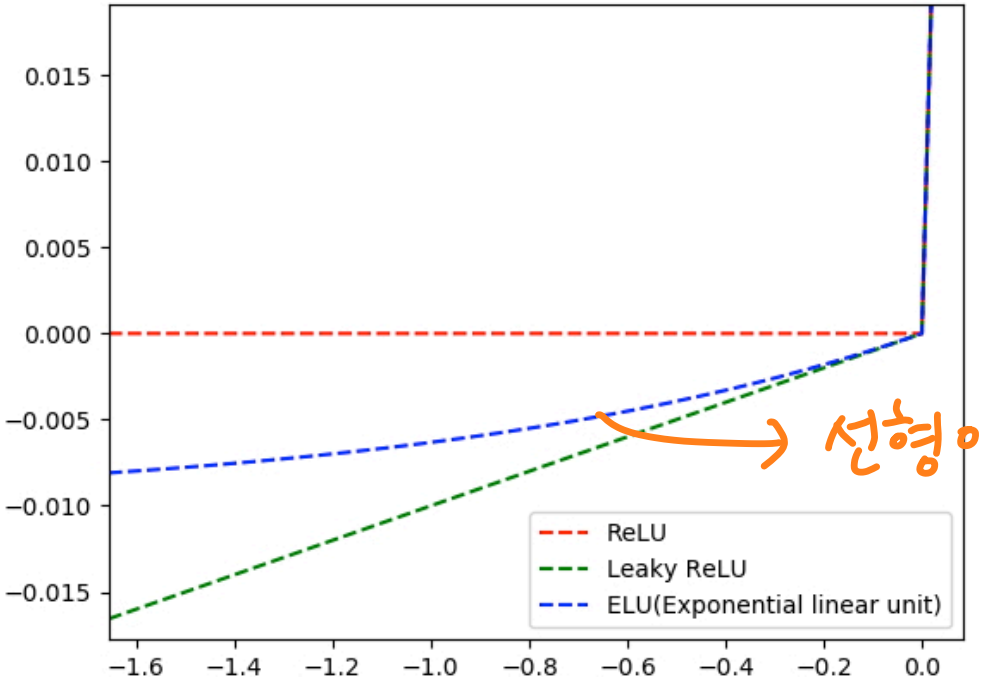
- Leaky ReLU는 "dying ReLU" 현상을 해결하기 위해 제시된 함수
- ReLU는 x<0인 경우 함수 값이 0이지만, Leaky ReLU는 작은 기울기를 부여함
- 보통 작은 기울기는 0.01을 사용함
- Leaky ReLU로 성능향상이 발생했다는 보고가 있으나 항상 그렇지는 않음



*Leaky Relu(x)=max(0.01*x,x)*

Activation Function: ELU (Exponential Linear Units)

- ReLU의 threshold를 -1로 낮춘 함수를 exp^x 를 이용하여 근사한 것
- dying ReLU 문제를 해결함
- 출력 값이 거의 zero-centered에 가까움
- 하지만 ReLU, Leaky ReLU와 달리 exp()를 계산해야하는 비용이 비듬

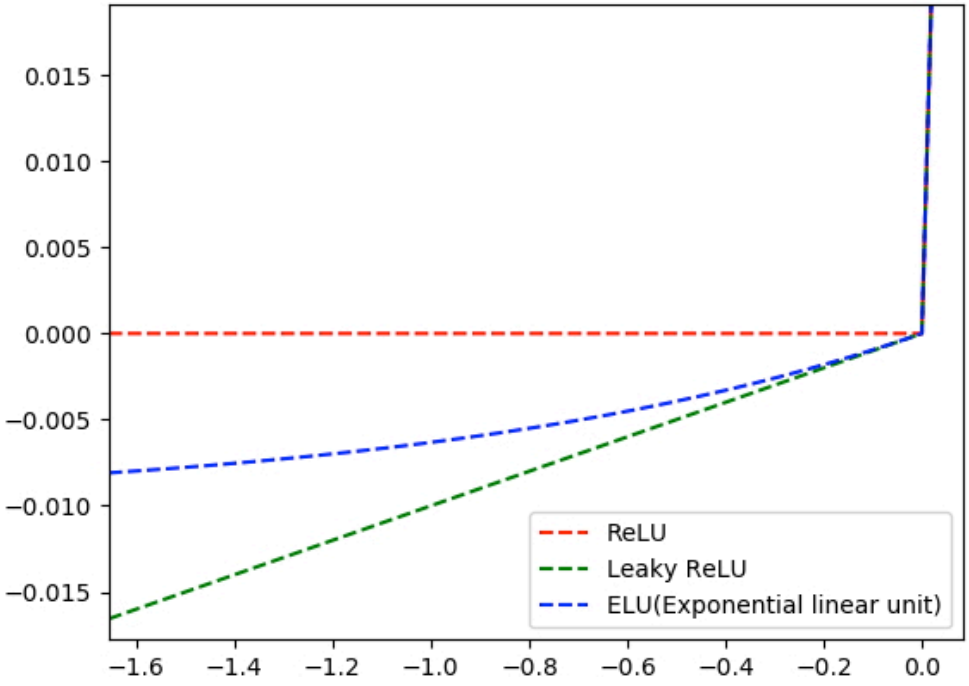


선형아님.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

Activation Function: Maxout

- 이 함수는 ReLU와 Leaky ReLU를 일반화 한 것. ReLU와 Leaky ReLU는 이 함수의 특수한 경우
- Maxout은 ReLU가 갖고 있는 장점을 모두 가지며, dying ReLU 문제도 해결
- ReLU 함수와 달리 한 뉴런에 대해 파라미터가 두배이기 때문에 전체 파라미터가 증가한다는 단점이 있음



$$f(x) = max(w_1^T x + b_1 + w_1^T x + b_1)$$

Activation Function: Conclusion

- **가장 먼저 ReLU를 시도**
  - 다양한 ReLU인 Leaky ReLU, ELU, Maxout등이 있지만, 현재까지 가장 많이 사용되는 activation은 ReLU임
- **다음으로 Leaky ReLU, Maxout, ELU를 시도**
  - 성능이 좋아 질 수 있는 가능성이 있음
  - 그러나 반드시 좋아지는 것은 아님
- **tanh를 사용해도 되지만 성능이 개선될 확률이 적음**
- **앞으로 Deep NN에서는 Sigmoid는 피한다**

Maxout > ELU, Leaky ReLU >= ReLU > tanh >= sigmoid

Activation Function: Conclusion

The compatibility of activation functions and initialization.
Dataset: CIFAR-10

Init method	maxout	ReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	89.28	n/c
OrthoNorm	93.78	91.74	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	–	n/c
Xavier	91.75	90.63	**89.82**	n/c
MSRA	n/c†	90.91	89.54	n/c

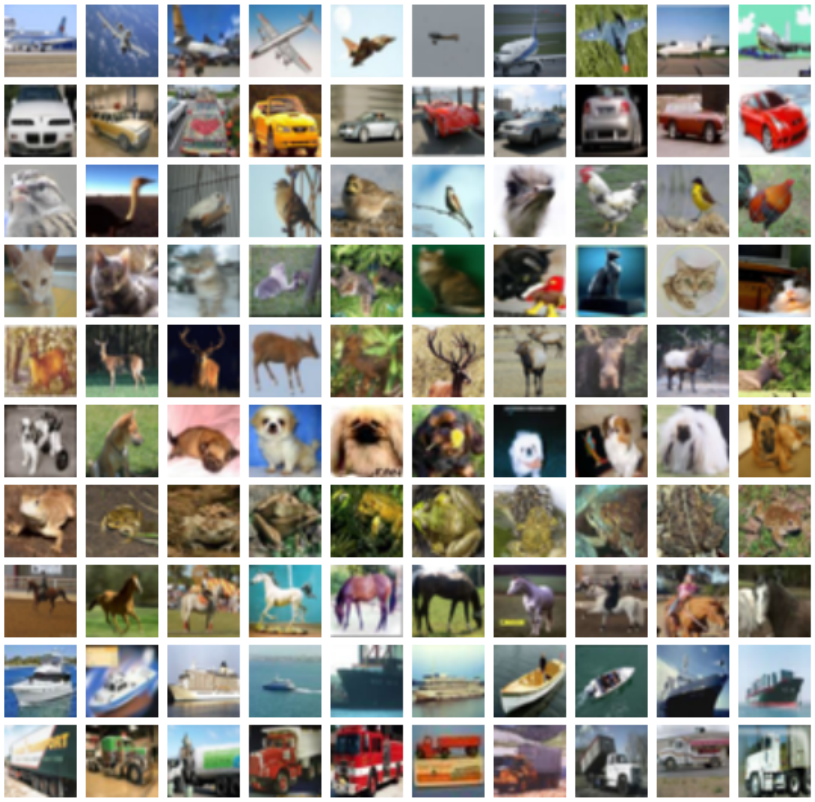
n/c symbol stands for "failed to converge, Architecture FitNets-17

Maxout > ELU, Leaky ReLU >= ReLU > tanh > sigmoid

"ALL YOU NEED IS A GOOD INIT", ICLR2016.

Appendix: CIFAR-10

비행기
자동차
새
고양이
사슴
개
개구리
말
배
트럭



- 32x32픽셀의 60000개 이미지
- 각 이미지는 10개의 클래스로 라벨링

Weight Initialization

"Initialize weights in a smart way”

Geoffrey Hinton's summary of findings up to today

➔ DNN에서 Gradient Vanishing 문제 해결법

- Our labeled datasets were thousands of times too small
- Our computers were millions of times too slow
- We initialized the weights in a stupid way
- We used the wrong type of non-linearity

Until Now

- 기본적인 선형 회귀나 Softmax 같은 알고리즘에서는 -1 ~ 1의 난수를 Weight로 사용
- Neural Network에서는 weight 선정에 주의 요망
- W=0이면 Backpropagation 시 gradient 값이 0되어 Gradient Vanishing 현상이 발생

주의!

Need to set the initial weight values wisely

- 절대 모두 0으로 초기화 하지 말 것
- 가중치를 어떻게 초기화 할 것이냐는 무척 도전적인 이슈
- Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets”

  : Restricted Boatman Machine (RBM)을 이용한 초기화 제안

Good news

❑ No need to use complicated RBM for weight initializations

❑ **Simple methods are OK**

✓ 노드의 입출력 수에 비례해서 초기값을 결정짓는 방법 제안

✓ **Xavier initialization**

- X. Glorot and Y. Bengio "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010

✓ **He's initialization**

- **K. He**, X. Zhang, S. Ren, and J. Sun "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," 2015

Xavier/He initialization

- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

입력수 출력수

```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

Pytorch 초기화 방법 ➔ https://pytorch.org/docs/stable/nn.init.html

Xavier initialization

`torch.nn.init.xavier_normal_(`*`tensor`*`, `*`gain=1.0`*`)` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}}$$

Also known as Glorot initialization.

Parameters

- **tensor** – an n-dimensional *torch.Tensor*
- **gain** – an optional scaling factor

Examples

```
>>> w = torch.empty(3, 5)
>>> nn.init.xavier_normal_(w)
```

```
[docs]def xavier_normal_(tensor, gain=1.):
    # type: (Tensor, float) -> Tensor
    r"""Fills the input `Tensor` with values according to the method
    described in `Understanding the difficulty of training deep feedforward
    neural networks` - Glorot, X. & Bengio, Y. (2010), using a normal
    distribution. The resulting tensor will have values sampled from
    :math:`\mathcal{N}(0, \text{std}^2)` where

    .. math::
        \text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan\_in} + \text{fan\_out}}}

    Also known as Glorot initialization.

    Args:
        tensor: an n-dimensional `torch.Tensor`
        gain: an optional scaling factor

    Examples:
        >>> w = torch.empty(3, 5)
        >>> nn.init.xavier_normal_(w)
    """
    fan_in, fan_out = _calculate_fan_in_and_fan_out(tensor)
    std = gain * math.sqrt(2.0 / float(fan_in + fan_out))

    return _no_grad_normal_(tensor, 0., std)
```

He initialization

`torch.nn.init.kaiming_normal_(`*`tensor`*`, `*`a=0`*`, `*`mode='fan_in'`*`, `*`nonlinearity='leaky_relu'`*`)` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* - He, K. et al. (2015), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}$$

Also known as He initialization.

Parameters

- **tensor** – an n-dimensional *torch.Tensor*
- **a** – the negative slope of the rectifier used after this layer (only
- **with 'leaky_relu')** (*used*) –
- **mode** – either `'fan_in'` (default) or `'fan_out'`. Choosing `'fan_in'` preserves the magnitude of the variance of the weights in the forward pass. Choosing `'fan_out'` preserves the magnitudes in the backwards pass.
- **nonlinearity** – the non-linear function (*nn.functional* name), recommended to use only with `'relu'` or `'leaky_relu'` (default).

```
[docs]def kaiming_normal_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu'):
    r"""Fills the input `Tensor` with values according to the method
    described in `Delving deep into rectifiers: Surpassing human-level
    performance on ImageNet classification` - He, K. et al. (2015), using a
    normal distribution. The resulting tensor will have values sampled from
    :math:`\mathcal{N}(0, \text{std}^2)` where

    .. math::
        \text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}

    Also known as He initialization.

    Args:
        tensor: an n-dimensional `torch.Tensor`
        a: the negative slope of the rectifier used after this layer (only
        used with ```'leaky_relu'```)
        mode: either ```'fan_in'``` (default) or ```'fan_out'```. Choosing ```'fan_in'```
            preserves the magnitude of the variance of the weights in the
            forward pass. Choosing ```'fan_out'``` preserves the magnitudes in the
            backwards pass.
        nonlinearity: the non-linear function (`nn.functional` name),
            recommended to use only with ```'relu'``` or ```'leaky_relu'``` (default).

    Examples:
        >>> w = torch.empty(3, 5)
        >>> nn.init.kaiming_normal_(w, mode='fan_out', nonlinearity='relu')
    """
    fan = _calculate_correct_fan(tensor, mode)
    gain = calculate_gain(nonlinearity, a)
    std = gain / math.sqrt(fan)
    with torch.no_grad():
        return tensor.normal_(0, std)
```

Activation Function: Conclusion

The compatibility of activation functions and initialization.
Dataset: CIFAR-10

Init method	maxout	ReLU	VLReLU	tanh	Sigmoid
LSUV	**93.94**	**92.11**	92.97	89.28	n/c
OrthoNorm	93.78	91.74	92.40	89.48	n/c
OrthoNorm-MSRA scaled	–	91.93	**93.09**	–	n/c
Xavier	91.75	90.63	92.27	**89.82**	n/c
MSRA	n/c†	90.91	92.43	89.54	n/c

↳ He's init

"ALL YOU NEED IS A GOOD INIT", ICLR2016.

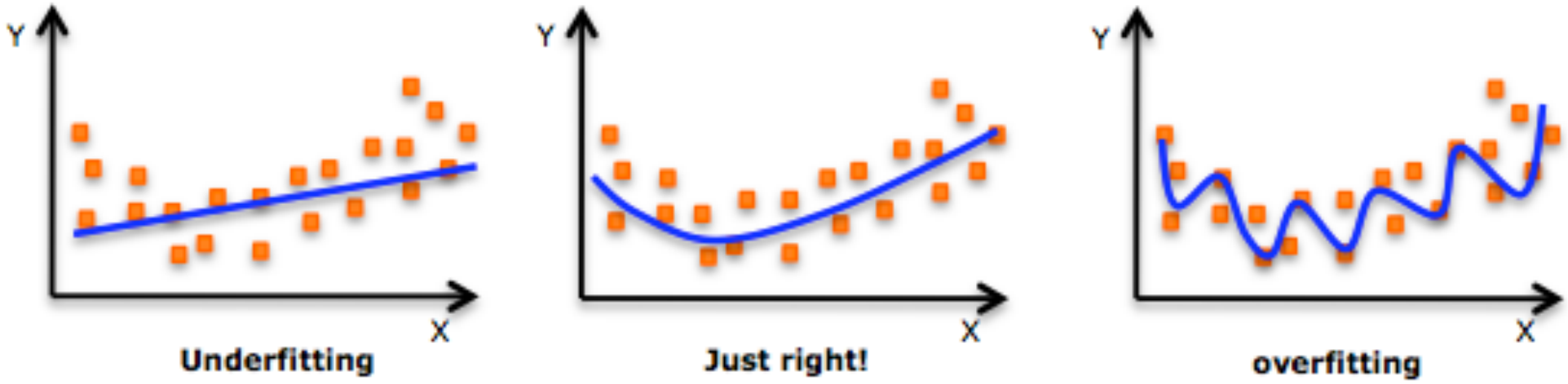
Still an active area of research

- **We don’t know how to initialize perfect weight values, yet**
- Many new algorithms
  - Batch normalization
  - Layer sequential uniform variance
  - …

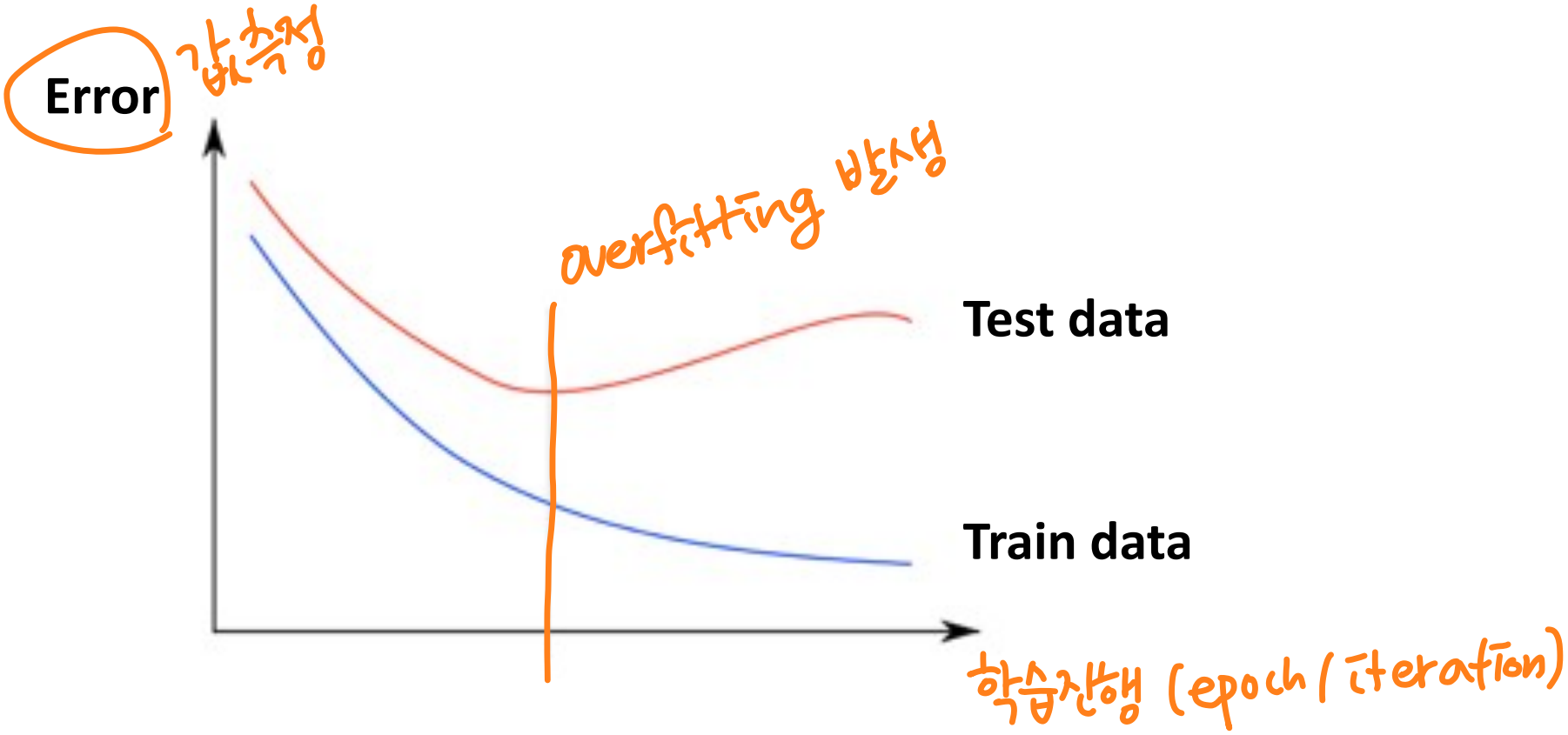
Dropout and model Ensemble

[복습] Overfitting

- 너무 과도하게 데이터에 대해 모델을 learning을 한 경우를 의미함
- 현 학습 데이터만 잘 표현하면, 새로운 데이터에 대한 대응력이 없어 모델 학습 의미 상실



[복습] Am I overfitting?



- ✓ Very high accuracy on the training dataset (0.99)
- ✓ Poor accuracy on the test data set (0.85)

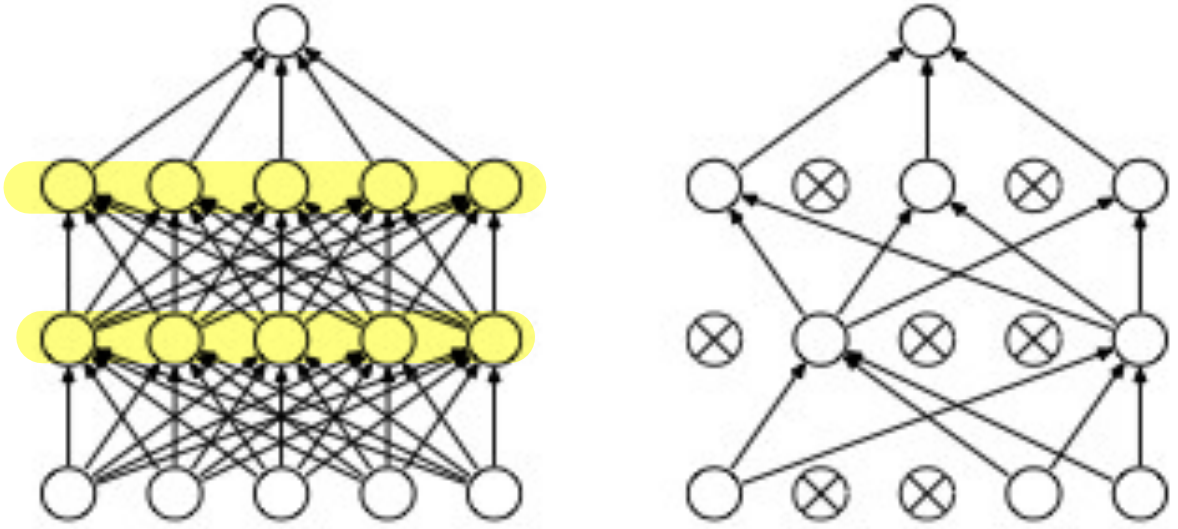
[복습] Solution for overfitting

- More training data!
- Reduce the number of features
- **Regularization (-Dropout)**

Dropout for overfitting

- 훈련 데이터에 대한 복잡한 공동 적응을 방지하여 신경망의 과적 합을 줄이기 위한 Google이 제안한 정규화 기술 (regularization)
- "드롭 아웃"이라는 용어는 신경망에서 유닛을 제거하는 것
- 학습 시에만 적용하고 테스트 시에는 모든 유닛을 사용함

0.3 / 0.5정도 쓴다면 ?



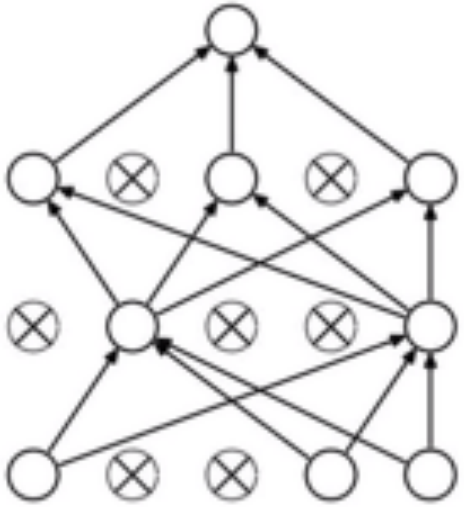
(a) Standard Neural Net (b) After applying dropout.

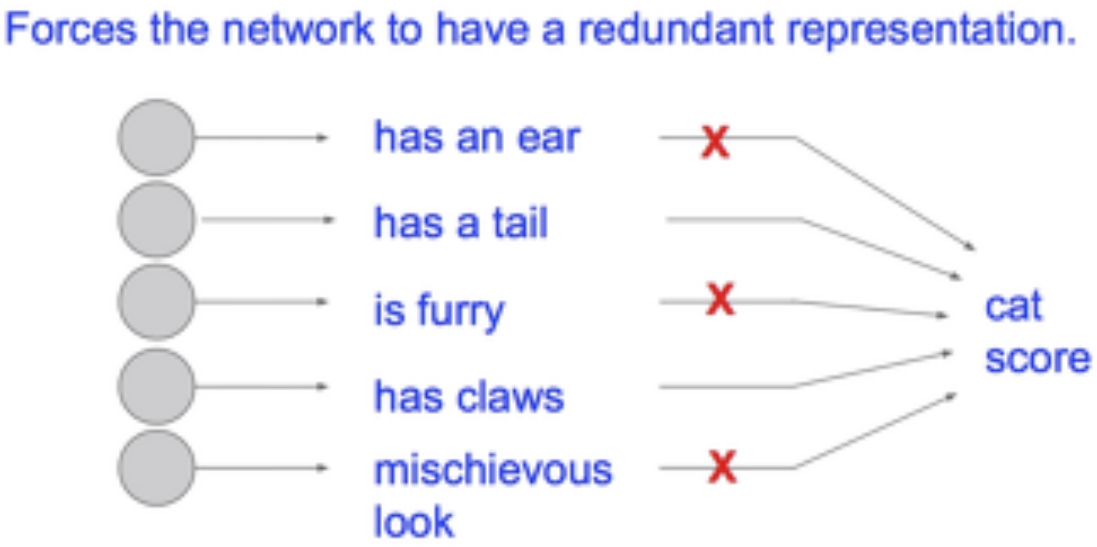
Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Dropout for overfitting

매학습↑ 항상 다른 edge를 끊음 아다

- 왜 성능향상을 가져오는가?
  - 드랍아웃을 통해 Ensemble model 학습과 같은 효과가 있기 때문이다.
  - Ensemble model이란 집단지성으로 이해할 수 있다.





Dropout

CLASS `torch.nn.Dropout(p=0.5, inplace=False)` [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters

- **p** – probability of an element to be zeroed. Default: 0.5
- **inplace** – If set to `True`, will do this operation in-place. Default: `False`

Shape:

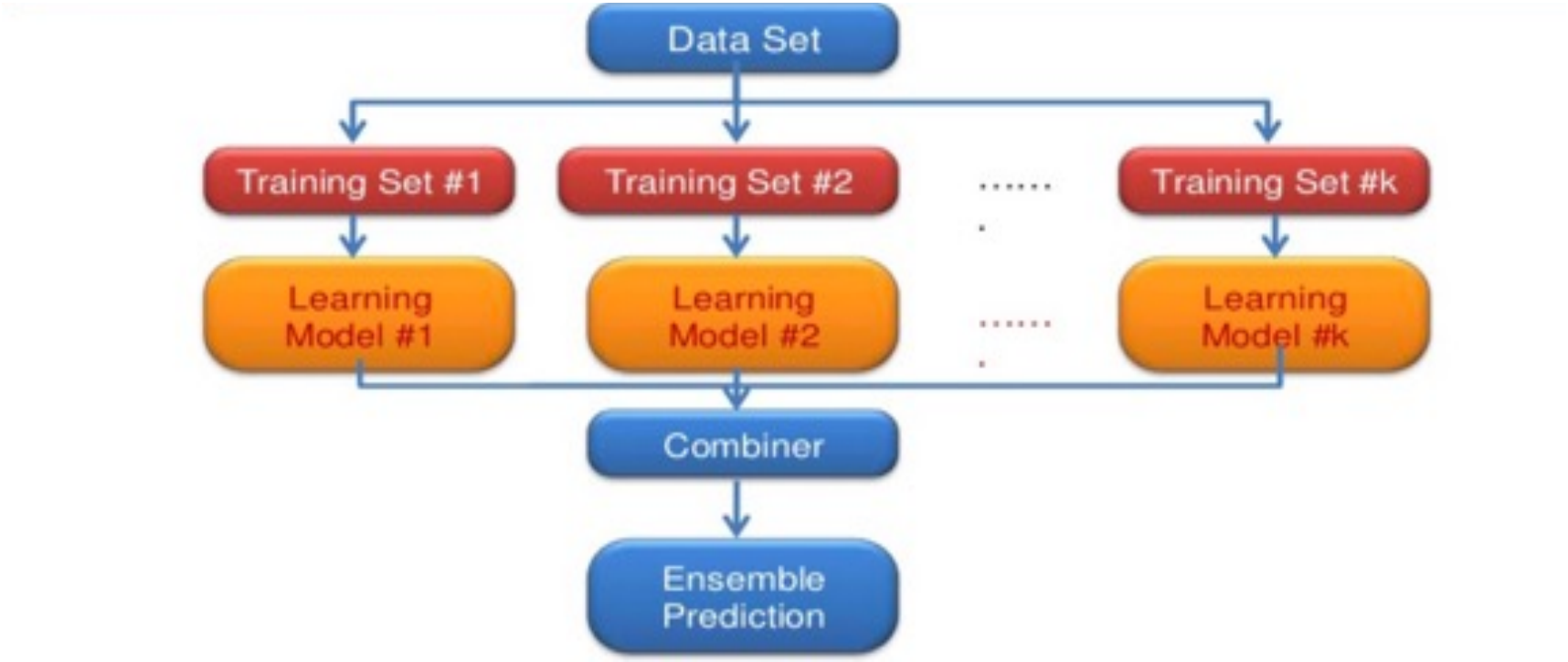
- Input: $(*)$. Input can be of any shape
- Output: $(*)$. Output is of the same shape as input

Examples:

```
>>> m = nn.Dropout(p=0.2)
>>> input = torch.randn(20, 16)
>>> output = m(input)
```

Ensemble

- (통계학과 기계 학습에서) 앙상블 학습법은 학습 알고리즘들을 따로 쓰는 경우에 비해 더 좋은 예측 성능을 얻기 위해 다수의 학습 알고리즘을 사용하는 방법을 말함



Summary

- DNN 모델 학습을 위한 팁

맨마지막 layer 활성함수 쓰지 X

NN-ReLU-NN-ReLU-NN

  - 활성 함수를 잘 선택한다.
    - ReLU가 가장 널리 사용된다.
  - 가중치 초기화 방법을 잘 선택한다.
    - Xavier가 가장 널리 사용된다
  - 드랍 아웃을 잘 적용한다.
    - "NN-ReLU-Dropout"을 하나의 블락으로 쌓는다.
  - BN을 잘 적용한다.
    - "NN-ReLU-BN"을 하나의 블락으로 쌓는다.

Batchnorm

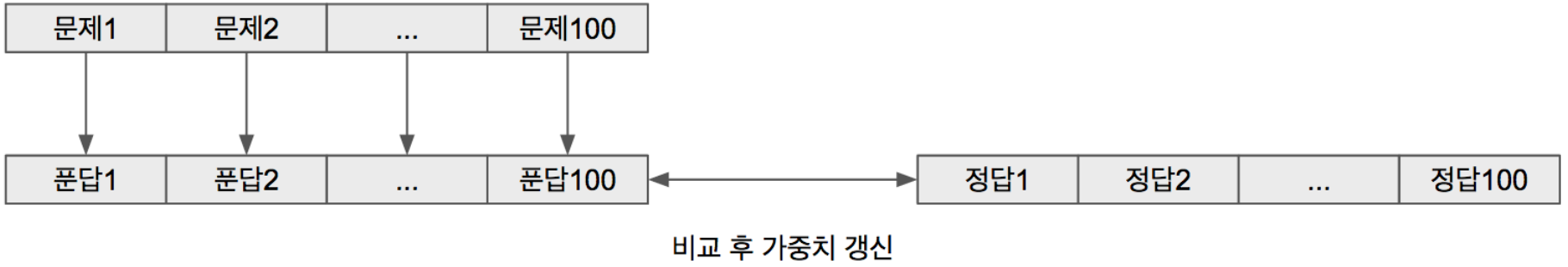
MNIST를 이용한 NN 실습

MNIST를 이용한 NN 실습
-사전 지식 쌓기-

Batch_size (배치사이즈)

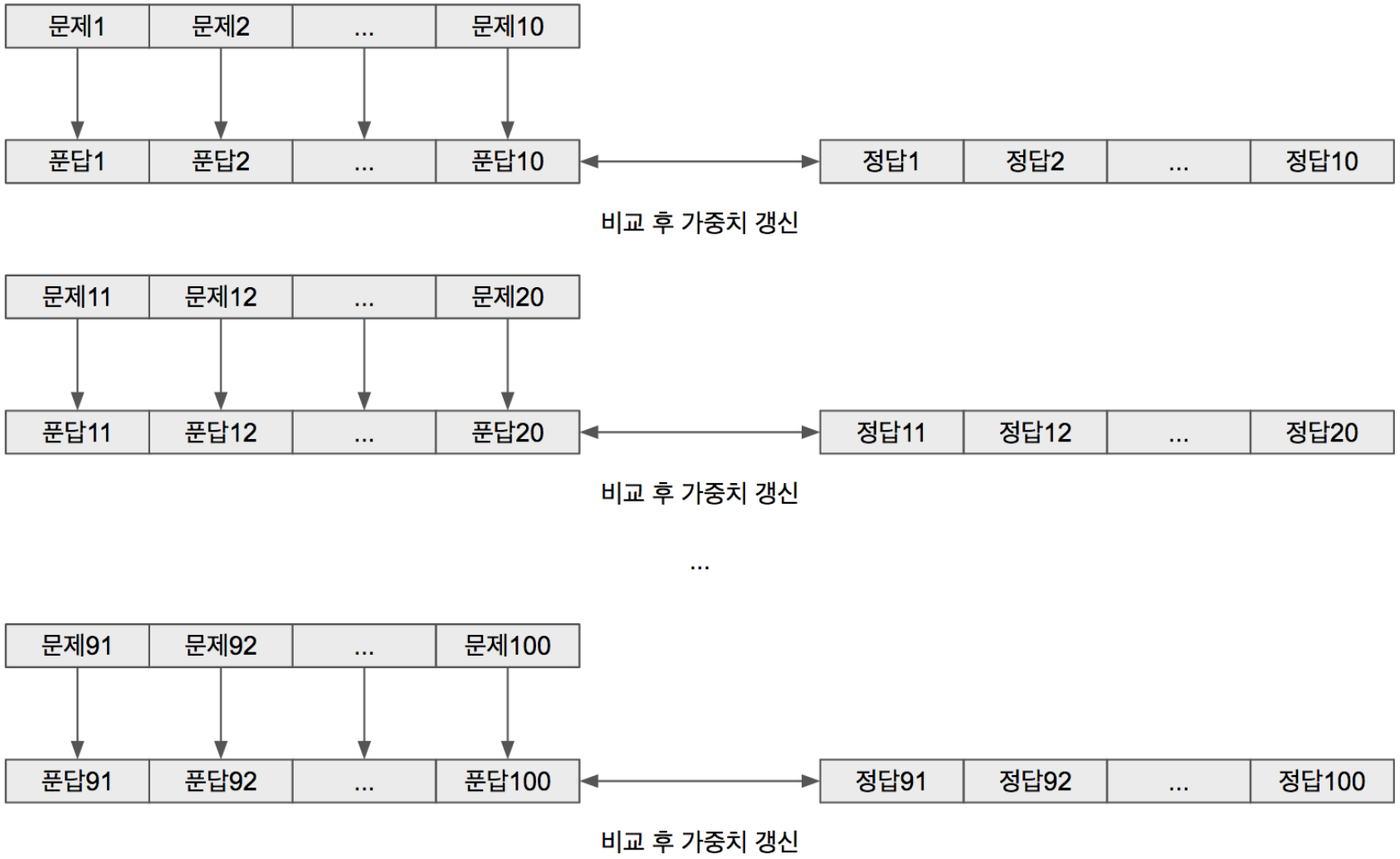
- 배치사이즈는 몇 문항을 풀고 해답을 맞추는 지를 의미
- 100문항일 때, 배치사이즈가 100이면 전체를 다 풀고 난 뒤에 해답을 맞춰보는 것
- 우리가 해답을 맞춰볼 때 '아하, 이렇게 푸는구나' 라고 느끼면서 학습하는 것처럼 모델도 이러한 과정을 통해 가중치가 갱신

전체 문제를 푼 뒤 해답과 맞추므로 이 때 가중치 갱신은 한 번만 일어납니다.



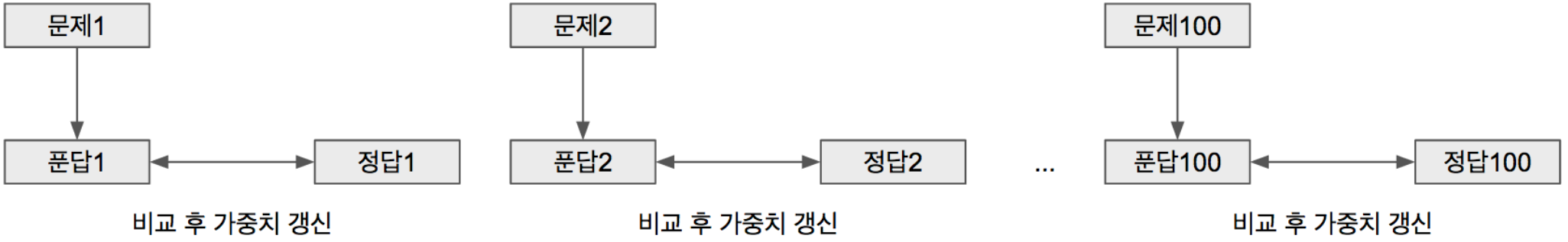
Batch_size (배치사이즈)

배치사이즈가 10이면 열 문제씩 풀어보고 해답 맞춰보는 것입니다.
100문항을 10문제씩 나누어서 10번 해답을 맞추므로 가중치 갱신은 10번 일어납니다.



Batch_size (배치사이즈)

배치사이즈가 1이면 한 문제 풀고 해답 맞춰보고 또 한 문제 풀고 맞춰보고 하는 것입니다.
한 문제를 풀 때마다 가중치 갱신이 일어나므로 횟수는 100번입니다.



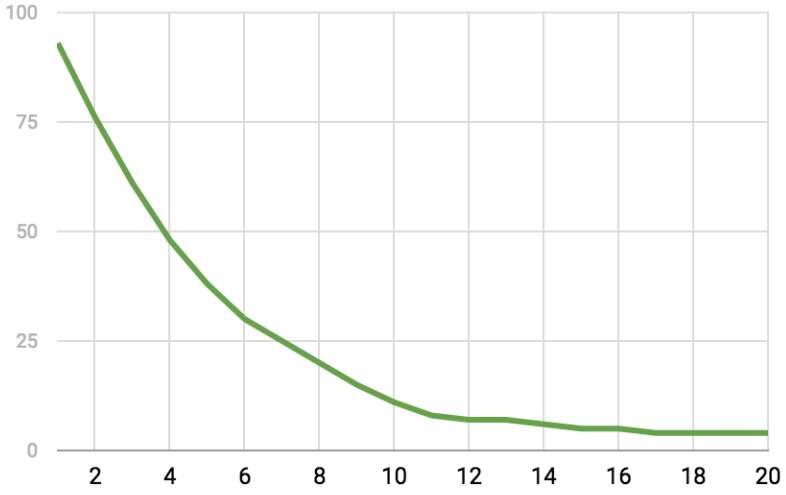
Batch_size = 100 vs Batch_size = 1

- **100문제 다 풀고 해답을 맞히는 것과 1문제씩 풀고 해답을 맞히는 것은 어떤 차이가 있을까요?**
- 언뜻 생각해서는 별 반 차이가 없어 보이지만 모의고사 1회분에 비슷한 문항이 있다고 가정했을 때, 배치사이즈가 100일 때는 다 풀어보고 해답을 맞춰보기 때문에 한 문제를 틀릴 경우 이후 유사 문제를 모두 틀릴 경우가 많습니다.
- 배치사이즈가 1인 경우에는 한 문제씩 풀어보고 해답을 맞춰보기 때문에 유사문제 중 첫 문제를 틀렸다고 하더라도 해답을 보면서 학습하게 되므로 나머지 문제는 맞추게 됩니다.

- **자 그럼 이 배치사이즈가 어떨 때 학습효과가 좋을까요?**
- 사람이 학습하는 것이랑 비슷합니다. 100문항 다 풀고 해답과 맞추어보려면 문제가 무엇이었는지 다 기억을 해야 맞춰보면서 학습이 되겠죠? 기억력(용량)이 커야합니다. 즉, GPU 메모리가 엄청 켜야 합니다.
- 1문항씩 풀고 해답 맞추면 학습은 꼼꼼히 잘 되겠지만 학습하는데 시간이 너무 걸리겠죠?

Epochs (에포크)

- 에포크는 모의고사 1회분을 몇 번 풀어볼까를 의미함
- 에포크가 20이면 모의고사 1회분을 20번 푸는 것.
- 우리가 같은 문제집을 여러 번 풀면서 점차 학습되듯이 모델도 같은 데이터셋으로 반복적으로 가중치를 갱신하면서 모델이 학습됨



그래프에서 세로축이 100문항 중 틀린 개수이고, 가로축이 모의고사 풀이 반복횟수를 의미함.

풀이를 반복할수록 틀린 개수가 적어지는 것을 보수 있음.

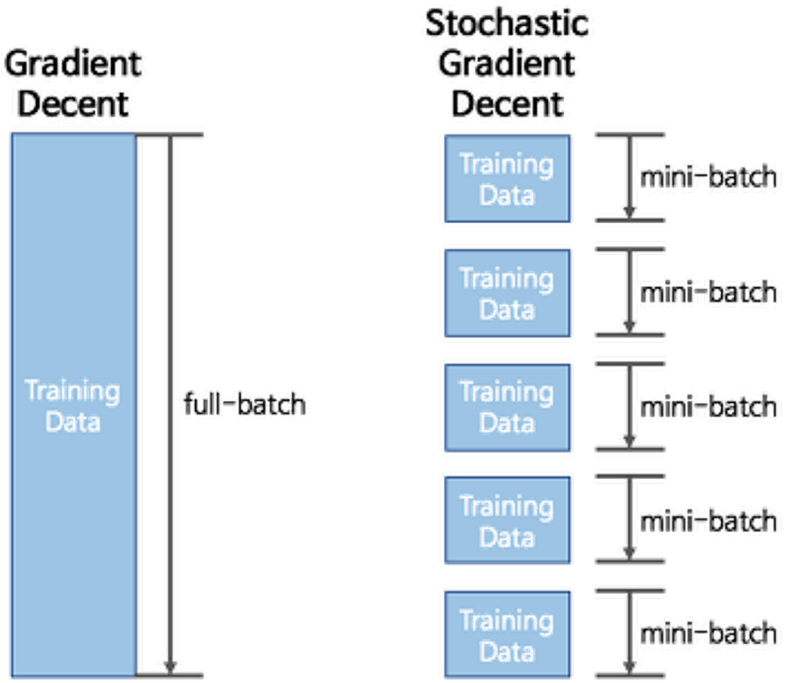
처음에는 틀린 개수가 확 줄어들지만, 반복이 늘어날수록 완만하게 틀린 오답 수가 줄어듬.

우리가 공부할 때도 낮은 점수에서는 공부를 조금하면 점수가 확 오르지만, 높은 점수에서 1~2점 올리는 것이 쉽지 않은 것과 비슷

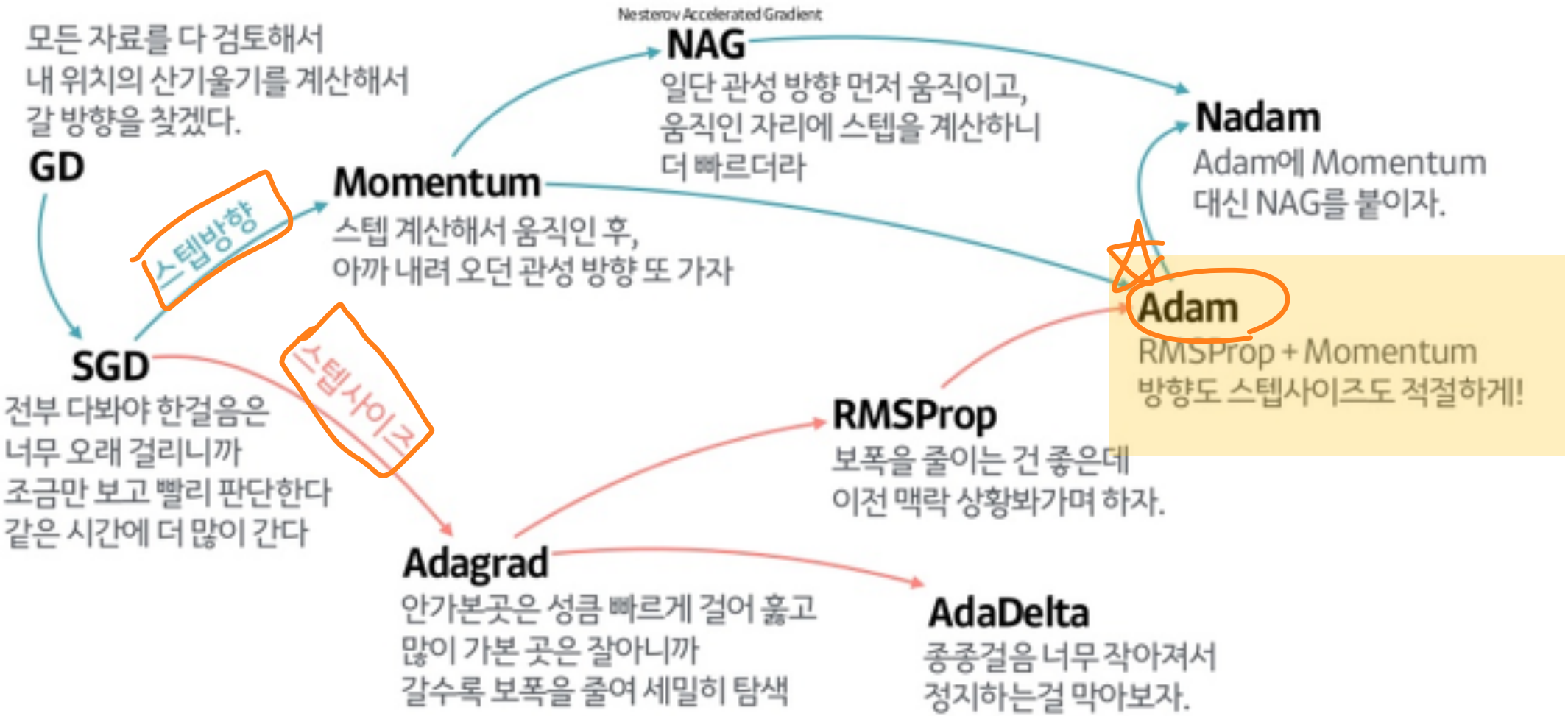
Optimizer: 기본

- Gradient Descent (GD)
  - 학습데이터 전체를 사용하는 최적화
- Stochastic Gradient Descent (SGD)
  - 학습데이터 일부(mini-batch)를 사용하는 최적화
  - BGD보다 빠르게 수렴
  - SGD를 여러 번 반복할 수록 BGD와 유사한 결과로 수렴

배치사이즈별로 최적화

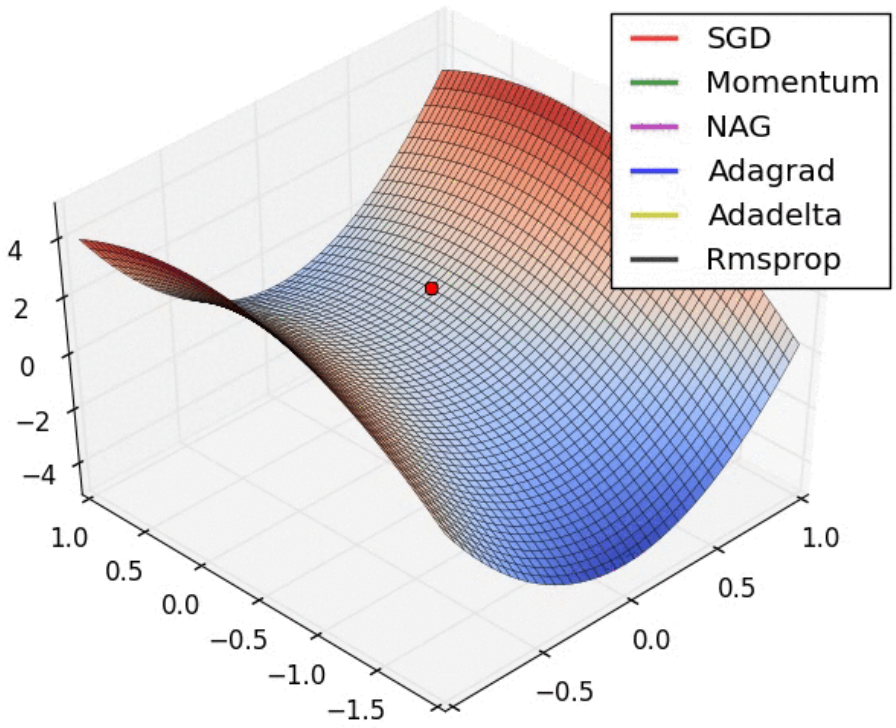


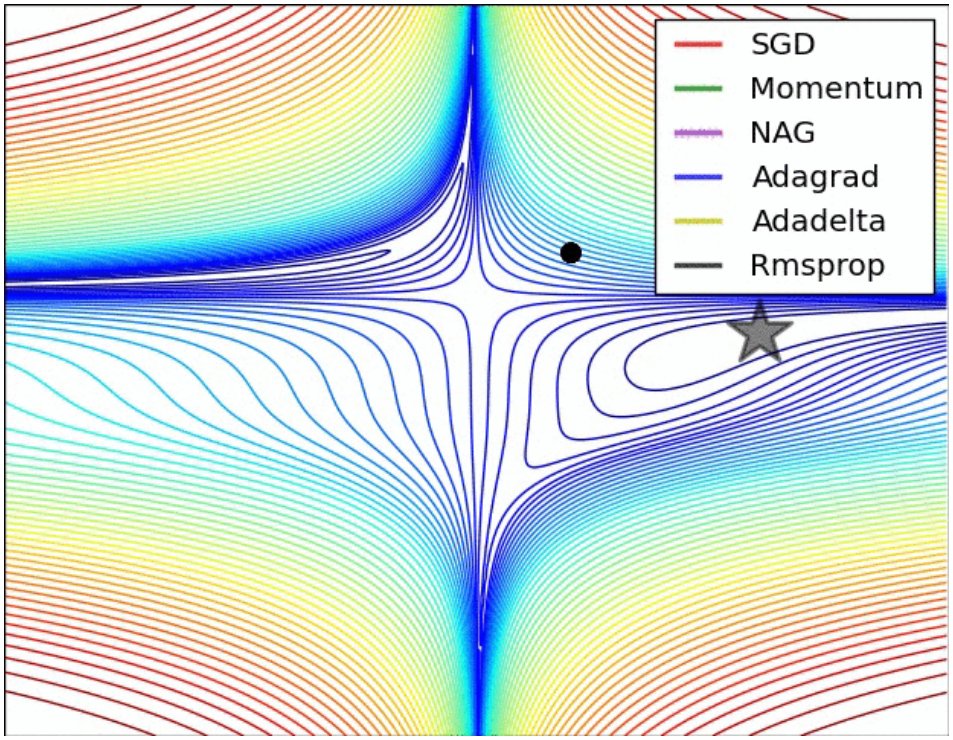
Optimizer : 발전 방향



From 하용호

Optimizer : 비교





http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

MNIST를 이용한 NN 실습
-배운지식활용하기-

MNIST Dataset 소개

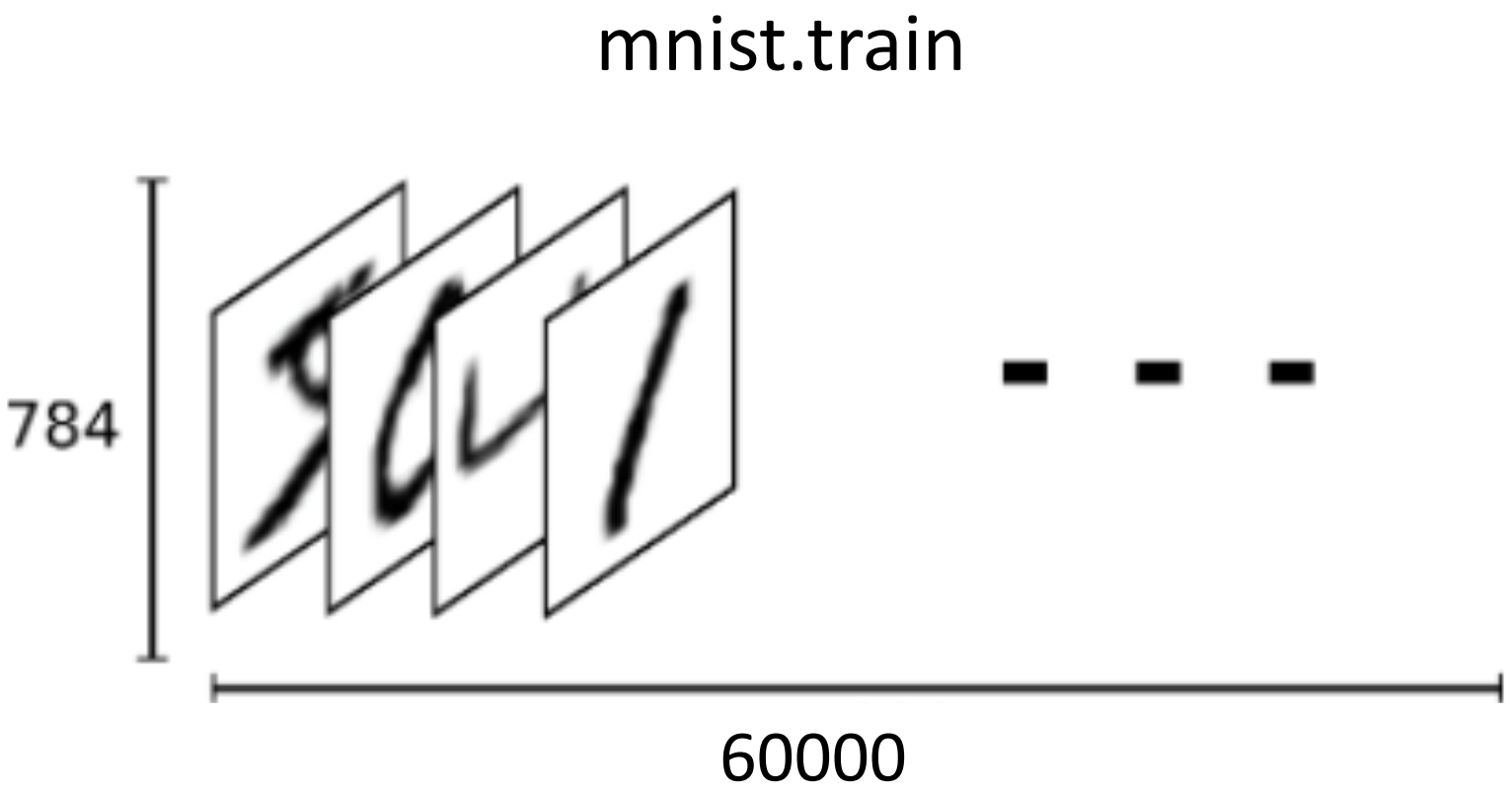
학습데이터: 60000x784
학습데이터 라벨: 60000x10
테스트데이터: 10000x784
테스트데이터 라벨: 10000x10

MNIST Dataset 소개

$$\sim$$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

MNIST Dataset 소개



tutorials.pytorch.kr/index.html

앱 RCV 연구실 세종RCV SAIR Gmail RCV insta. SAIR RCV 국가과제 RCV-DB-NAS-1 RCV-DB-NAS-2 RCV-DGX-NAS AI601-NAS 일상생활 기타 북마크

Fork me on GitHub

Get Started Features Ecosystem Blog Tutorials Docs Resources Github

1.4.0

Search Tutorials

시작하기 (Getting Started)

파이토치(PyTorch)로 딥러닝하기: 60분만에 끝장내기
사용자 정의 Dataset, Dataloader, Transforms 작성하기
TensorBoard로 모델, 데이터, 학습 시각화하기

이미지 (Image)

TorchVision 객체 검출 미세조정(Finetuning) 튜토리얼
컴퓨터 비전(Vision)을 위한 전이학습(Transfer Learning)
공간 변형기 네트워크(Spatial Transformer Networks) 튜토리얼
PyTorch를 이용한 신경망-변환(Neural-Transfer)
적대적 예제 생성(Adversarial Example Generation)

오디오 (Audio)

torchaudio Tutorial

텍스트 (Text)

기초부터 시작하는 NLP: 문자-단위 RNN으로 이름 분류하

Tutorials > 파이토치(PyTorch) 튜토리얼에 오신 것을 환영합니다

Shortcuts

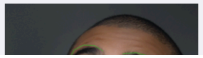
파이토치(PYTORCH) 튜토리얼에 오신 것을 환영합니다

PyTorch를 어떻게 사용하는지 알고 싶다면 시작하기(Getting Started) 튜토리얼부터 시작해보세요. 파이토치(PyTorch)로 딥러닝하기: 60분만에 끝장내기 가 가장 일반적인 출발점으로, 심층 신경망(deep neural network)을 구축할 때 PyTorch를 어떻게 사용하는지에 대한 전반적인 내용을 기본부터 제공합니다.

아래 내용도 한 번 살펴보시면 좋습니다:

- 사용자들이 튜토리얼과 연관된 노트북을 Google Colab에서 열어볼 수 있는 기능이 추가되었습니다. 이 페이지 를 방문하셔서 자세히 알아보세요!
- 각 튜토리얼에는 Jupyter 노트북과 Python 소스코드를 다운로드할 수 있는 링크가 있습니다. IPython 또는 Jupyter를 이용하여 대화식으로 실행하시려면 이용해보세요.
- 이미지 분류, 비지도 학습, 강화 학습, 기계 번역을 비롯한 다양한 고품질의 예제가 PyTorch Examples 에 준비되어 있습니다.
- PyTorch API, 계층(layer)에 대한 참고 문서는 PyTorch Docs 를 참고해주세요.
- 튜토리얼 개선에 참여하시려면 이곳 에 의견과 함께 이슈를 남겨주세요. (역자 주: 한국어 번역과 관련한 오타 및 오역 등은 한국어 번역 저장소 에 부탁드립니다.)
- PyTorch 치트 시트(Cheat Sheet) 를 참고하시면 유용한 정보들을 얻으실 수 있습니다.
- 마지막으로 PyTorch 릴리즈 노트(Release Notes) 도 참고해보세요.

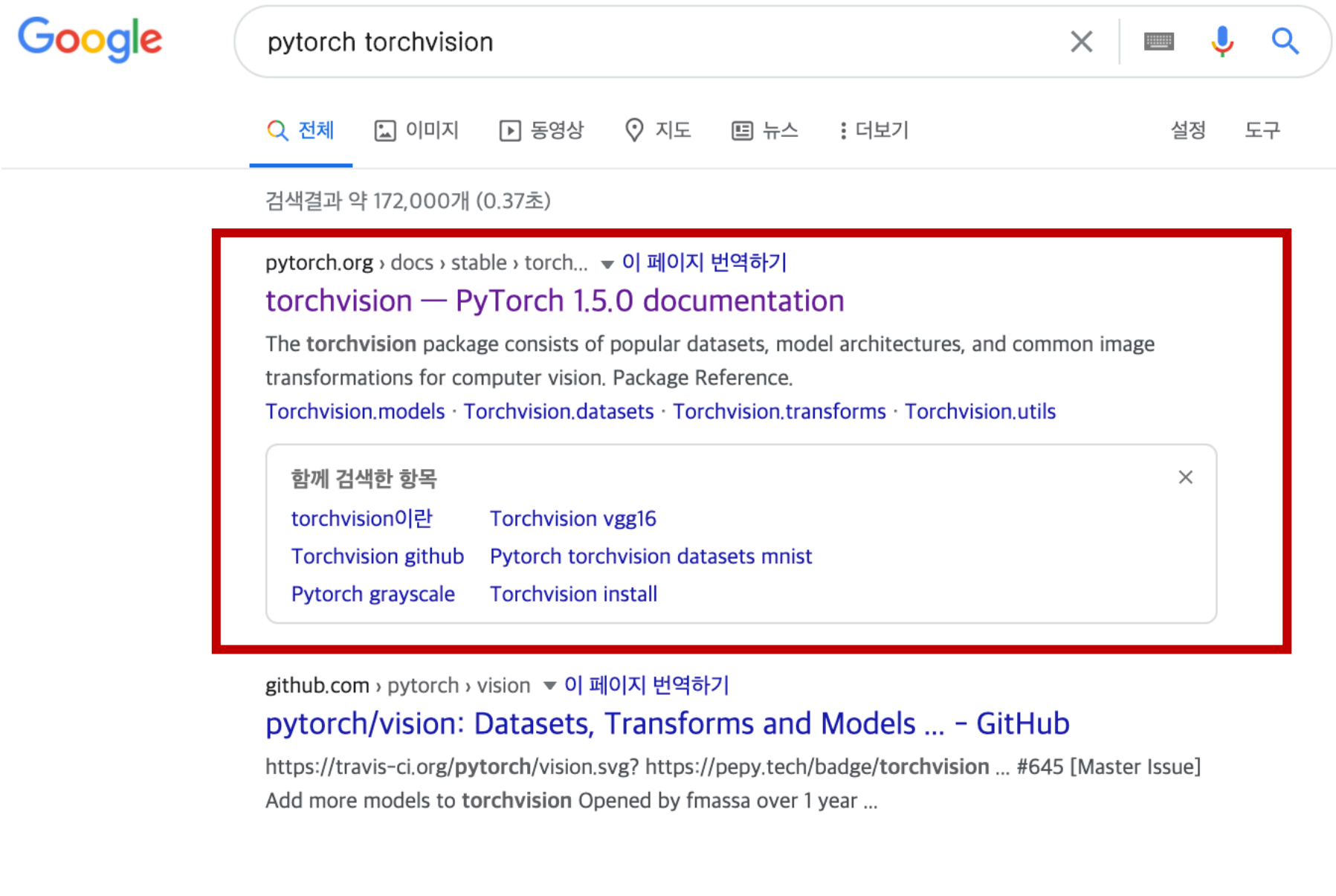
시작하기 (Getting Started)



PyTorch

파이토치(PyTorch) 튜토리얼에 오신 것을 환영합니다
시작하기 (Getting Started)
이미지 (Image)
Named Tensor (experimental)
오디오 (Audio)
텍스트 (Text)
강화 학습 (Reinforcement Learning)
PyTorch 모델을 운영환경 (Production)에 배포하기
병렬 & 분산 학습 (Parallel and Distributed Training)
PyTorch 확장하기
모델 최적화 (Model Optimization
다른 언어에서의 PyTorch (PyTorc in Other Languages)
+ PyTorch Fundamentals In-Depth

torchvision



torchvision

- The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision.
- https://pytorch.org/docs/stable/torchvision/index.html

Package Reference

- torchvision.datasets
  - MNIST
  - Fashion-MNIST
  - KMNIST
  - EMNIST
  - QMNIST
  - FakeData
  - COCO
  - LSUN
  - ImageFolder
  - DatasetFolder
  - ImageNet
  - CIFAR
  - STL10
  - SVHN
  - PhotoTour
  - SBU
  - Flickr
  - VOC
  - Cityscapes

다양한 실험

0.3 / 0.9809

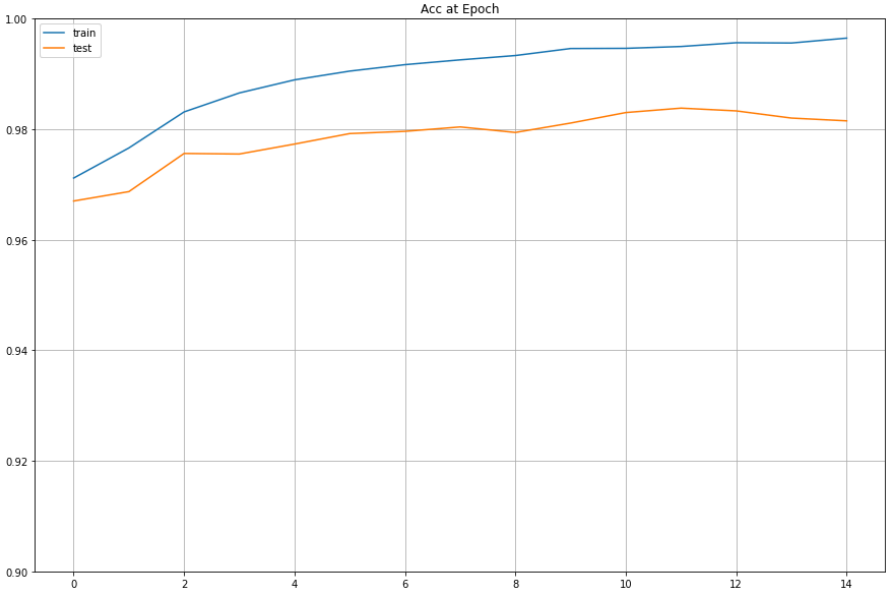
- ① Random Init / NN Layer #1 (784=>10) / **SGD** / CrossEntropy
  - https://colab.research.google.com/drive/15ArrVdJELyV-PeuATH5poe2uZoHm6Ih8
- ② Random Init / NN Layer #1 (784=>10) / **Adam** / CrossEntropy
  - https://colab.research.google.com/drive/1bimNZtout48XzSy7VoWbmdpOHQ-SAkHX
- ③ Random Init / **NN Layer #3** (784=>256=>10) / Adam / CrossEntropy
  - https://colab.research.google.com/drive/1jsP6IkoSZW5roMip53QSPJjiVP6Ej3Zk
- **Xavier** Init / NN Layer #3 (784=>256=>10) / Adam / CrossEntropy
  - https://colab.research.google.com/drive/1CI11CA5otqB7-RsQLKak3LJW3xeCUr9q
- Xavier Init / **DNN Layer #5** (784=>256=>10) / Adam / CrossEntropy
  - https://colab.research.google.com/drive/1y9qF3D4vbQVhX_dZCgplRuw_EcgW8Sg2
- Xavier Init / DNN Layer #5 (784=>256=>256=>256=>10) / Adam / CrossEntropy / **dropout (0.3)**
  - https://colab.research.google.com/drive/15Q5GSAPgsHeR0Y70zaiigHBBo4kwJ86o

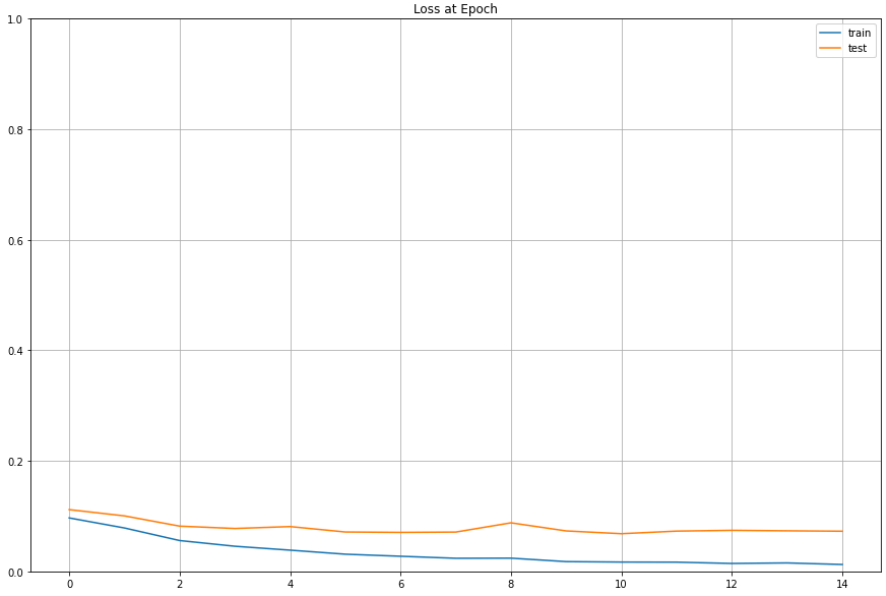
Adam이 더 낫다.

SGD	Adam	NN	Xavier	DNN	Dropout
0.42	0.77	0.94	0.9791	0.9824	0.9776

학습 과정 Plot하기

- Xavier Init / NN Layer #5 (784=>256=256=>256=>10) / Adam / CrossEntropy / dropout (0.3)
  - https://colab.research.google.com/drive/1EzvjlRNjMvo4ECnFuxx6ReuXih_vPA8k





Test Accuracy: 0.9815