KOREA UNIVERSITY

Deep Learning (Training Networks)

고려대학교 산업경영공학과 김동화

DS·BA
Data science & Business analyties

Contents

Background

❖ 영상인식(Image Recognition)의 어려운 점들(Challenges)

Challenges: Illumination

Challenges: Deformation

Challenges: Occlusion

Challenges: Background Clutter

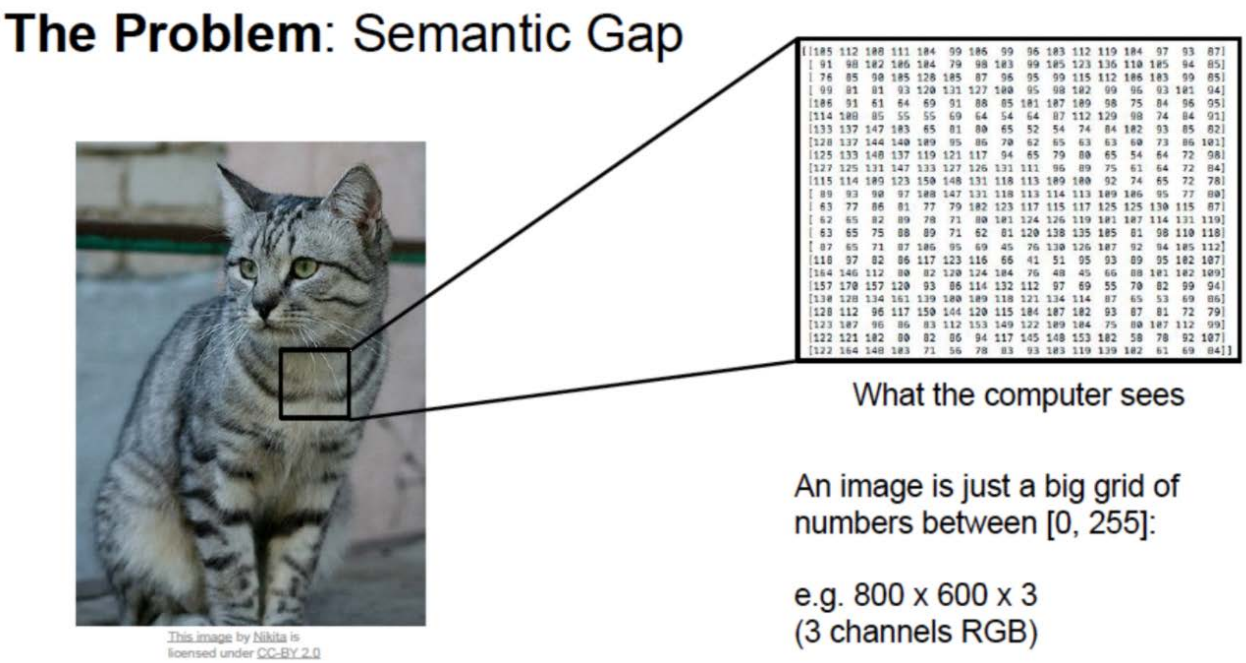
Challenges: Intraclass variation

Slide credit: CS231n
(https://goo.gl/J7KvrR)

Background

❖ 영상인식(Image Recognition)의 어려운 점들(Challenges)

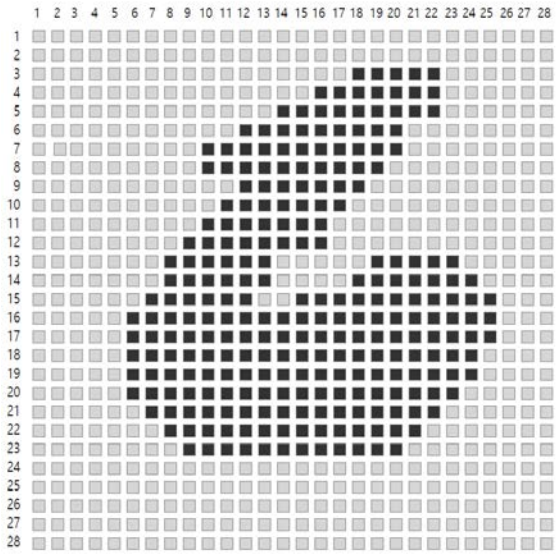
- 컴퓨터가 바라보는 화면



Neural Network Architecture

- Perceptron
- Multi Layer perceptron
- Convolution Neural Network
- Recurrent Neural Network

Sample images of MNIST data



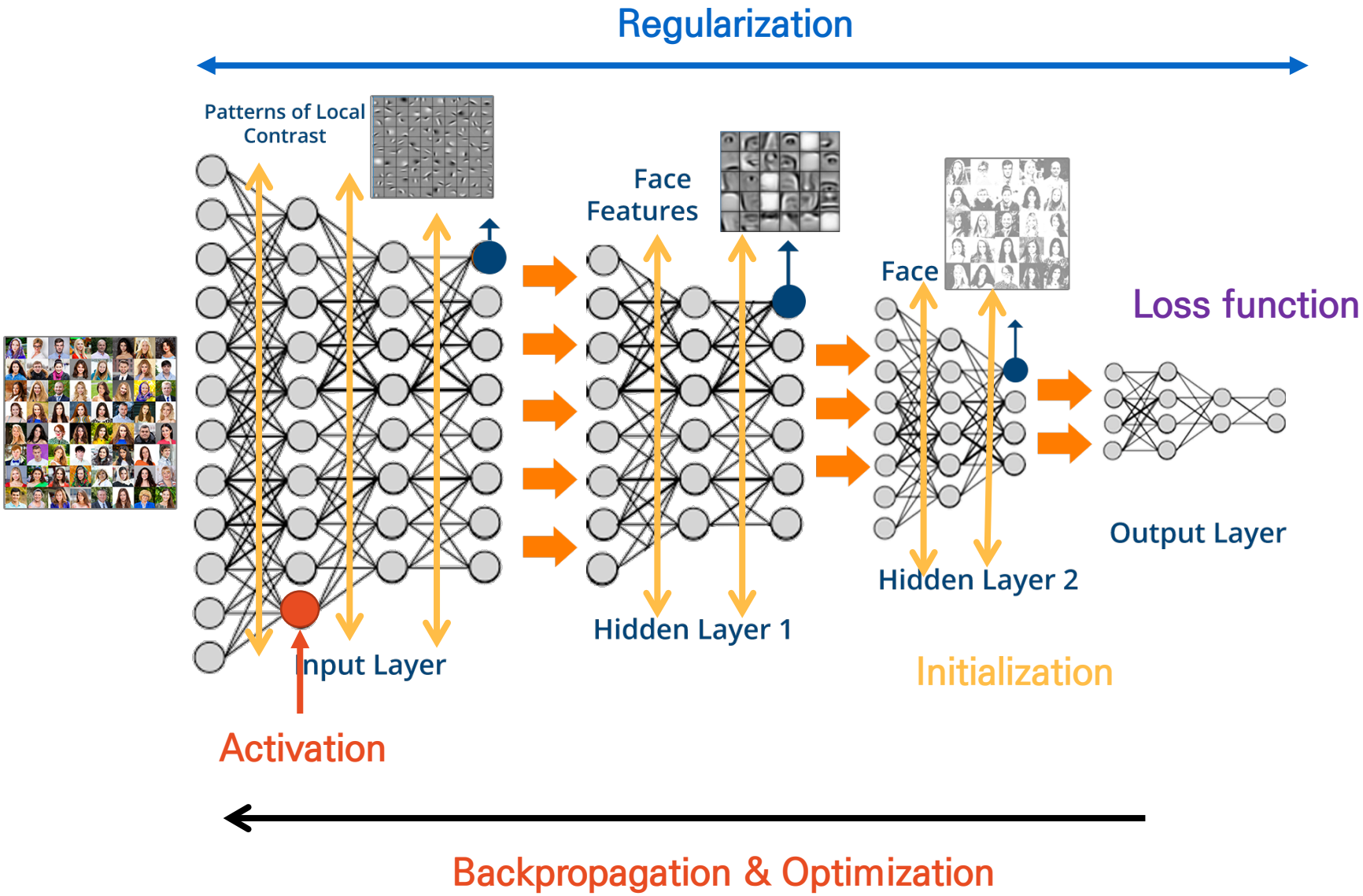
How the pixels look:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

How the pixels are stored:

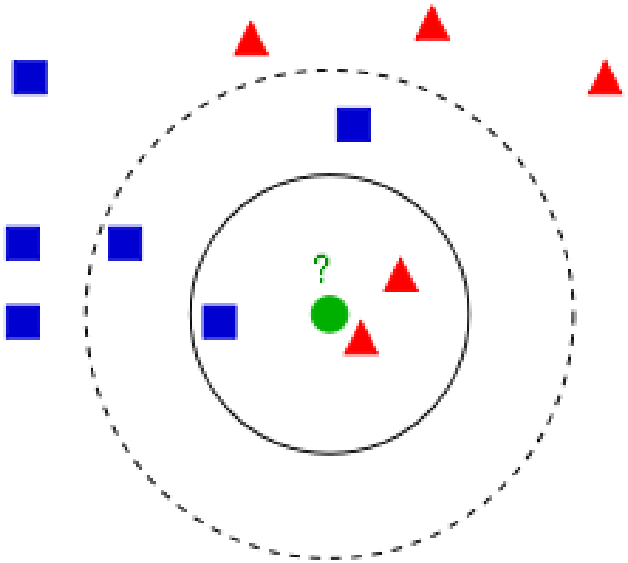
0	1	2	3	4	5	6	7	8	9	.	.	.		

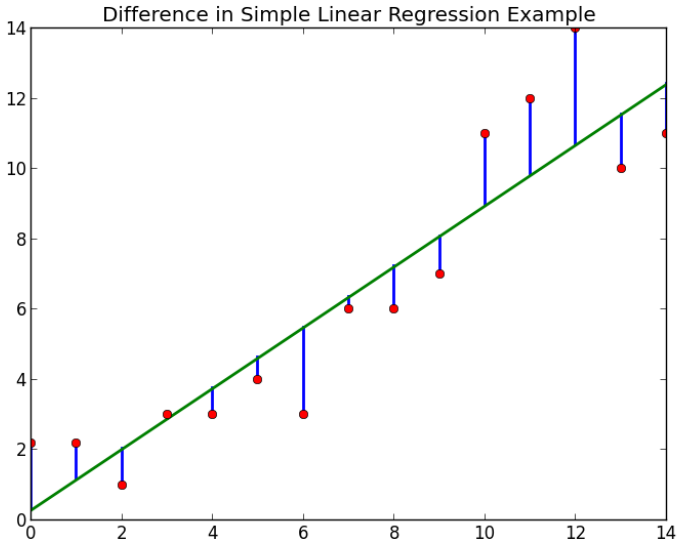
Deep Learning



Background

- KNN: 가장 가까운 examples에 대한 majority vote
- Linear regression: Least square estimation





Nearest Neighbor Classifier

- **Nearest Neighbor Classifier**
  - L1 distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

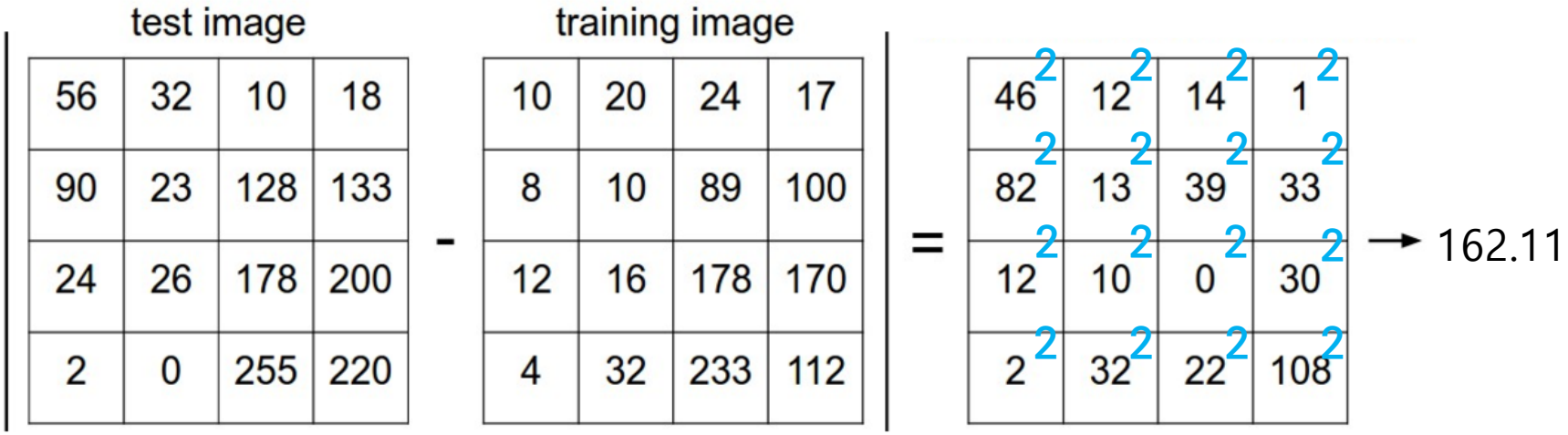
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

Nearest Neighbor Classifier

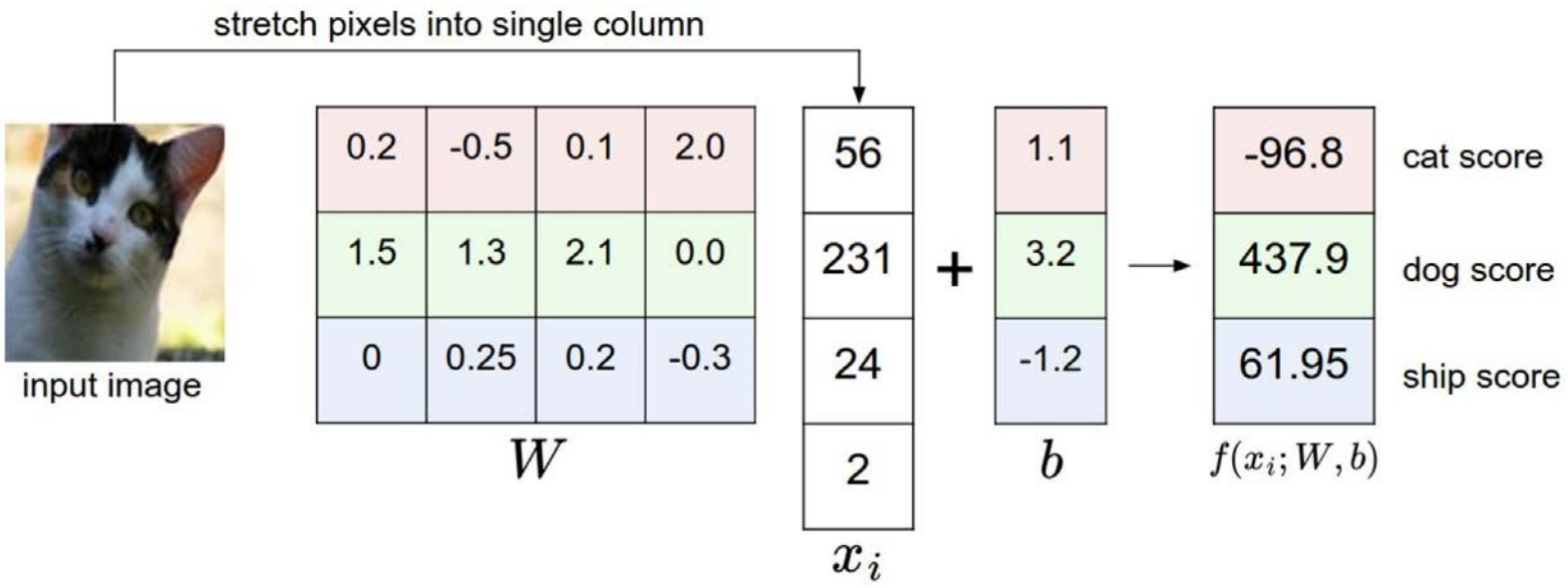
- **Nearest Neighbor Classifier**
  - L2 distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



Linear Classification

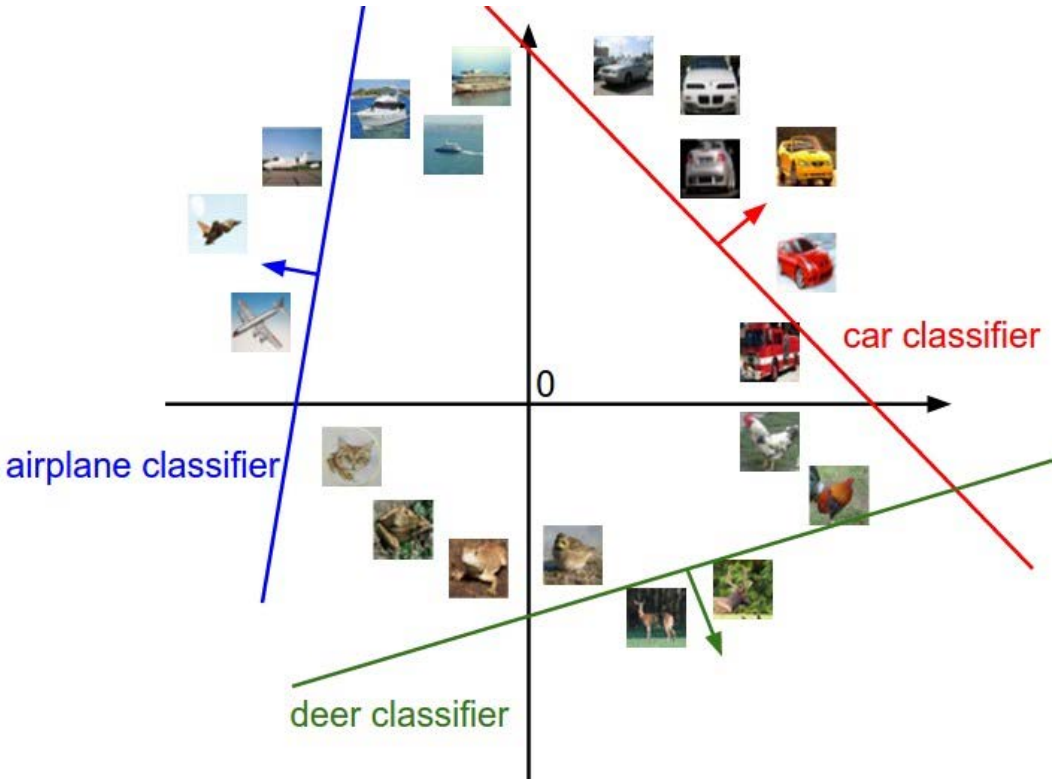
- **Linear classifier**
  - The score of a class as a weighted sum of all of its pixel values across all 3 of its color channels
  - Pixels $*$ Weights



Linear Classification

- **Linear classifier**
  - W: line in the pixel space will rotate in different directions
  - b: allow our classifiers to translate the lines

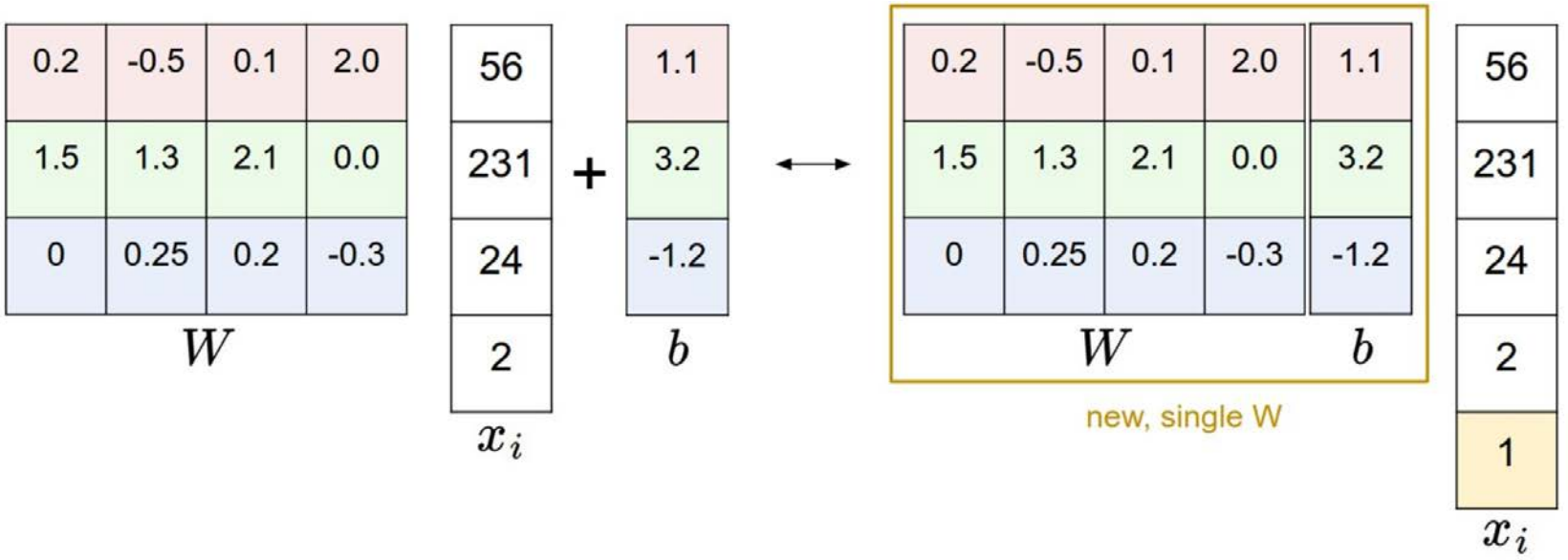
$$f(x_i, W, b) = Wx_i + b$$



Linear Classification

❖ Linear classifier

- Bias trick
  - The biases b and weights W 두가지 파라미터를 학습하기 까다로움
  - 하나의 파라미터로 Concatenation



Linear Classification

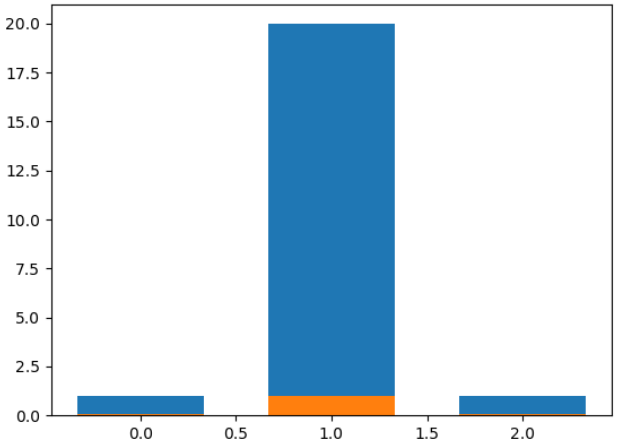
❖ **Image data preprocessing**

- Center your data by subtracting the mean from every feature
  - computing a mean image across the training images
  - subtracting it from every image to get images

이미지의 픽셀 값의 범위

[-127 … 127] ➡ [-1, 1]

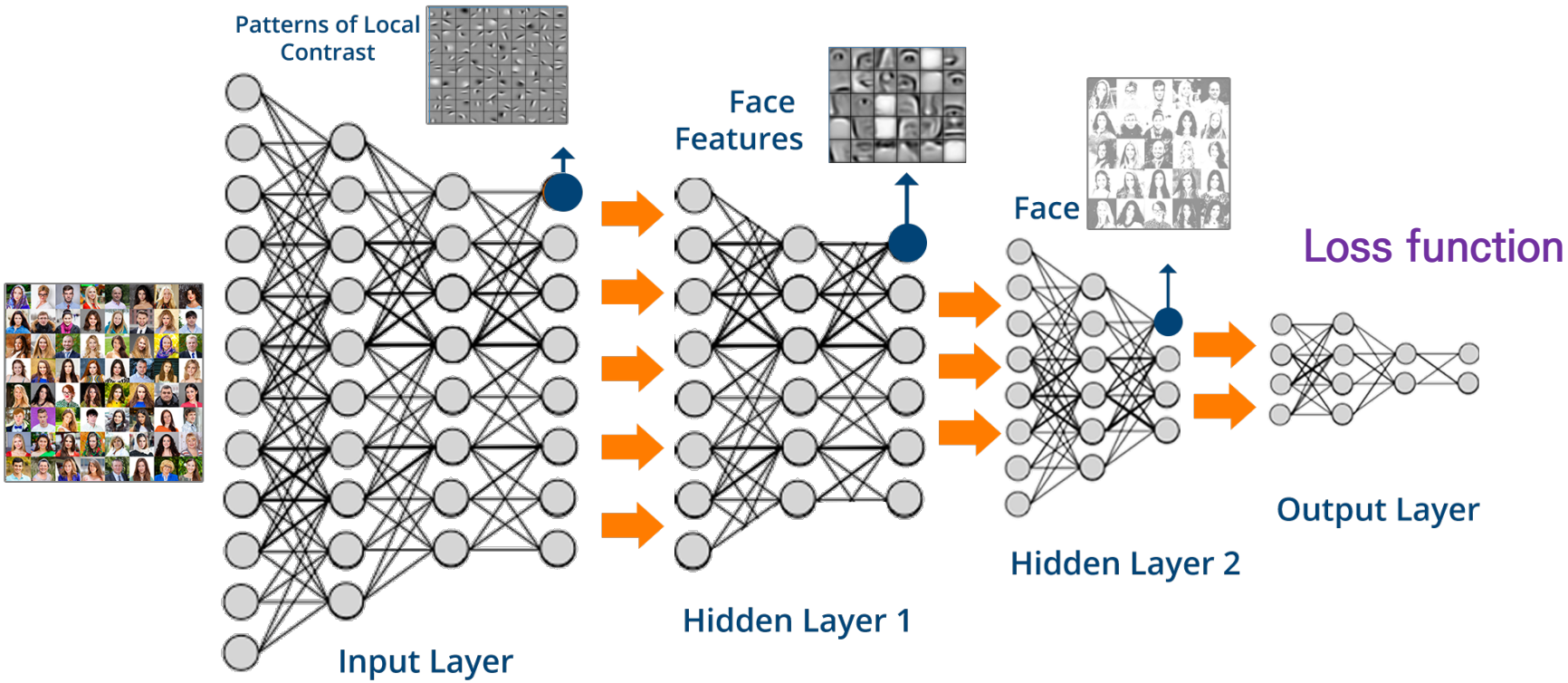
Normalization:
정당한 비교(그래디언트 전파) 가능




```
import matplotlib.pyplot as plt
examples = np.array([1,20,1])
N = len(examples)
x = range(N)
width = 1/1.5
plt.bar(x, examples,width)
plt.bar(x, examples/np.sqrt(sum(np.power(examples, 2))), width)
```

Contents

Loss functions



Loss functions

- **Multiclass SVM loss**
  - 2-class examples
  - 3-class examples
- **Softmax and cross-entropy with logits**
  - Logits
  - Softmax
  - Cross-entropy

Multiclass SVM loss

Loss functions

❖ **Multiclass SVM loss: (실제클래스 − 다른클래스) < Δ**

- Score = $s_j = f(x_i, W)_j$.
  - j번째 class 의 스코어
  - y_i= 실제 클래스

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Margin

실제클래스가 아닌 예측값

실제 값

Loss functions

❖ **Multiclass SVM loss: (실제클래스 − 다른클래스) < Δ**

- Score = $s_j = f(x_i, W)_j$.
  - j번째 class 의 스코어
  - y_i= 실제 클래스

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Margin

실제클래스가 아닌 예측값

실제 값

	Image #1	Image #2
Dog	4.26	3.76
Cat	1.33	-1.2

Image#1

Multi-class SVM Loss

```
>>> max(0, 1.33 - 4.26 + 1)
0
>>>
```

Image#2

Multi-class SVM Loss

```
>>> max(0, 3.76 - (-1.2) + 1)
5.96
>>>
```

Loss functions

❖ **Multiclass SVM loss : (실제클래스 − 다른클래스) < Δ**

- Score = $s_j = f(x_i, W)_j$.
  - j번째 class 의 스코어
  - y_i= 실제 클래스

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Margin

실제클래스가 아닌 예측값

실제 값

	Class1	Class2	Class3	True class
example1	13	−7	11	Class1
example2	…	…	…	…

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

2번째 Class

3번째 Class

Loss functions

❖ **Multiclass SVM loss : (실제클래스 − 다른클래스) < Δ**

- Score = $s_j = f(x_i, W)_j$.
  - j번째 class 의 스코어
  - y_i= 실제 클래스

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

실제클래스가 아닌 예측값 : s_j

실제 값 : s_{y_i}

Margin : Δ

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

For linear classification

	Class1	Class2	Class3	True class
example1	13	−7	11	Class1
example2	…	…	…	…

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

2번째 Class 3번째 Class

Softmax and cross-entropy with logits

Loss functions

❖ **Softmax and cross-entropy with logits**

- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

Loss functions

❖ **Softmax and cross-entropy with logits**

- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

f_{y_i} ← $f(x_i; W) = Wx_i$

f_j ← $f(x_i; W) = Wx_i$

Logit

클래스 예측확률

Softmax

Loss functions

❖ **Softmax and cross-entropy with logits**

- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret

$$P(y_i \mid x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

f_{y_i} ← $f(x_i; W) = Wx_i$

f_j ← $f(x_i; W) = Wx_i$

Logit

클래스 예측확률

⇕

클래스 실제확률

Cross-entropy 평가지표

클래스들의 불확실성의 합 / 예측된 클래스확률

$$H(p, q) = -\sum_x p(x) \log q(x)$$

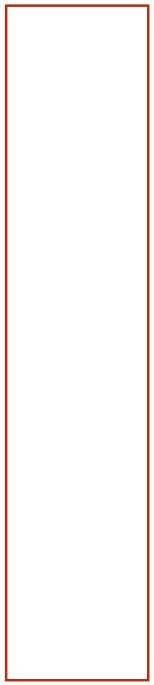
실제 클래스확률

Loss functions

❖ **Softmax and cross-entropy with logits**

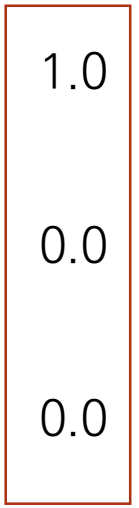
- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret

X(Input)



1-hot Labels

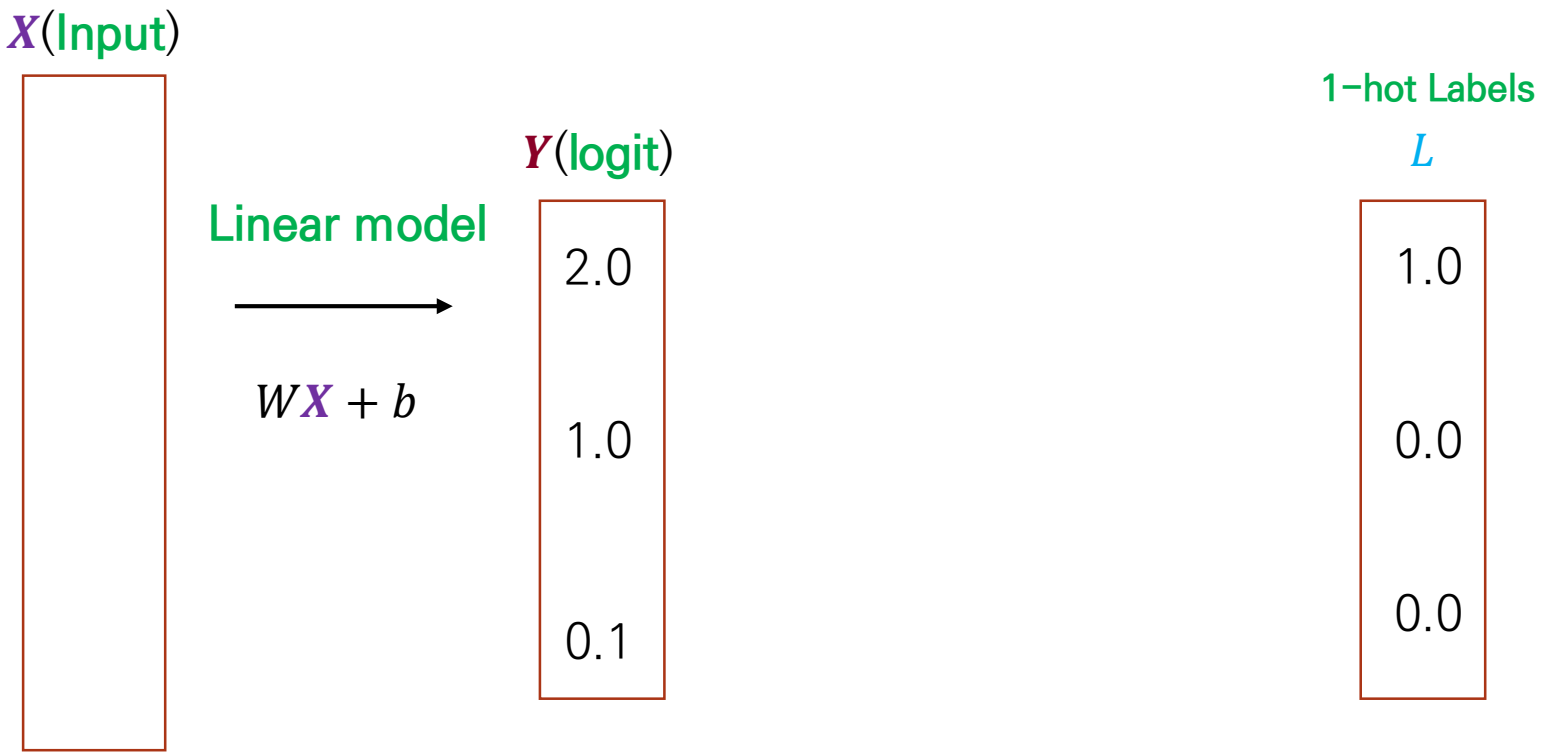
L



Loss functions

❖ Softmax and cross-entropy with logits

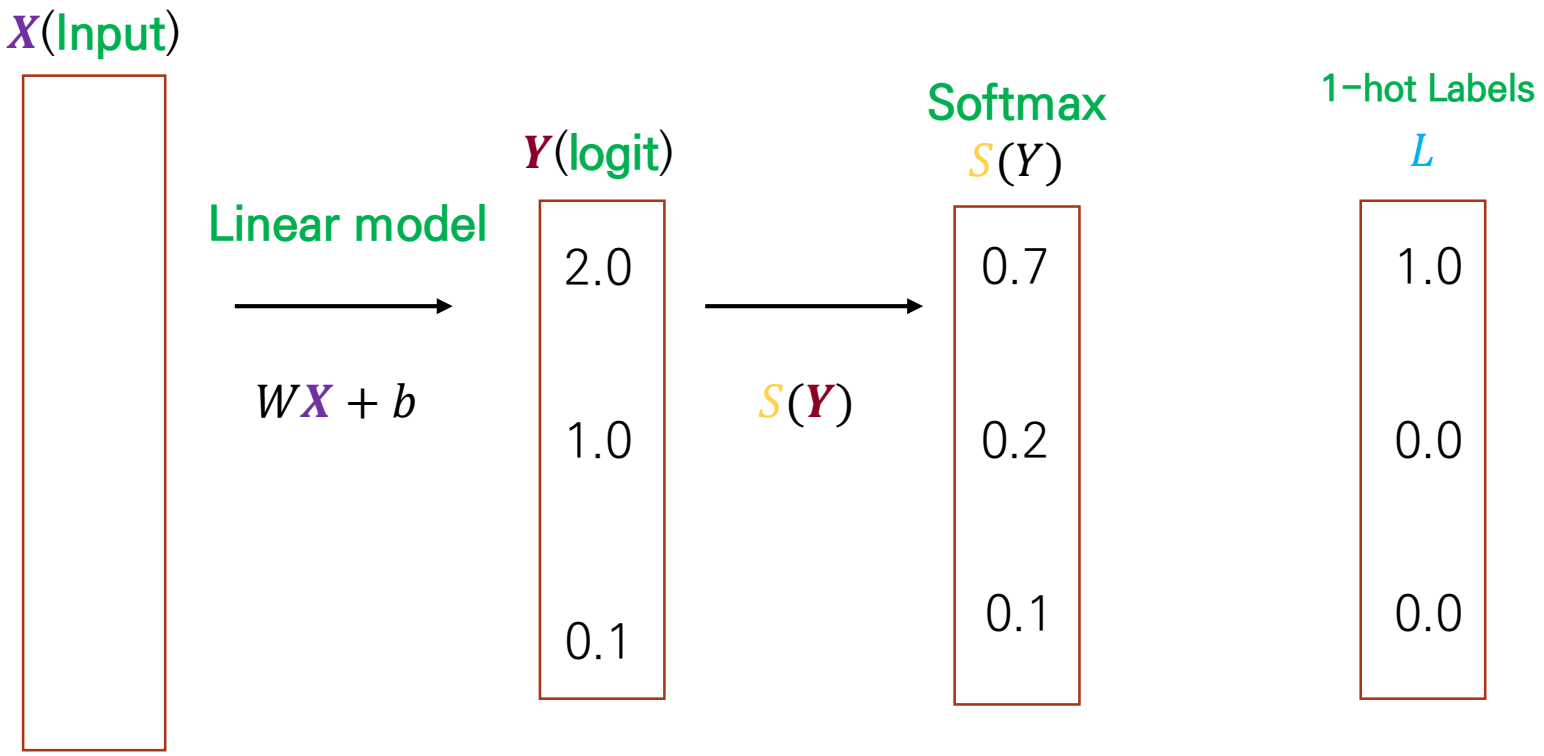
- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret



Loss functions

❖ Softmax and cross-entropy with logits

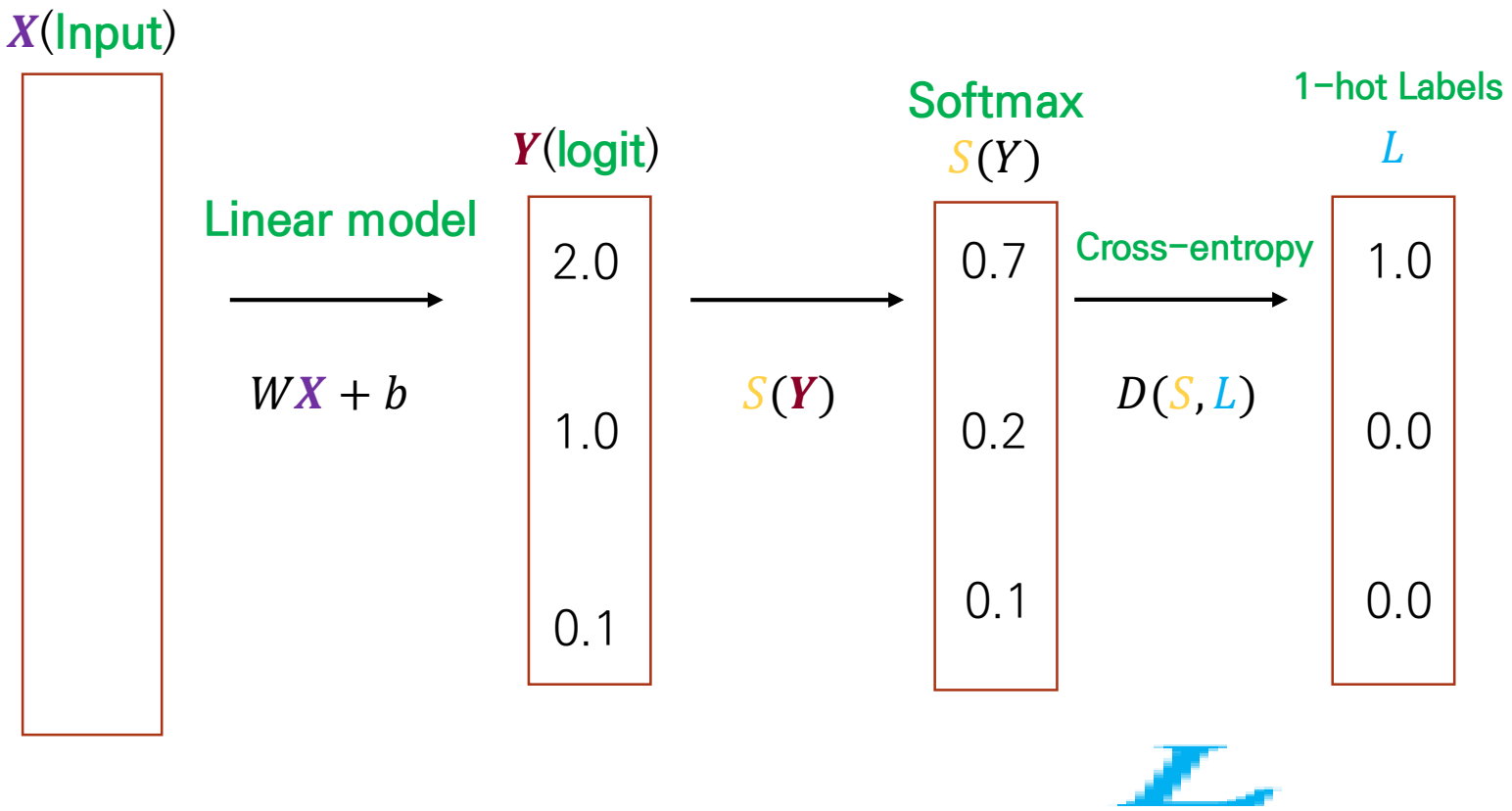
- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret



Loss functions

❖ Softmax and cross-entropy with logits

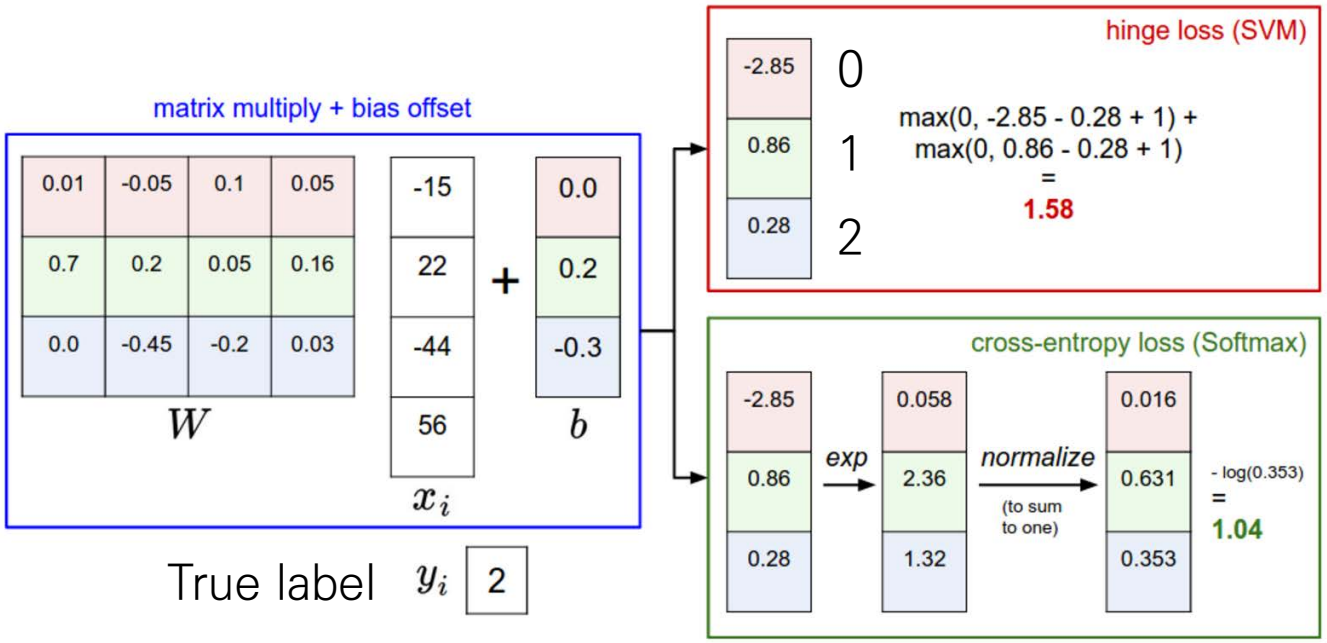
- Normalized class probabilities: Range(0~1) = probability
- possibly easy to interpret



Loss functions

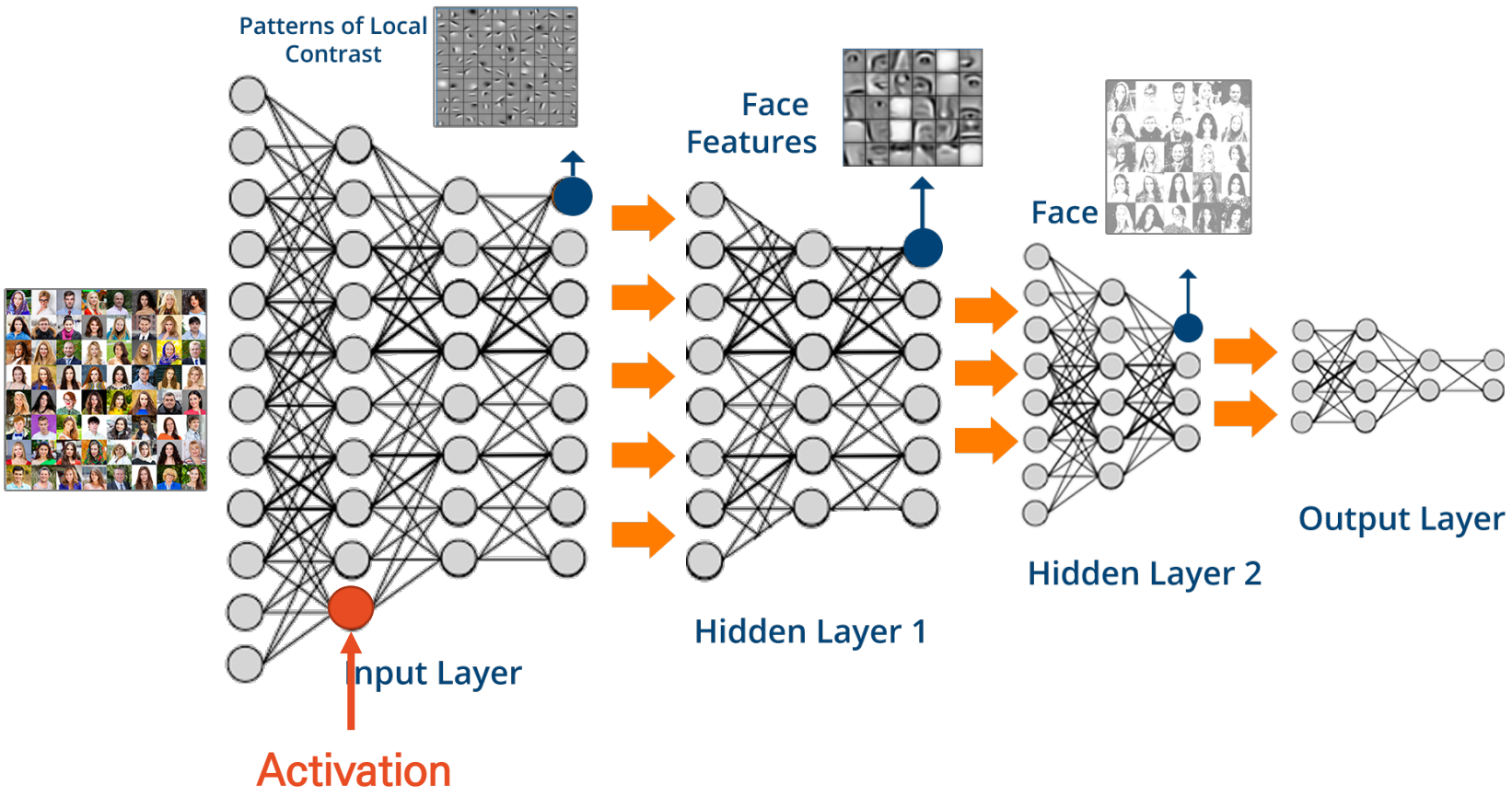
❖ SVM vs. Softmax

- 일반적으로 Softmax가 직관적이기 때문에 주로 사용
- 업데이트 관점
  - Softmax는 매 iteration 마다 업데이트
  - Hinge loss는 Loss가 0이면 업데이트를 하지 않음



Contents

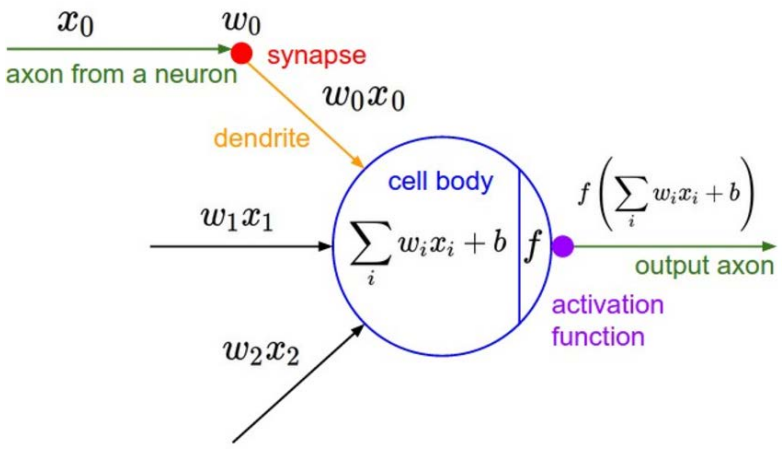
Activations

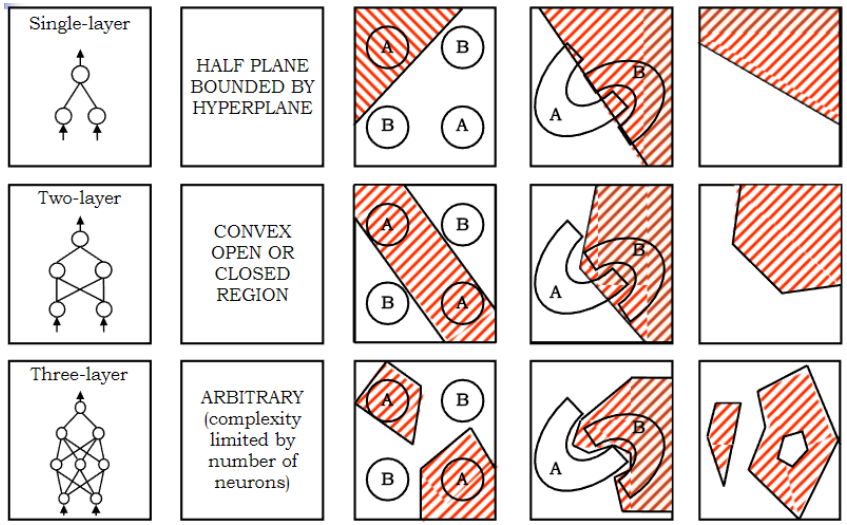


Activations

❖ Activation

- 모델의 non-linearity를 향상시키기 위한 활성함수
- Overfitting을 야기함





Activations

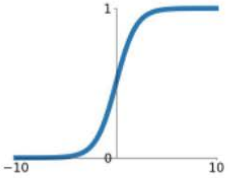
❖ **Activation functions**

- Overfitting을 방지하기 위해 최근에 제안된 다양한 activation function들
- ReLU: sigmoid 함수보다 약 6배 빠른 학습 속도
- Maxout: 너무 많은 파라미터를 가짐

Activation Functions

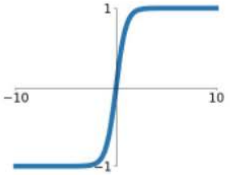
Sigmoid

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



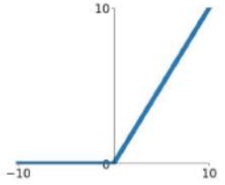
tanh

$$\tanh(x)$$



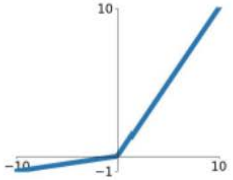
ReLU

$$\max(0, x)$$



Leaky ReLU

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

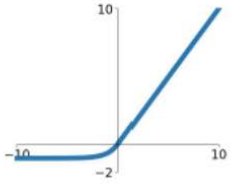


Maxout

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

ELU

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



Contents

Backpropagation

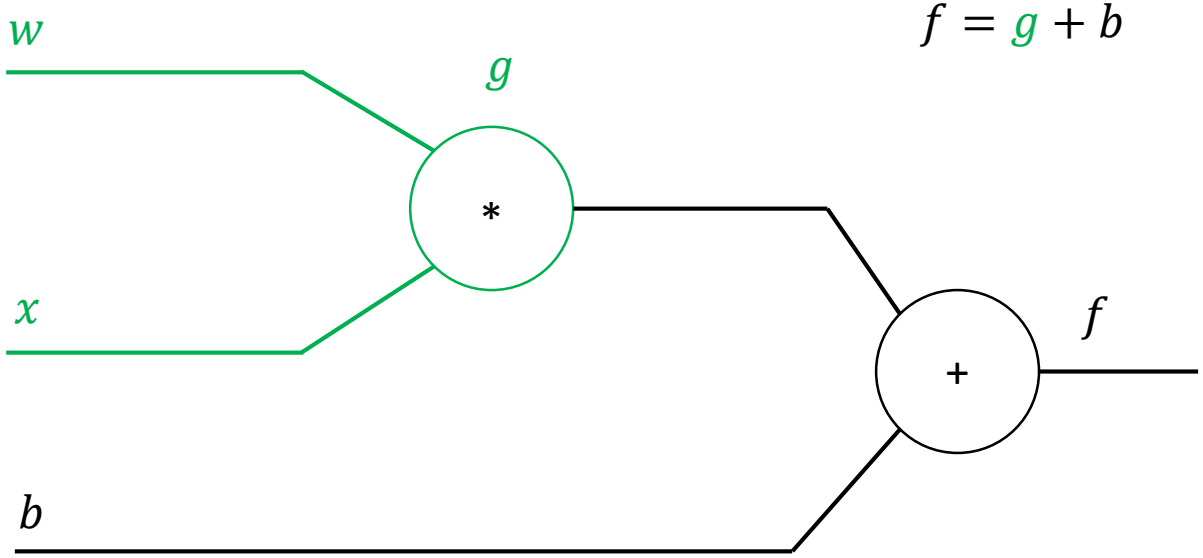
❖ Backpropagation

- 목적함수에 미치는 영향

$$f = wx + b$$

$$g = wx$$

$$f = g + b$$



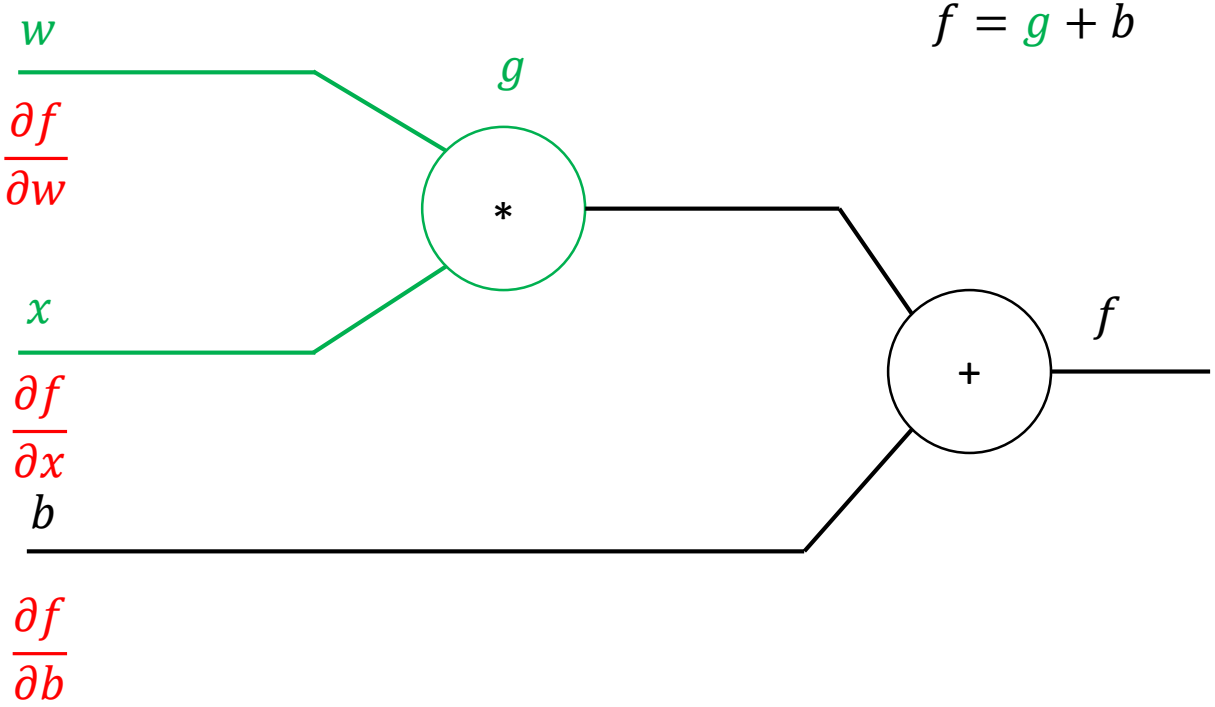
Backpropagation

❖ Backpropagation
- 목적함수에 미치는 영향을 이용
- 쉽게 최적화 문제를 풀 수 잇음

$$f = wx + b$$

$$g = wx$$

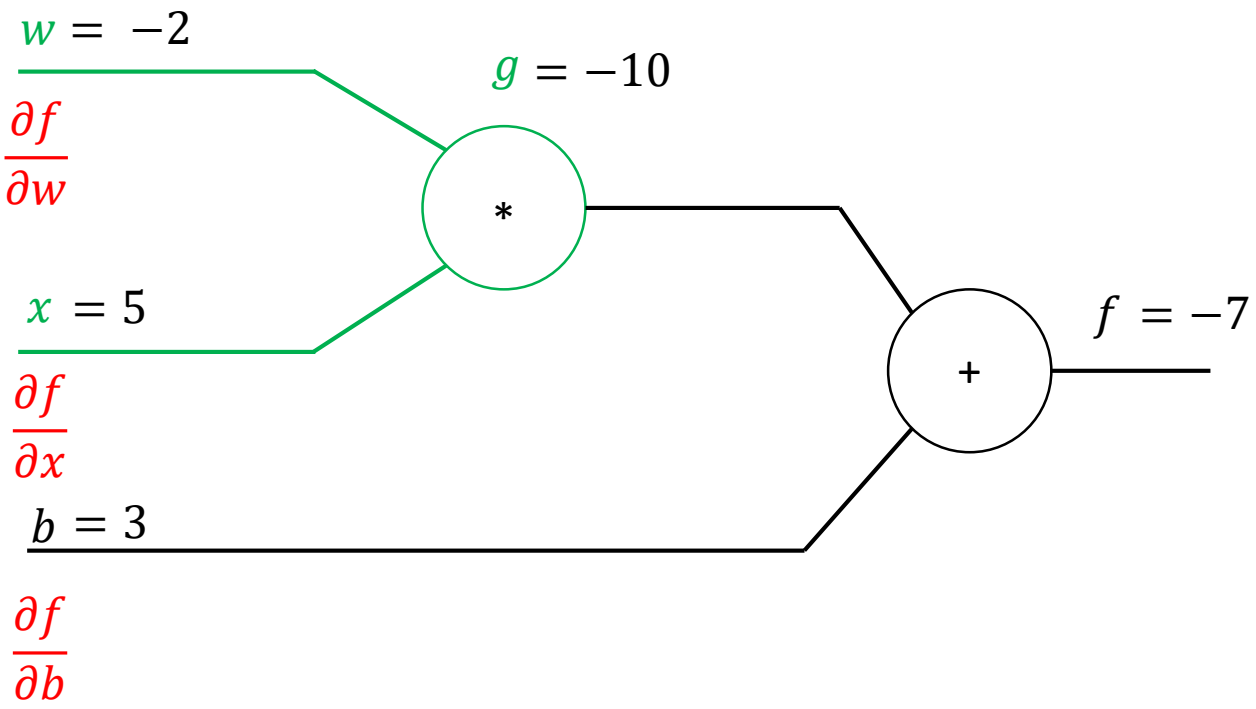
$$f = g + b$$



Backpropagation

- **Backpropagation**
  - Step 01) Forward
    - $[w = -2\,,\ x = 5,\ b = 3]$

$$f = wx + b \qquad g = wx \qquad f = g + b$$



Backpropagation

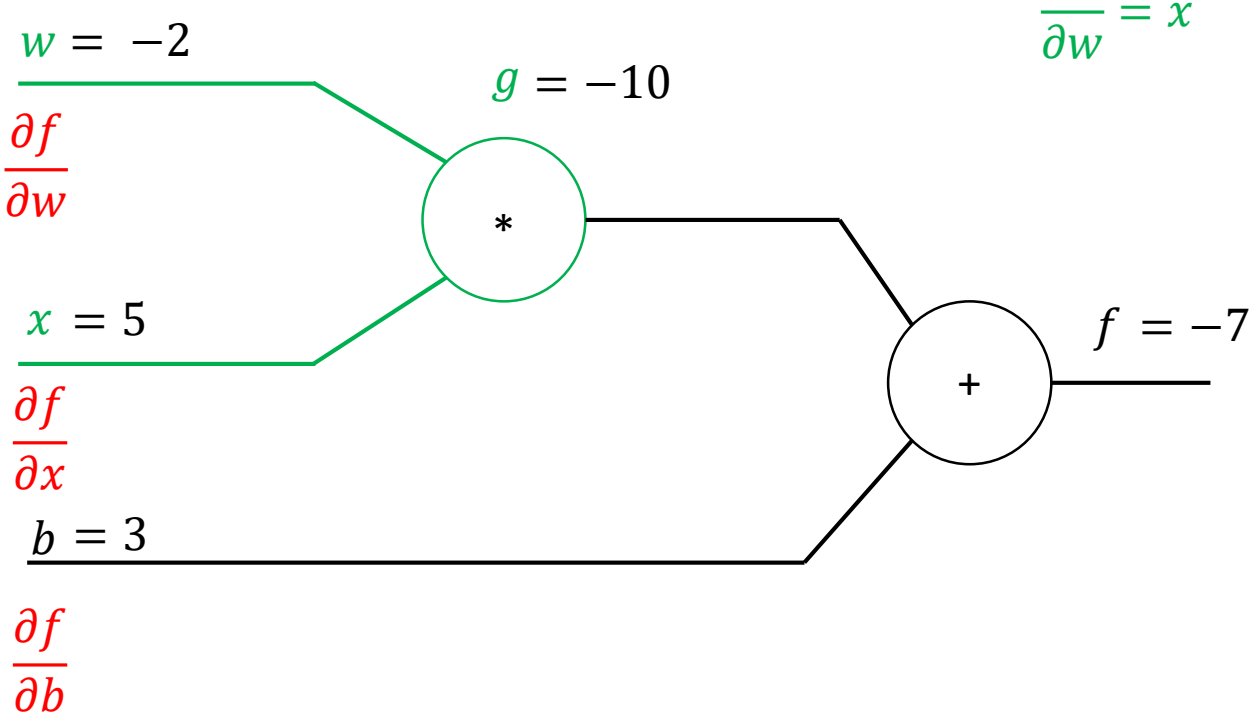
❖ **Backpropagation**

- Step 01) partial derivative
  - $[w = -2\,,\ x = 5,\ b = 3]$

$$\frac{\partial f}{\partial g} = 1 \qquad \frac{\partial f}{\partial b} = 1$$

$$f = wx + b \qquad g = wx \qquad f = g + b$$

$$\frac{\partial g}{\partial w} = x \qquad \frac{\partial g}{\partial x} = w$$



Backpropagation

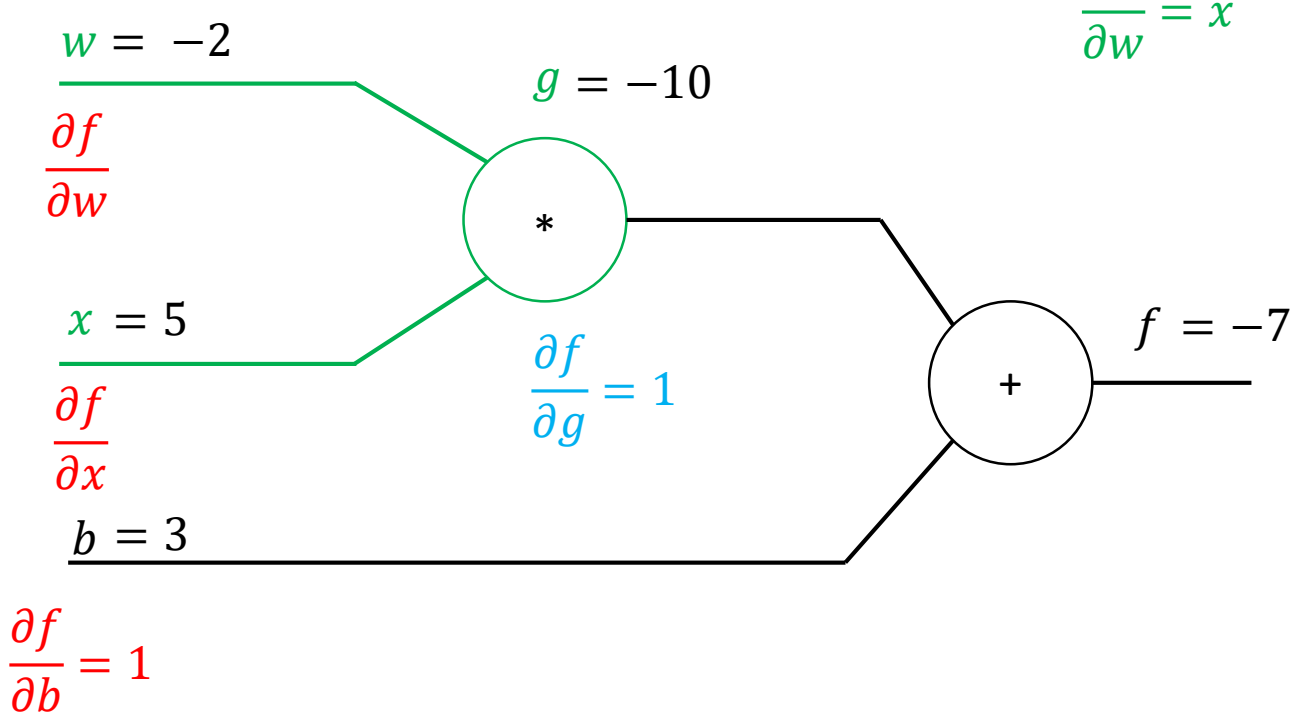
❖ **Backpropagation**

- Step 01) partial derivative
  - $[w = -2\,,\ x = 5,\ b = 3]$

$$\frac{\partial f}{\partial g} = 1 \qquad \frac{\partial f}{\partial b} = 1$$

$$f = wx + b \qquad g = wx \qquad f = g + b$$

$$\frac{\partial g}{\partial w} = x \qquad \frac{\partial g}{\partial x} = w$$



Backpropagation

❖ **Backpropagation**

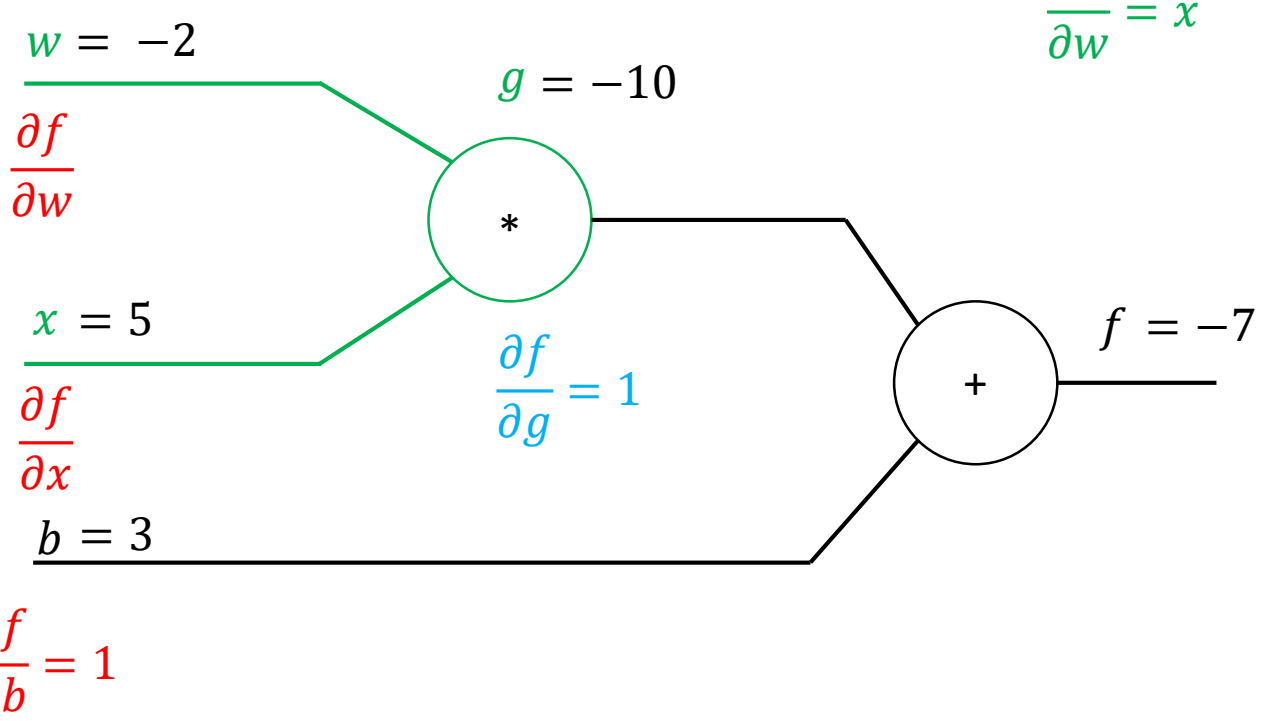
- Step 01) Chain rule
  - $[w = -2\,,\ x = 5,\ b = 3]$

$$\frac{\partial f}{\partial g} = 1 \qquad \frac{\partial f}{\partial b} = 1$$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial w}$$

$$f = wx + b \qquad g = wx \qquad f = g + b$$

$$\frac{\partial g}{\partial w} = x \qquad \frac{\partial g}{\partial x} = w$$



Backpropagation

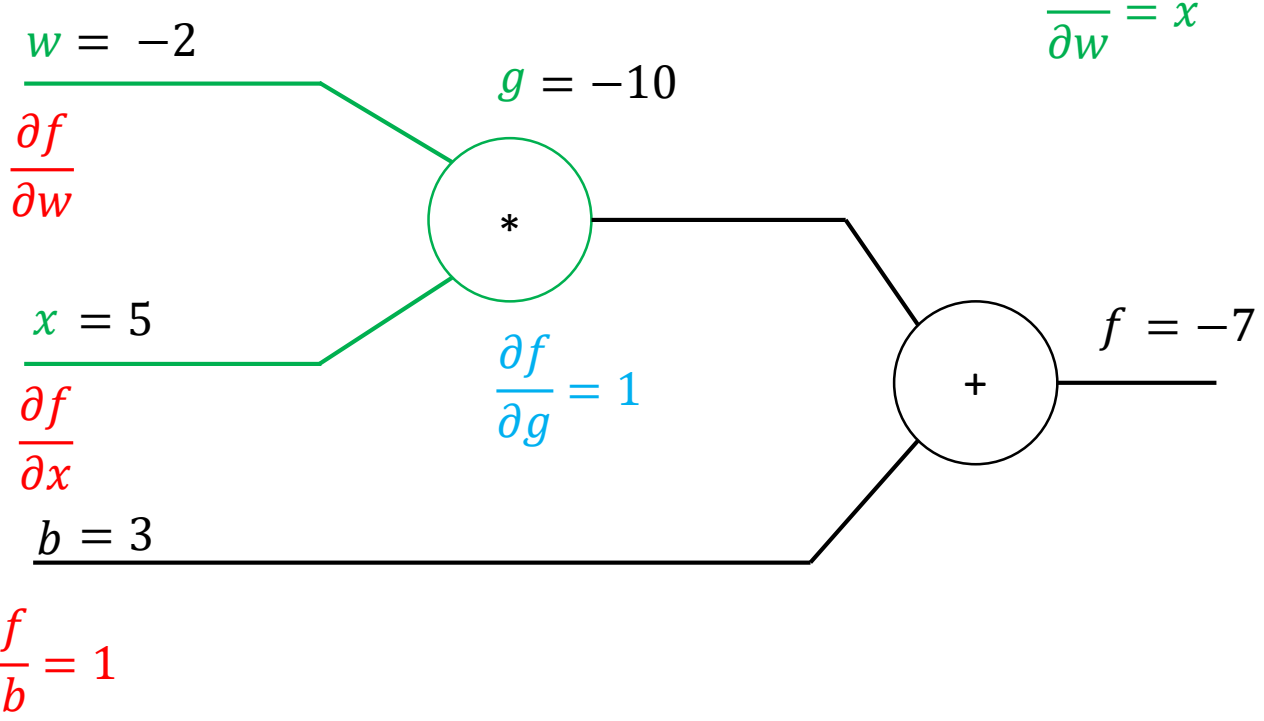
- **Backpropagation**
  - Chain rule
    - $[w = -2\,,\ x = 5,\ b = 3]$

$$\frac{\partial f}{\partial g} = 1 \qquad \frac{\partial f}{\partial b} = 1$$

$$f = wx + b \qquad g = wx \qquad f = g + b$$

$$\frac{\partial g}{\partial w} = x \qquad \frac{\partial g}{\partial x} = w$$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial w} = 1 * x = 1 * 5 = 5$$



Backpropagation

- **Backpropagation**
  - Chain rule
    - $[w = -2\,,\ x = 5,\ b = 3]$

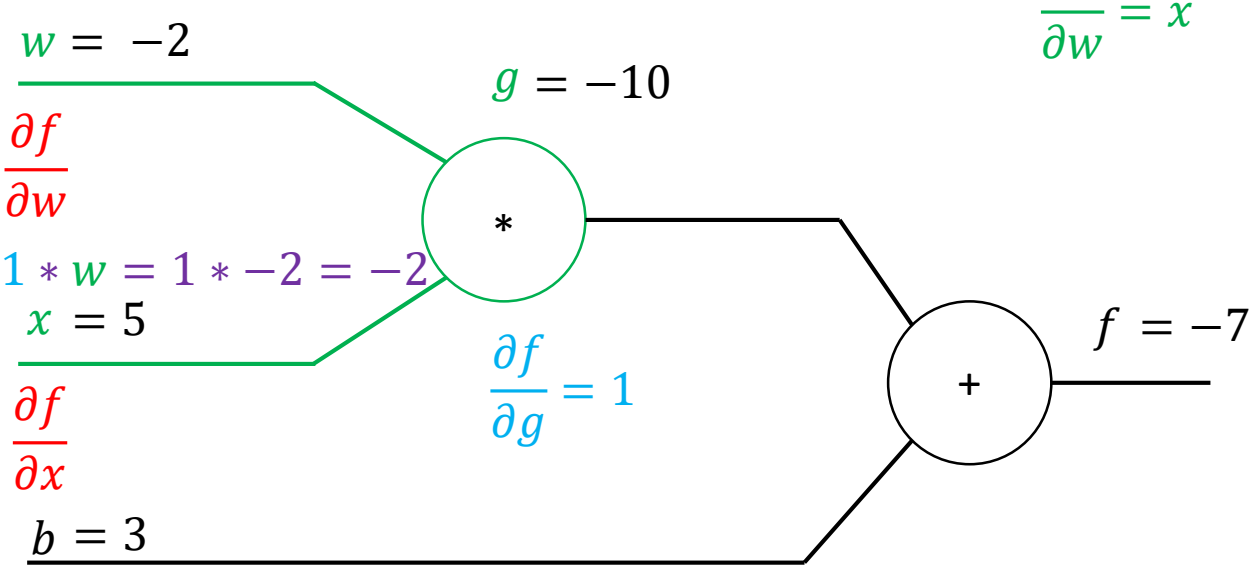
$$\frac{\partial f}{\partial g} = 1 \qquad \frac{\partial f}{\partial b} = 1$$

$$f = wx + b \qquad g = wx \qquad f = g + b$$

$$\frac{\partial g}{\partial w} = x \qquad \frac{\partial g}{\partial x} = w$$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial w} = 1 * x = 1 * 5 = 5$$

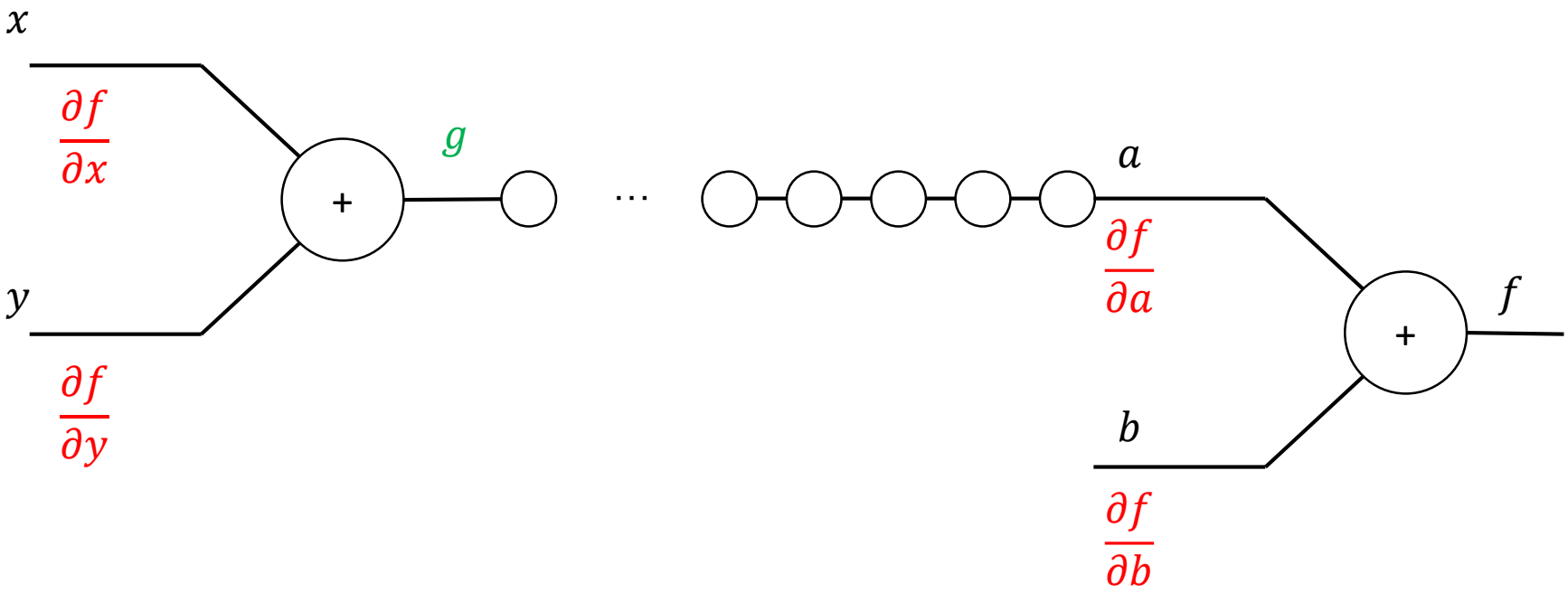
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial x} = 1 * w = 1 * -2 = -2$$



$$\frac{\partial f}{\partial b} = 1$$

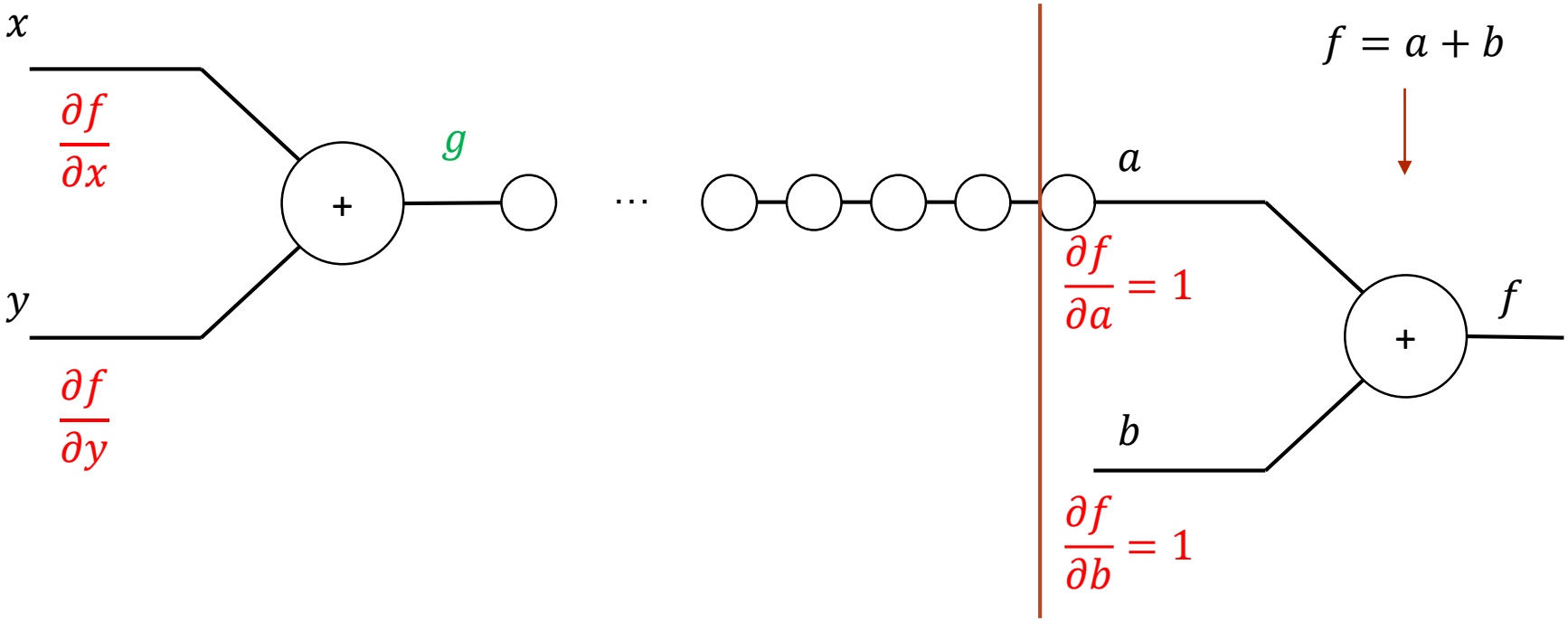
Backpropagation

- **Backpropagation**
  - Backward
    - $[w = -2\,,\ x = 5,\ b = 3]$



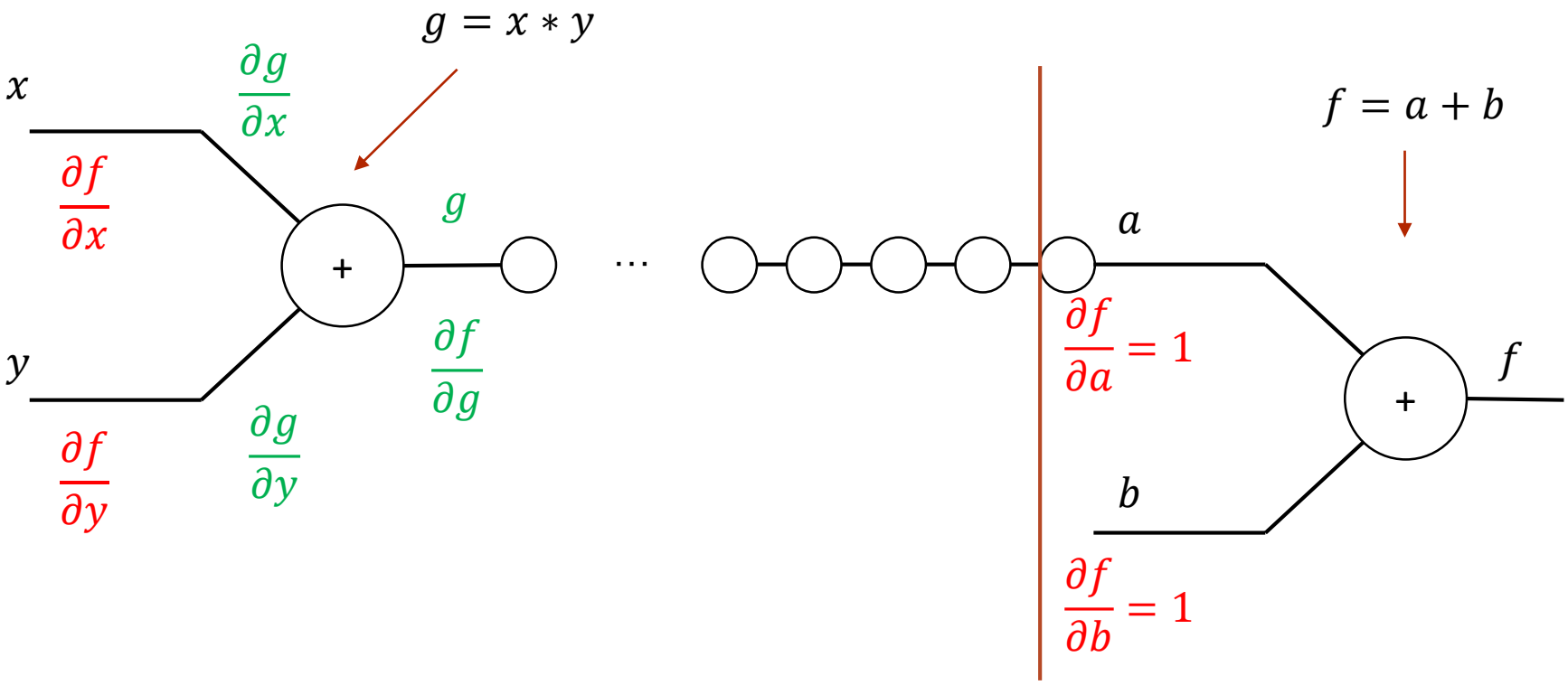
Backpropagation

- **Backpropagation**
  - Backward
    - $[w = -2\,,\ x = 5,\ b = 3]$



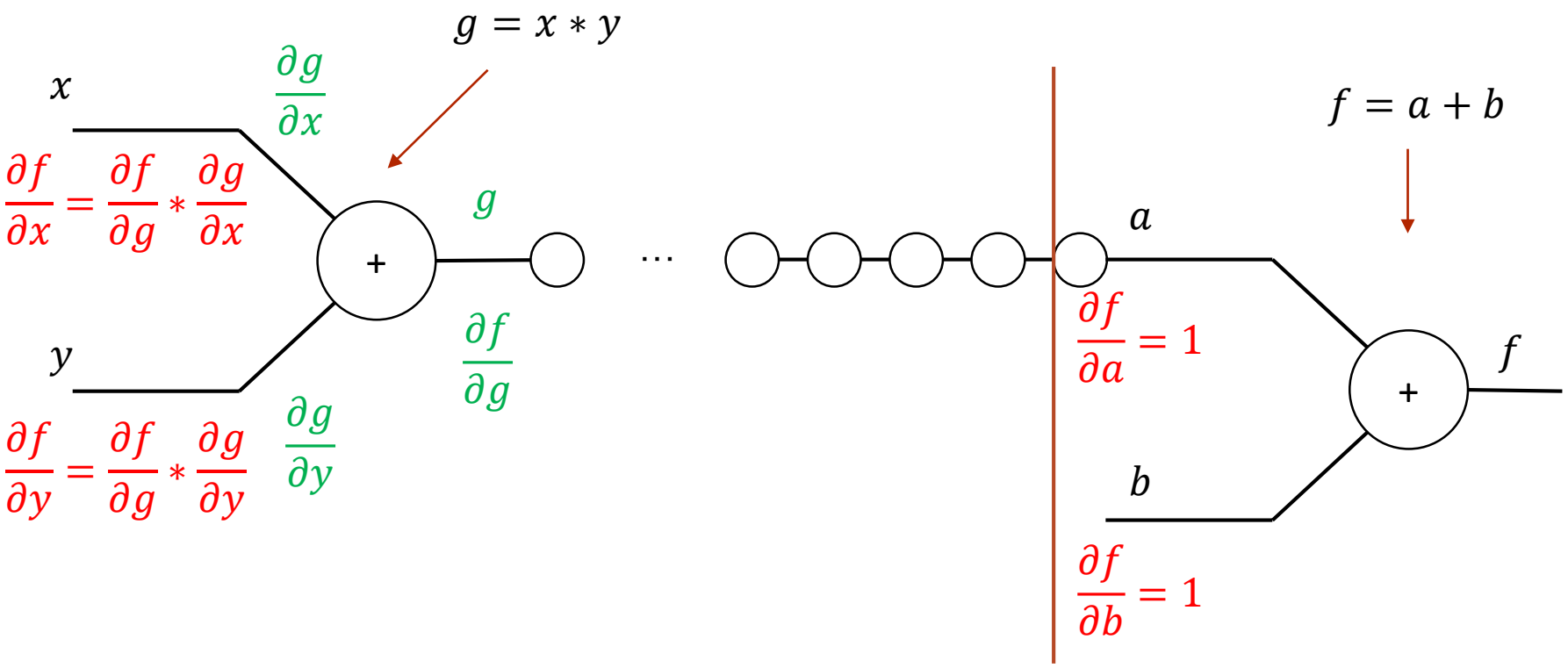
Backpropagation

- **Backpropagation**
  - Backward
    - $[w = -2\,,\ x = 5,\ b = 3]$



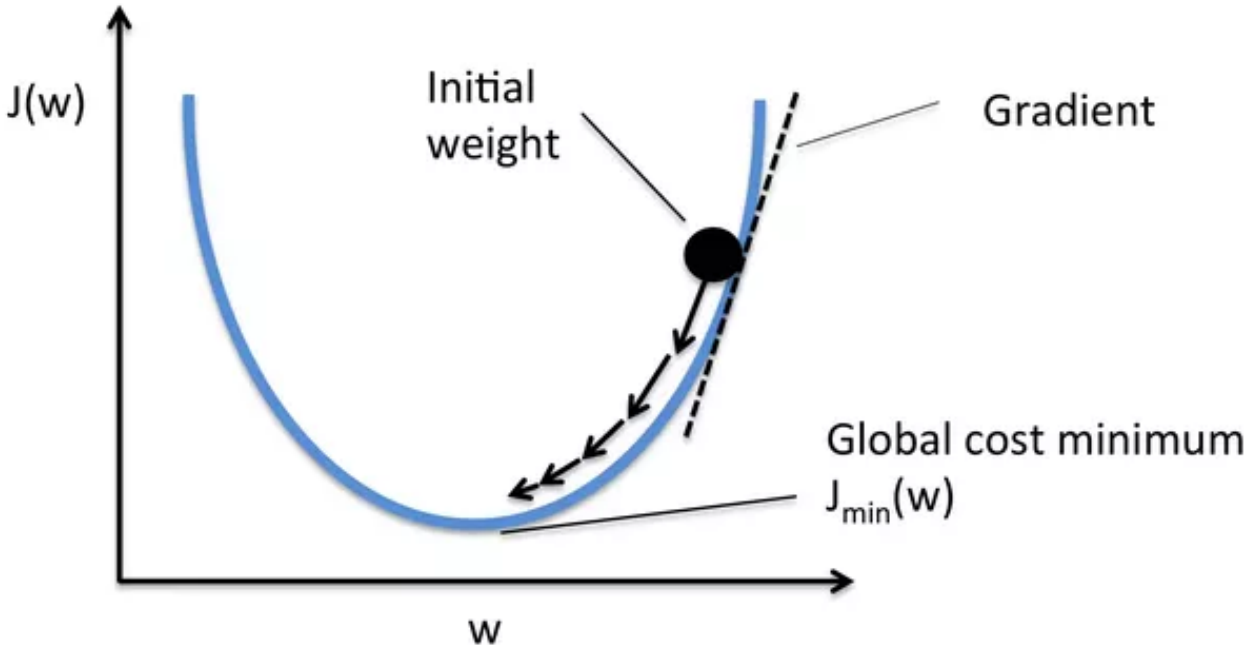
Backpropagation

- **Backpropagation**
  - Backward
    - $[w = -2\,,\ x = 5,\ b = 3]$



Backpropagation

- **Backpropagation**
  - Step 04) Update

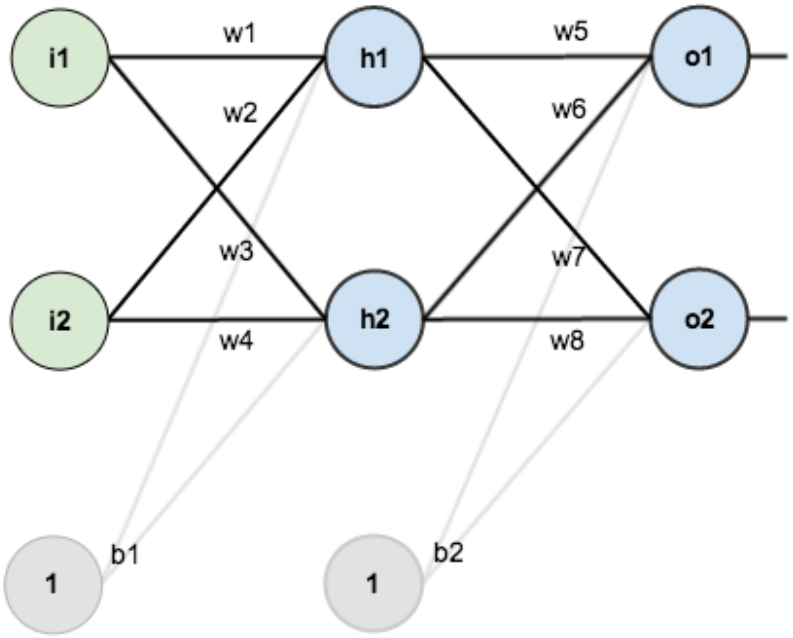


$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j}$$

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$$

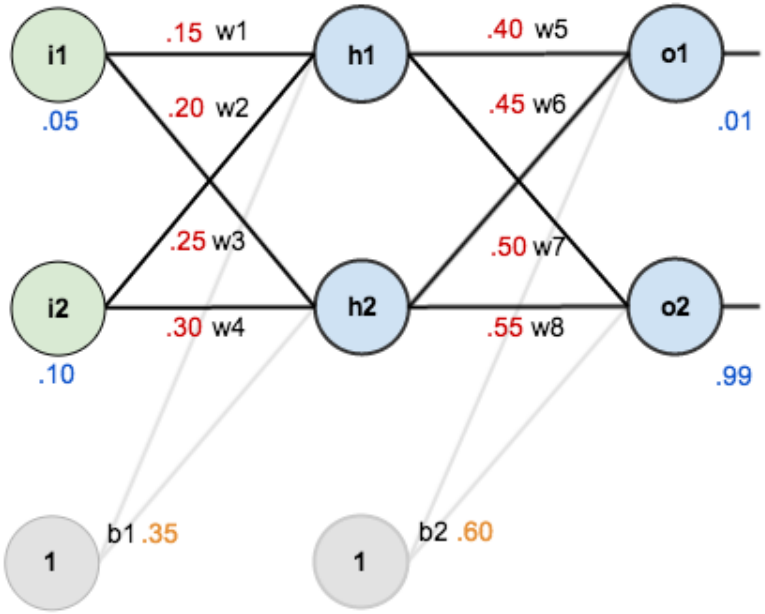
Backpropagation

- An example
  - Basic structure



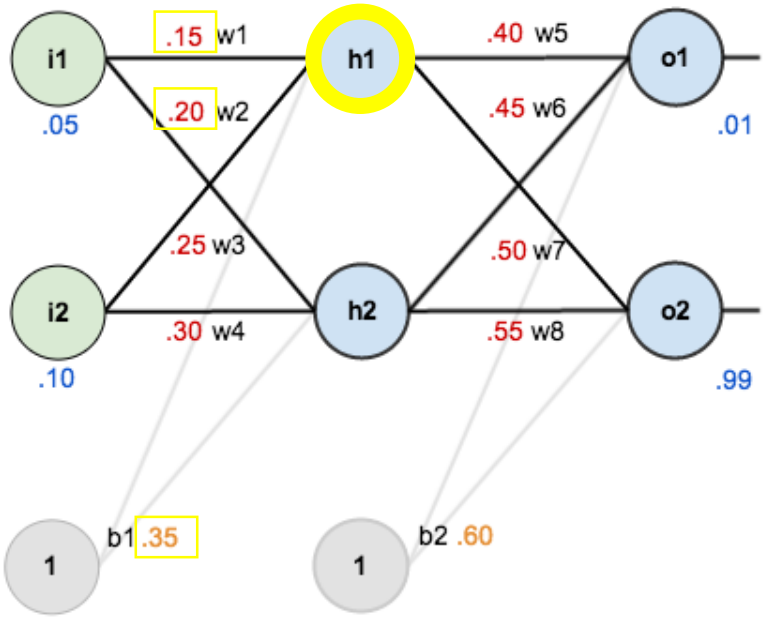
Backpropagation

- **An example**
  - Step01) Initialize weights, biases



Backpropagation

- **An example**
  - Step02) Forward pass

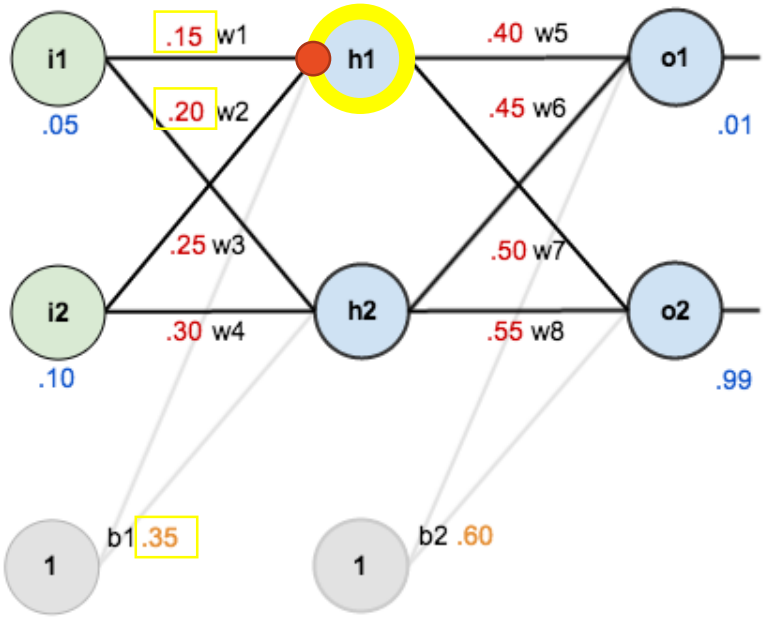


$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

Backpropagation

❖ An example

- Step02) Forward pass

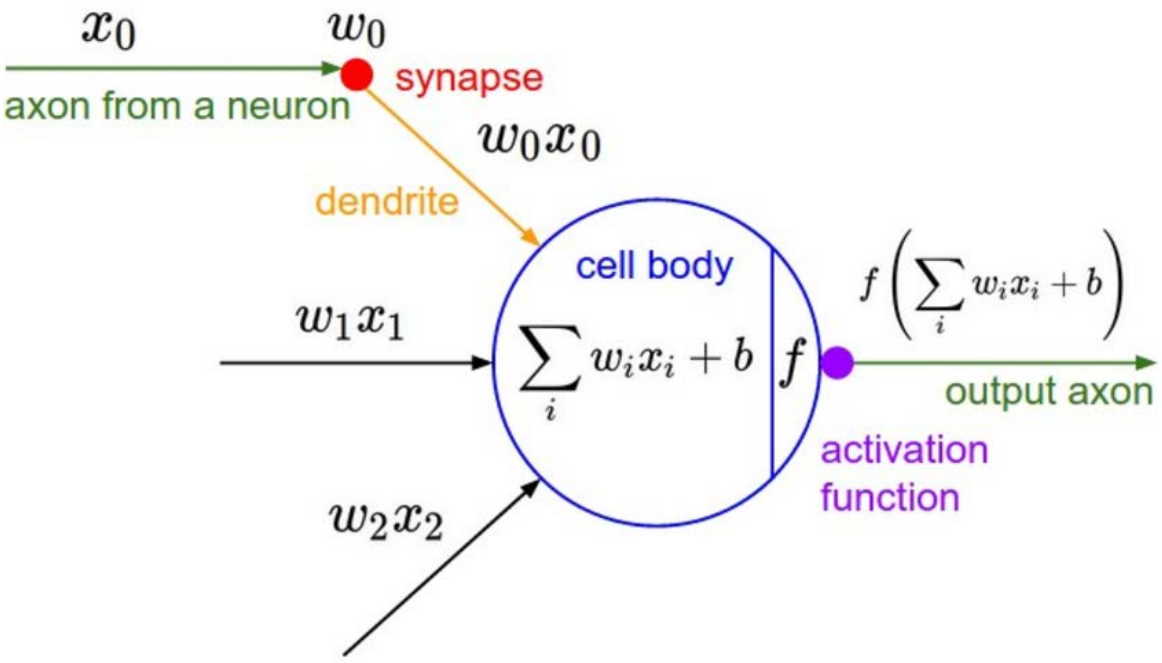


$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1$$
$$= 0.3775$$

Backpropagation

❖ An example

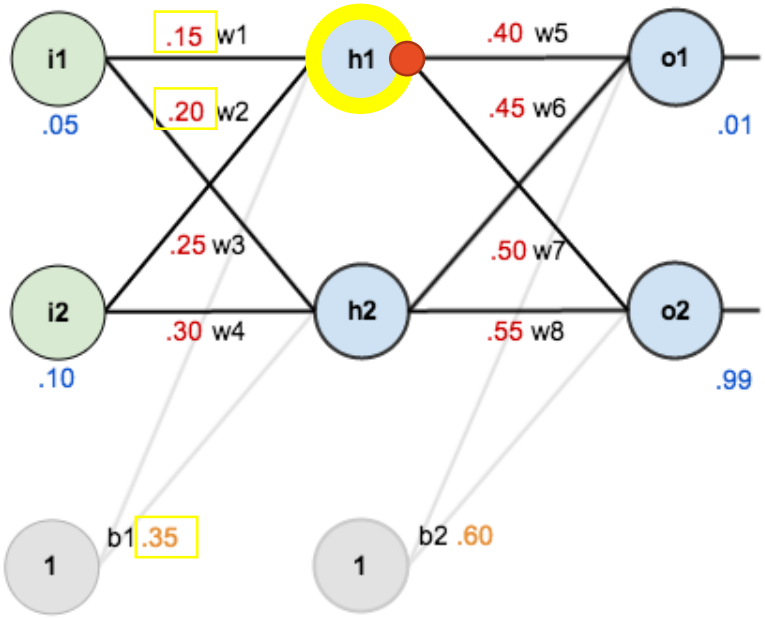
- Step02) Forward pass
  - Activation



Backpropagation

❖ An example

- Step02) Forward pass
  - Sigmoid 함수 사용



$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

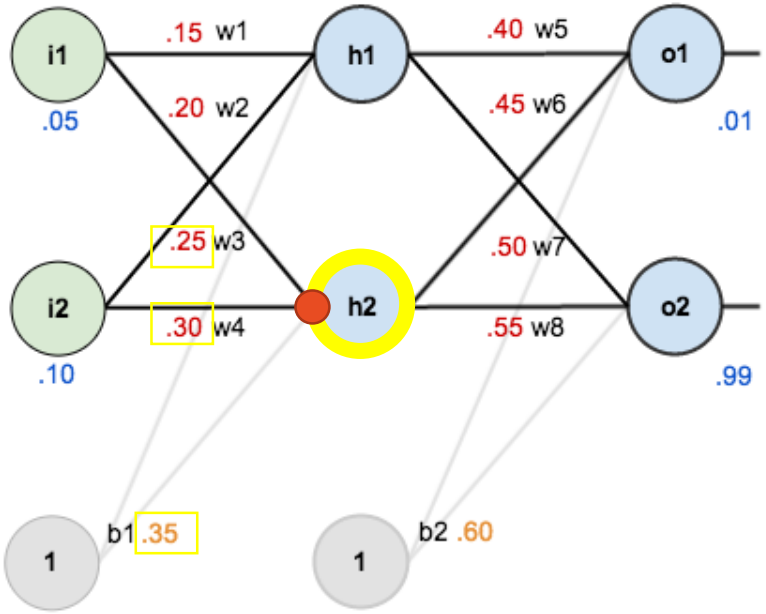
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1$$

$$= 0.3775$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.5932$$

Backpropagation

- **An example**
  - Step02) Forward pass

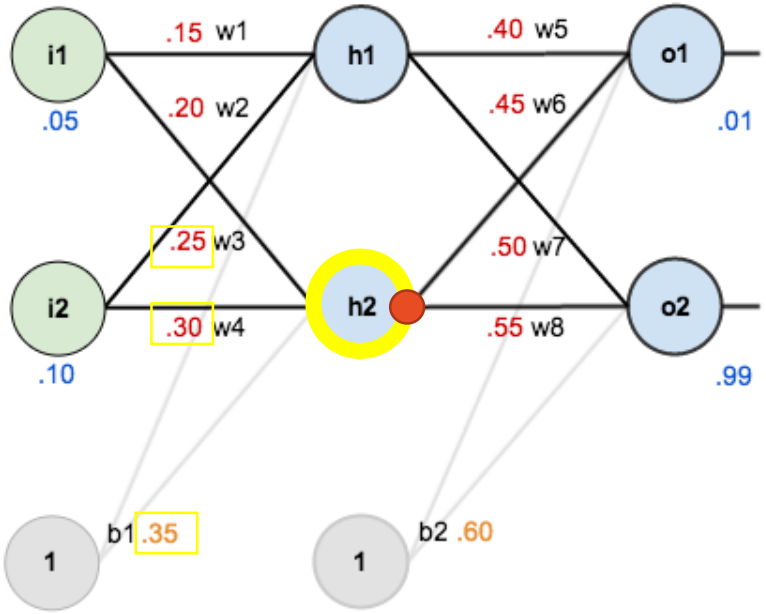


$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1$$
$$= 0.3775$$
$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.5932$$

$$net_{h2} = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$
$$net_{h2} = 0.25 * 0.05 + 0.3 * 0.1 + 0.35 * 1$$
$$= 0.3925$$

Backpropagation

- **An example**
  - Step02) Forward pass

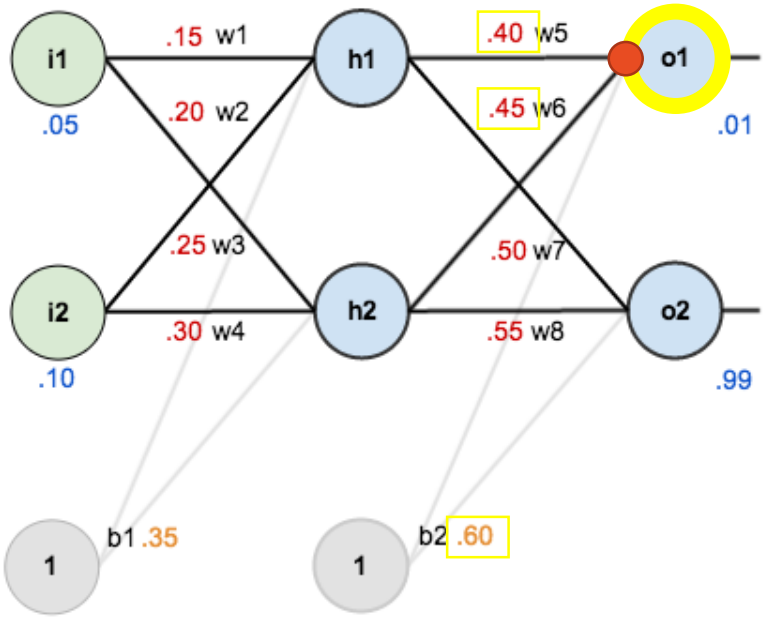


$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1$$
$$= 0.3775$$
$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}} = \frac{1}{1 + e^{-0.3775}} = 0.5932$$

$$net_{h2} = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$
$$net_{h2} = 0.25 * 0.05 + 0.3 * 0.1 + 0.35 * 1$$
$$= 0.3925$$
$$out_{h2} = \frac{1}{1 + e^{-net_{h2}}} = \frac{1}{1 + e^{-0.3925}} = 0.5969$$

Backpropagation

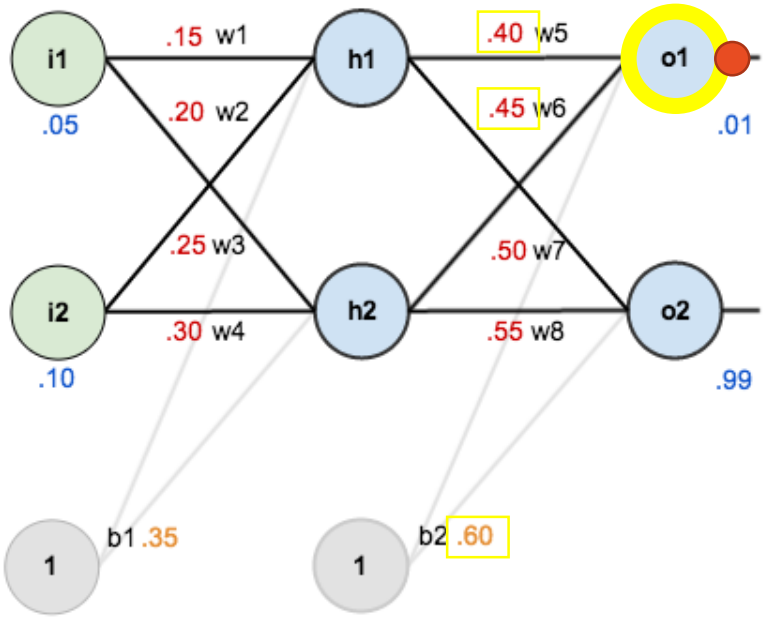
- **An example**
  - Step02) Forward pass



$$net_{o1} = w_5 * h_1 + w_6 * h_2 + b_2 * 1$$
$$net_{o1} = 0.4 * 0.5932 + 0.45 * 0.5969 + 0.6 * 1$$
$$= 1.106$$

Backpropagation

- An example
  - Step02) Forward pass



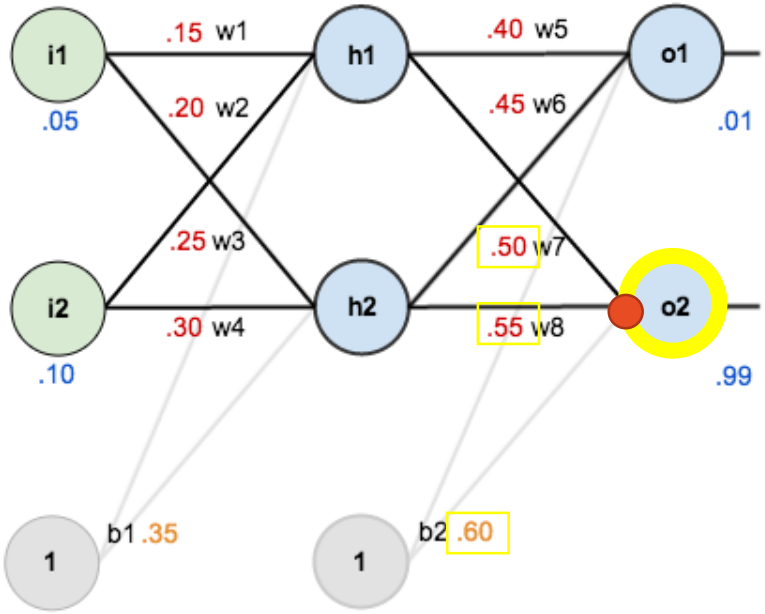
$$net_{o1} = w_5 * h_1 + w_6 * h_2 + b_2 * 1$$

$$net_{o1} = 0.4 * 0.5932 + 0.45 * 0.5969 + 0.6 * 1 = 1.106$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.106}} = 0.7514$$

Backpropagation

- An example
  - Step02) Forward pass

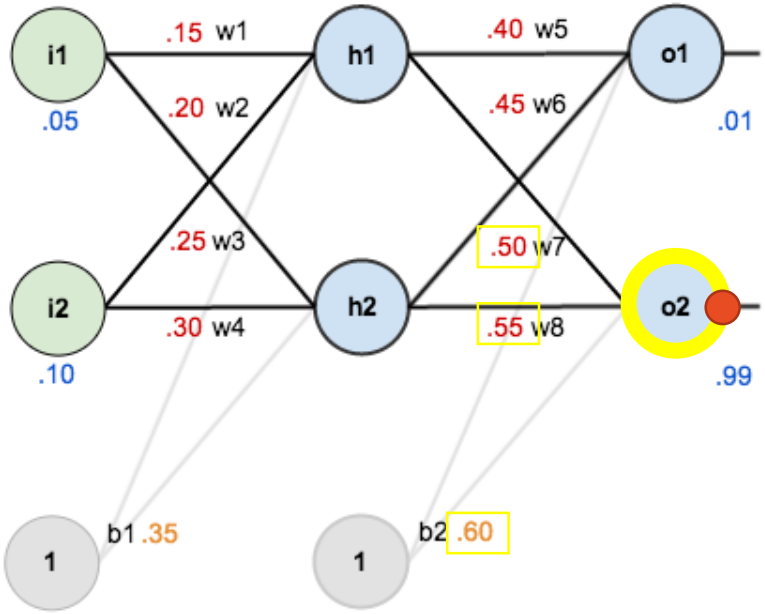


$$net_{o1} = w_5 * h_1 + w_6 * h_2 + b_2 * 1$$
$$net_{o1} = 0.4 * 0.5932 + 0.45 * 0.5969 + 0.6 * 1 = 1.106$$
$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.106}} = 0.7514$$

$$net_{o2} = w_7 * h_1 + w_8 * h_2 + b_2 * 1$$
$$net_{o2} = 0.5 * 0.5932 + 0.55 * 0.5969 + 0.6 * 1 = 1.2249$$

Backpropagation

- An example
  - Step02) Forward pass

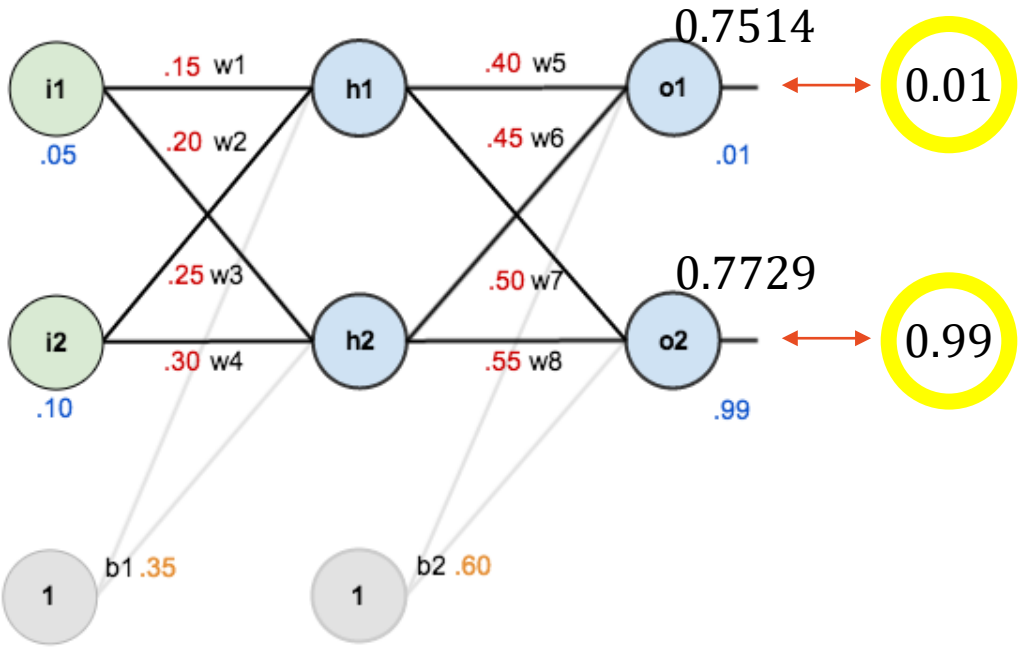


$$net_{o1} = w_5 * h_1 + w_6 * h_2 + b_2 * 1$$
$$net_{o1} = 0.4 * 0.5932 + 0.45 * 0.5969 + 0.6 * 1 = 1.106$$
$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}} = \frac{1}{1 + e^{-1.106}} = 0.7514$$

$$net_{o2} = w_7 * h_1 + w_8 * h_2 + b_2 * 1$$
$$net_{o2} = 0.5 * 0.5932 + 0.55 * 0.5969 + 0.6 * 1 = 1.2249$$
$$out_{o2} = \frac{1}{1 + e^{-net_{o2}}} = \frac{1}{1 + e^{-1.2249}} = 0.7729$$

Backpropagation

- An example
  - Step03) Calculating the Total Error



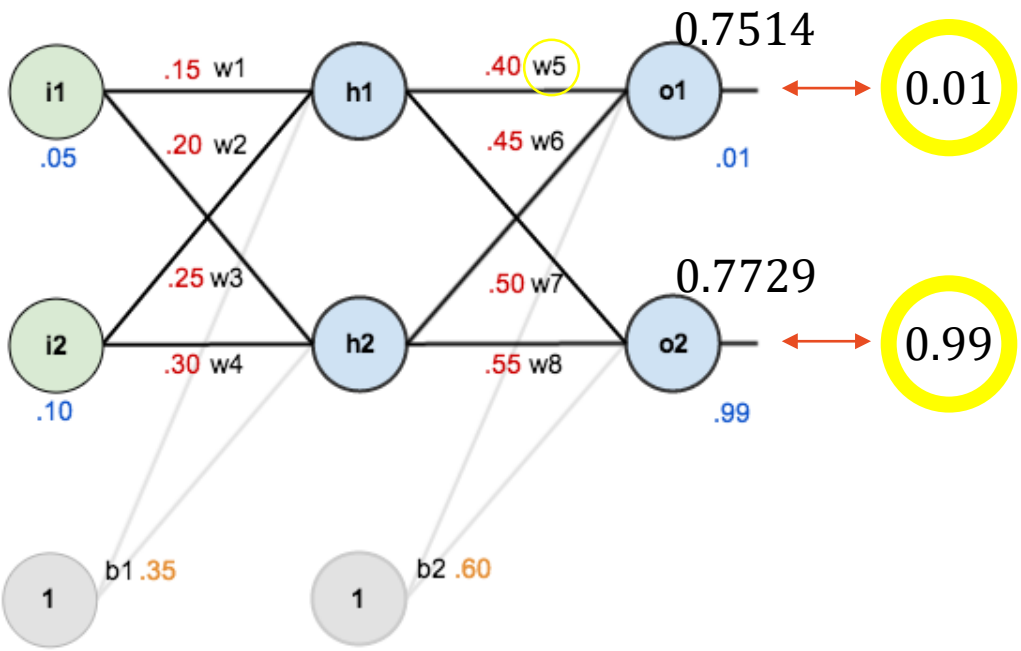
$$E_{o1} = \frac{1}{2}\left(target_{o_1} - out_{o_1}\right)^2$$
$$= \frac{1}{2}(0.01 - 0.7614)^2 = 0.2748$$

$$E_{o2} = \frac{1}{2}\left(target_{o_2} - out_{o_2}\right)^2$$
$$= \frac{1}{2}(0.99 - 0.7729)^2 = 0.0237$$

$$E_{Total} = E_{o_1} + E_{o2}$$
$$= 0.2748 + 0.0237 = 0.2985$$

Backpropagation

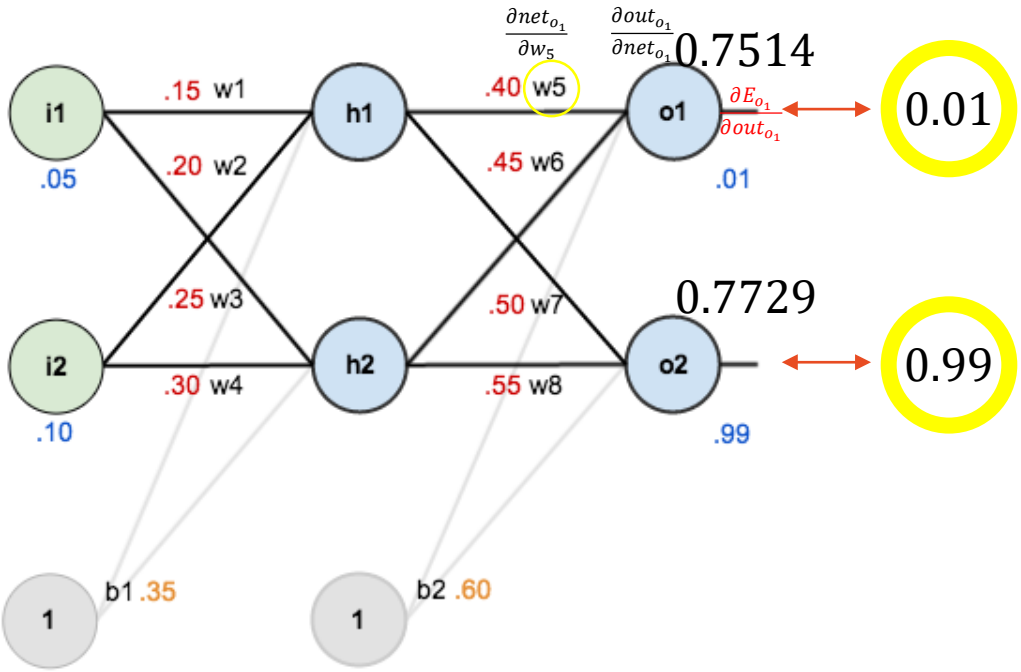
- An example
  - Step04) Backward Pass



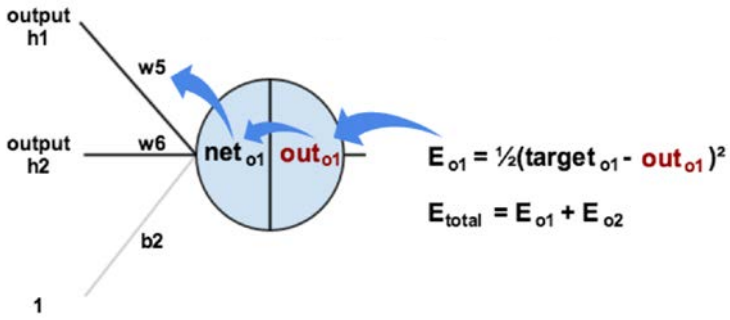
$$\frac{\partial E_{o_1}}{\partial w_5}$$

Backpropagation

- **An example**
  - Step04) Backward Pass

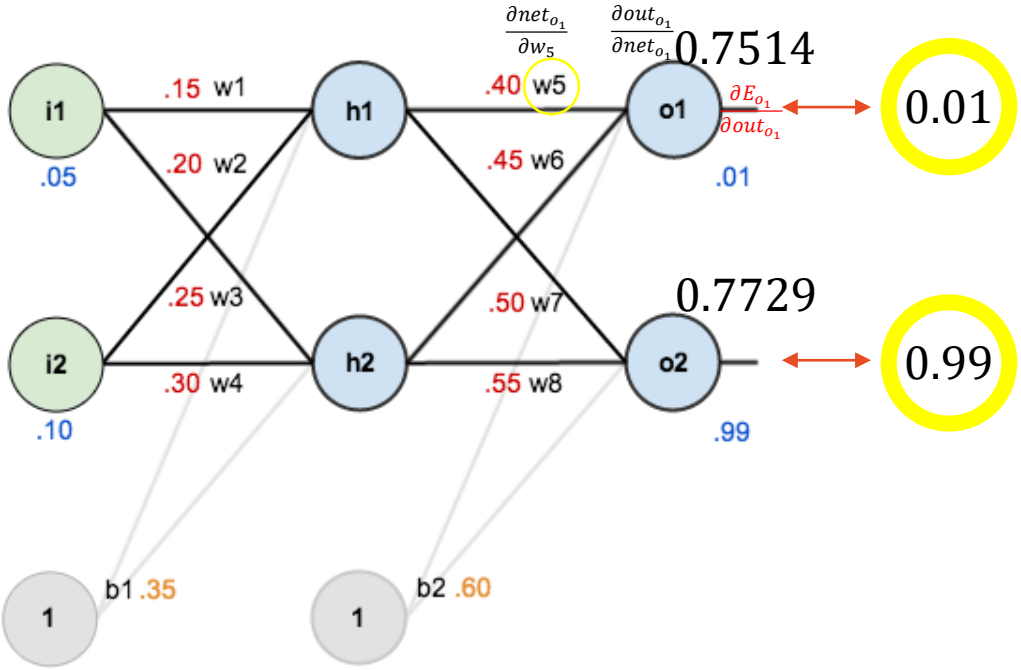


$$\frac{\partial E_{o_1}}{\partial w_5} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$



Backpropagation

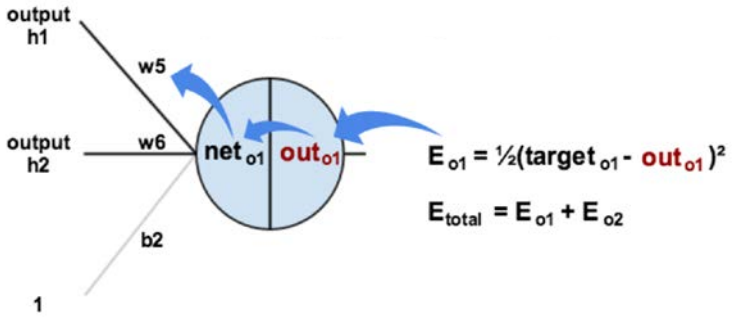
- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{o_1}}{\partial w_5} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

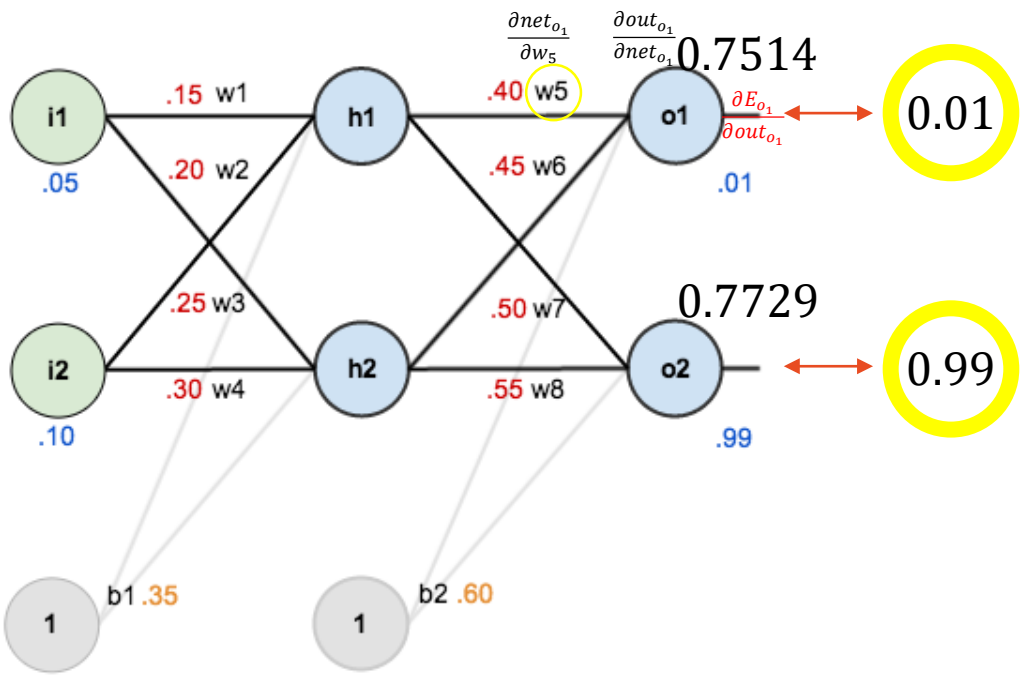
$$\frac{\partial E_{o_1}}{\partial out_{o_1}} = 2 * \frac{1}{2}(target_{o_1} - out_{o_1})$$

$$= -(0.01 - 0.751365) = 0.74137$$



Backpropagation

- An example
  - Step04) Backward Pass



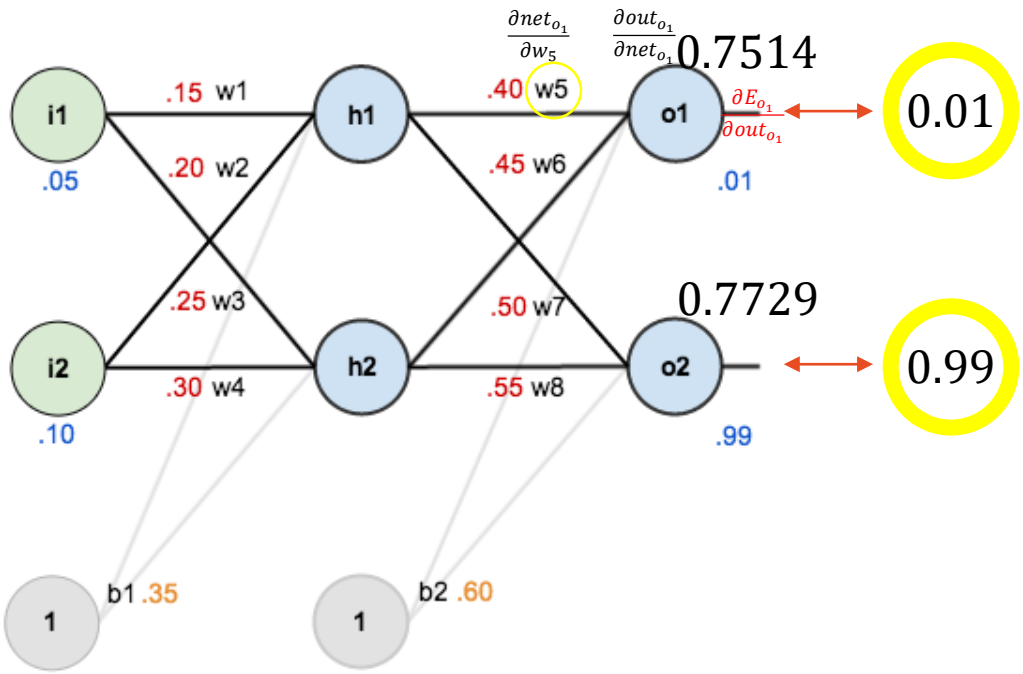
$$\frac{\partial E_{o_1}}{\partial w_5} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$= 0.74137$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}}$$

Backpropagation

- An example
  - Step04) Backward Pass



$$\frac{\partial E_{o_1}}{\partial w_5} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$= 0.74137$

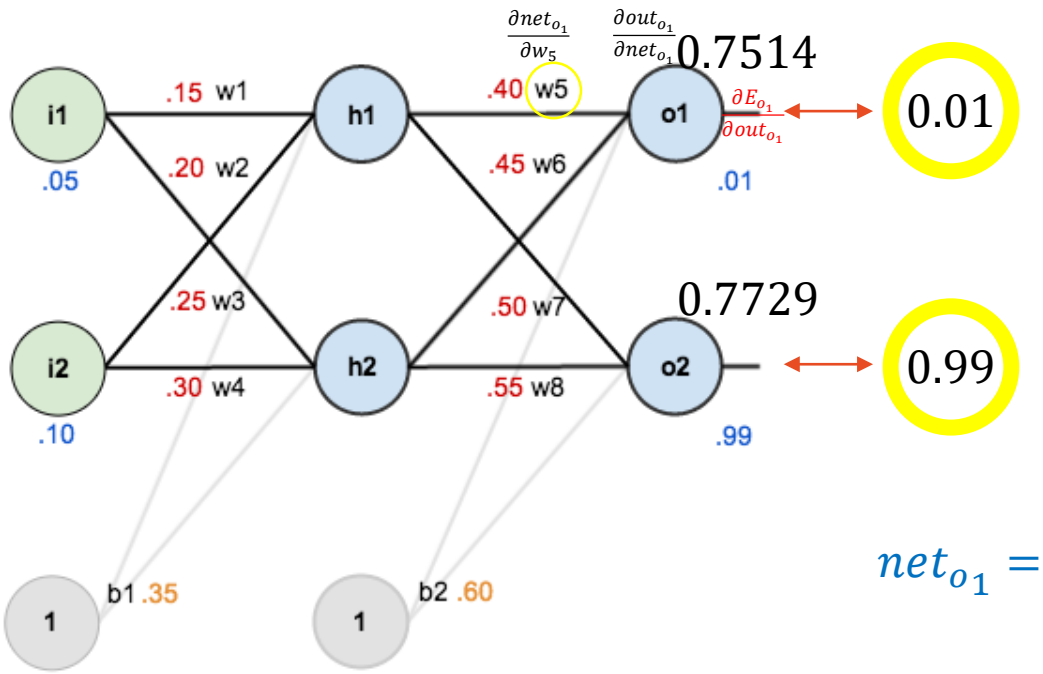
$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}}$$

$$\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1}(1 - out_{o_1})$$

$$= 0.7514(1-0.7614)=0.1868$$

Backpropagation

- An example
  - Step04) Backward Pass



$$\frac{\partial E_{o_1}}{\partial w_5} = \frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$= 0.74137$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}}$$

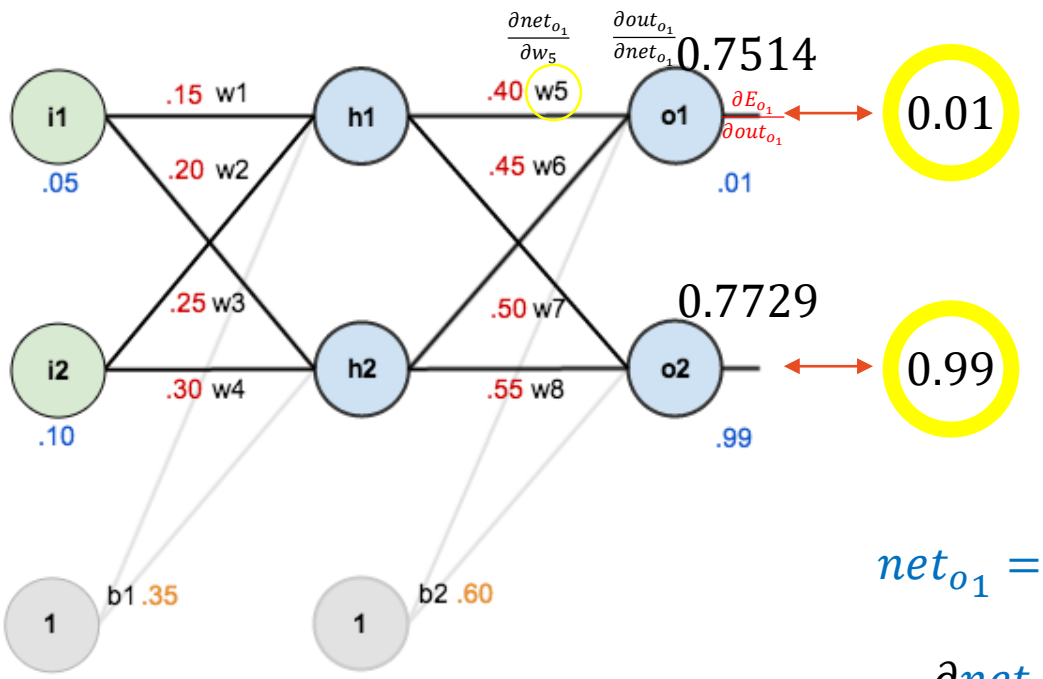
$$\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1}(1 - out_{o_1})$$

$$= 0.7514(1-0.7614)=0.1868$$

$$net_{o_1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{o_1}}{\partial w_5} = \underbrace{\frac{\partial E_{o_1}}{\partial out_{o_1}}}_{= 0.74137} * \frac{\partial out_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

$$out_{o_1} = \frac{1}{1 + e^{-net_{o_1}}}$$

$$\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1}(1 - out_{o_1})$$

$$= 0.7514(1-0.7614)=0.1868$$

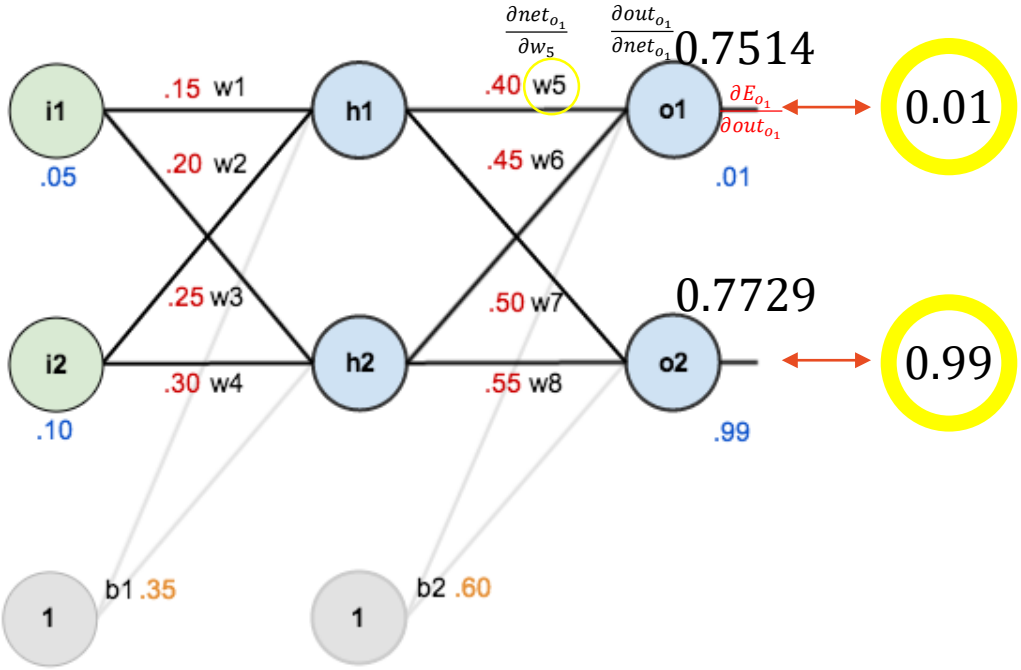
$$net_{o_1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o_1}}{\partial w_5} = 1 * out_{h1} + 0 + 0 = out_{h1}$$

$$= 0.5932$$

Backpropagation

- An example
  - Step04) Backward Pass



$$\frac{\partial E_{o_1}}{\partial w_5} = \underbrace{\frac{\partial E_{o_1}}{\partial out_{o_1}} * \frac{\partial out_{o_1}}{\partial net_{o_1}}}_{\delta_{o_1}} * \frac{\partial net_{o_1}}{\partial w_5}$$

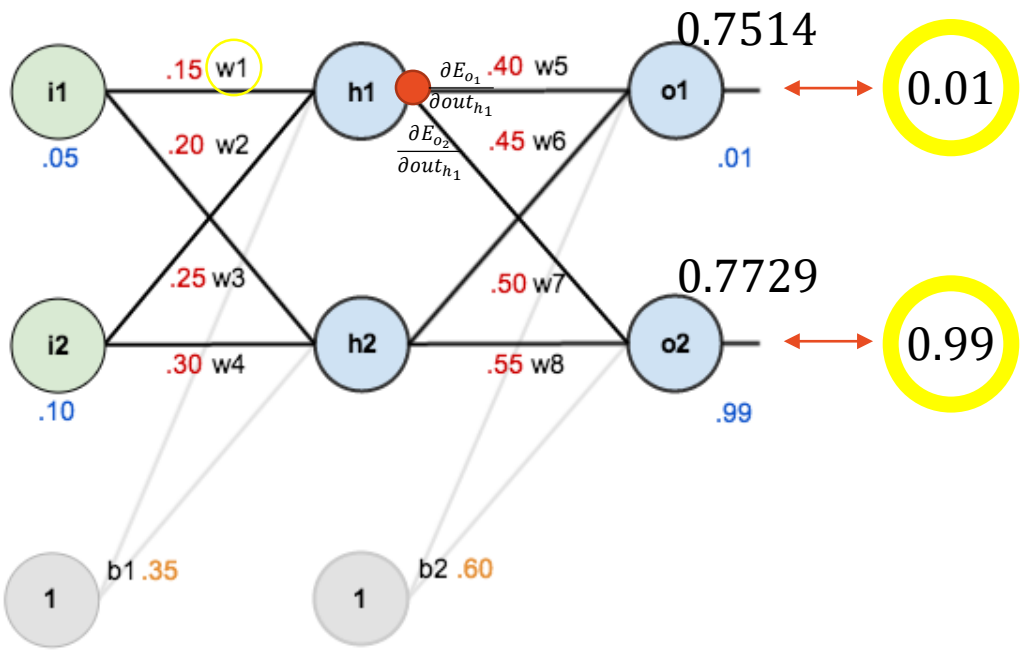
$$= 0.7414 * 0.1868 * 0.5933 = 0.0822$$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{o_1}}{\partial w_5}$$

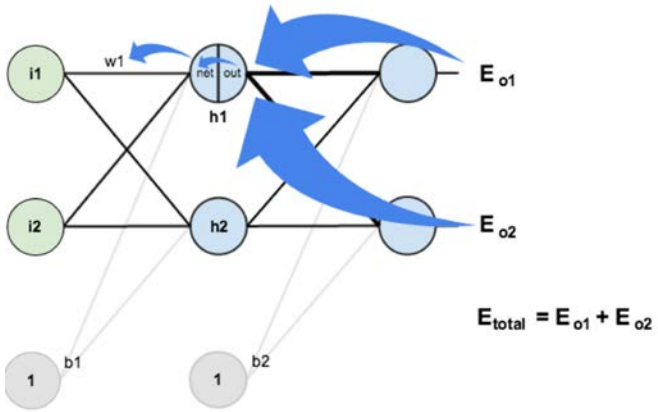
$$= 0.4 - 0.5 * 0.0822 = 0.3589$$

Backpropagation

- An example
  - Step04) Backward Pass

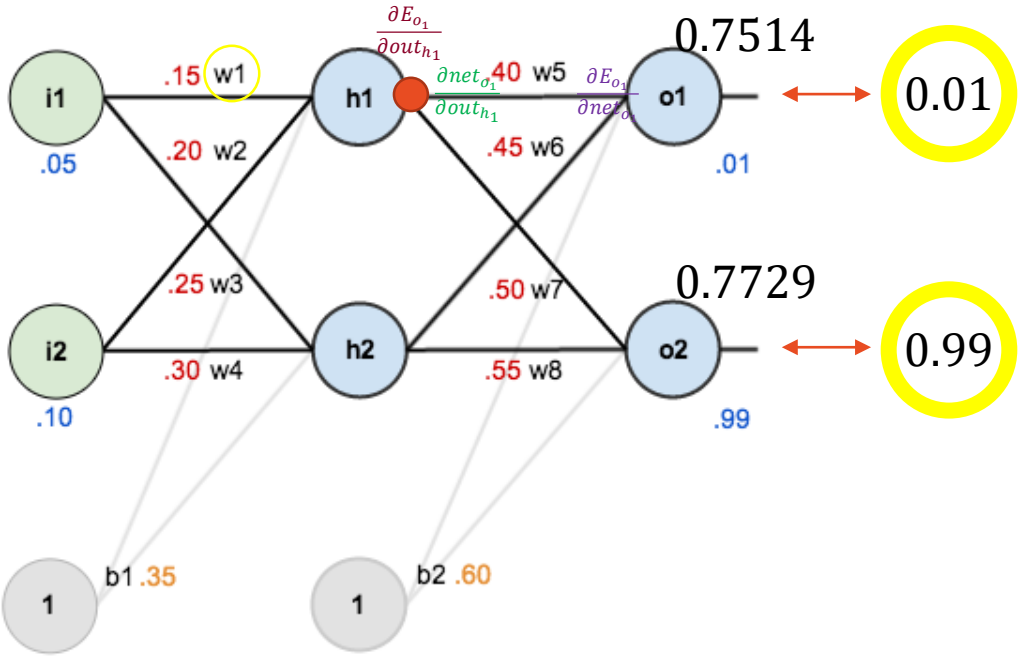


$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$



Backpropagation

- **An example**
  - Step04) Backward Pass

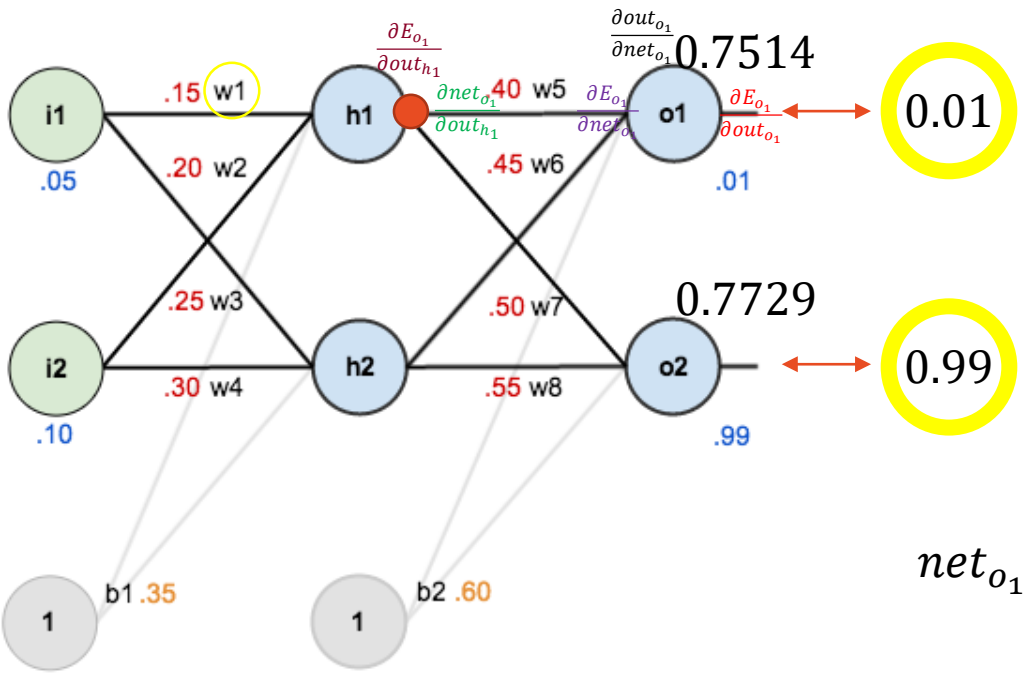


$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}}$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial net_{o_1}} * \frac{\partial net_{o_1}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial net_{o_1}} = 0.7414 * 0.1868$$

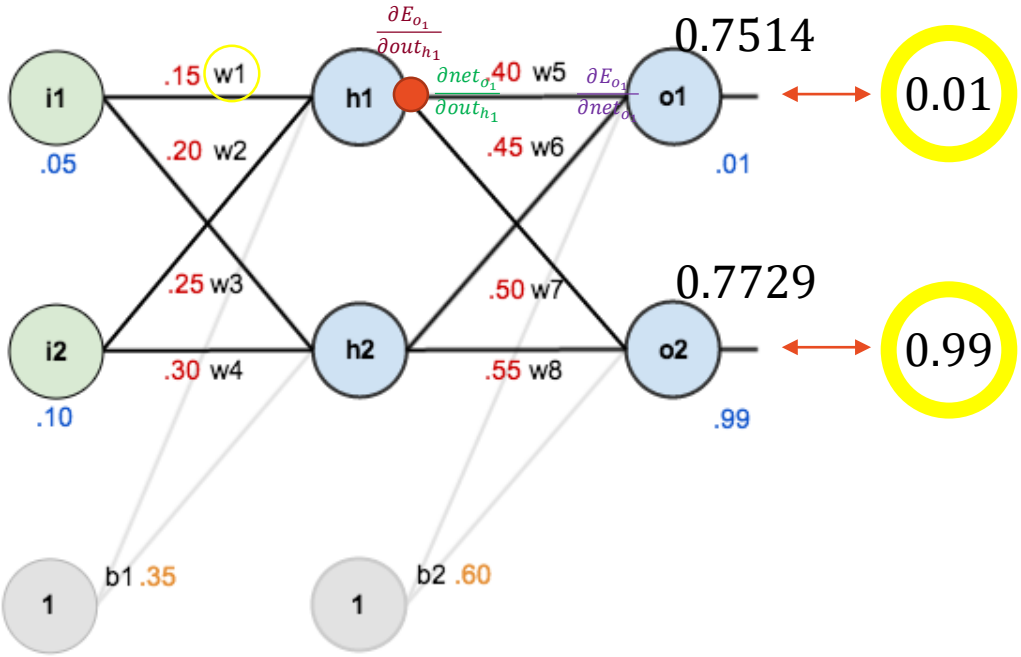
$$= 0.1385$$

$$net_{o_1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o_1}}{\partial out_{h_1}} = w_5 = 0.4$$

Backpropagation

- An example
  - Step04) Backward Pass

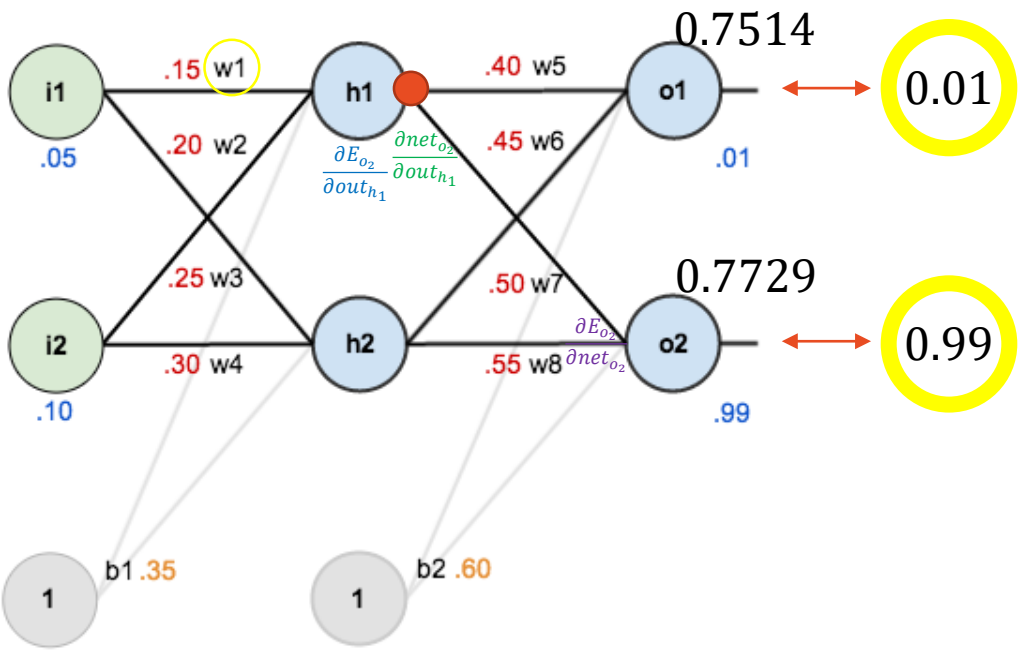


$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.1385 * 0.4 = 0.055399$$

Backpropagation

- **An example**
  - Step04) Backward Pass



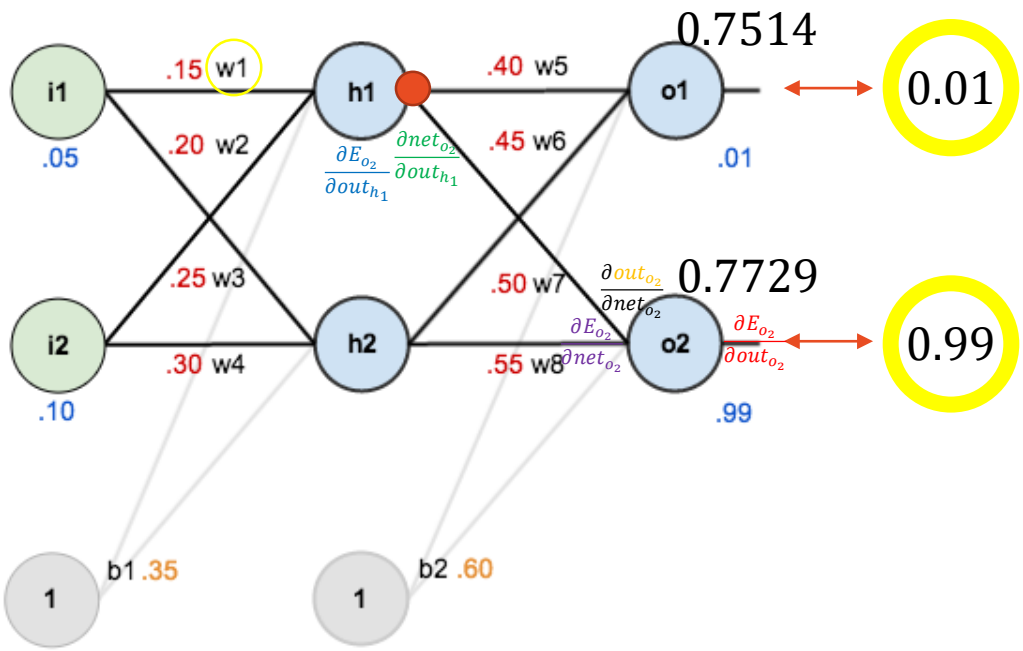
$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = \frac{\partial E_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial out_{h_1}}$$

Backpropagation

- An example
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

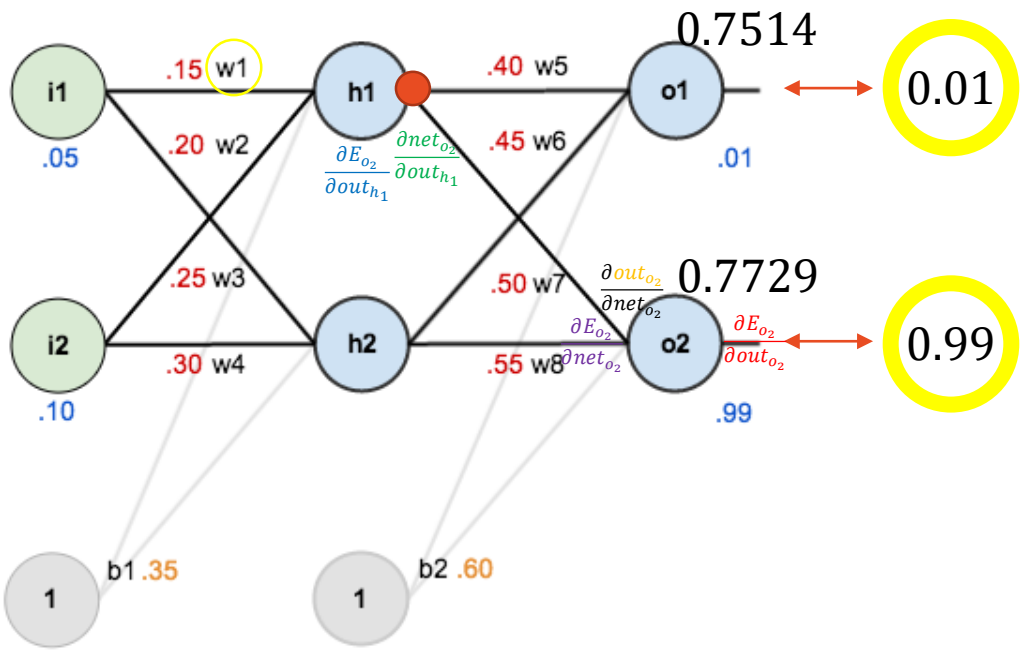
$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = \frac{\partial E_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_2}}{\partial net_{o_2}} = \frac{\partial E_{o_2}}{\partial out_{o_2}} * \frac{\partial out_{o_2}}{\partial net_{o_2}}$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = \frac{\partial E_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial out_{h_1}}$$

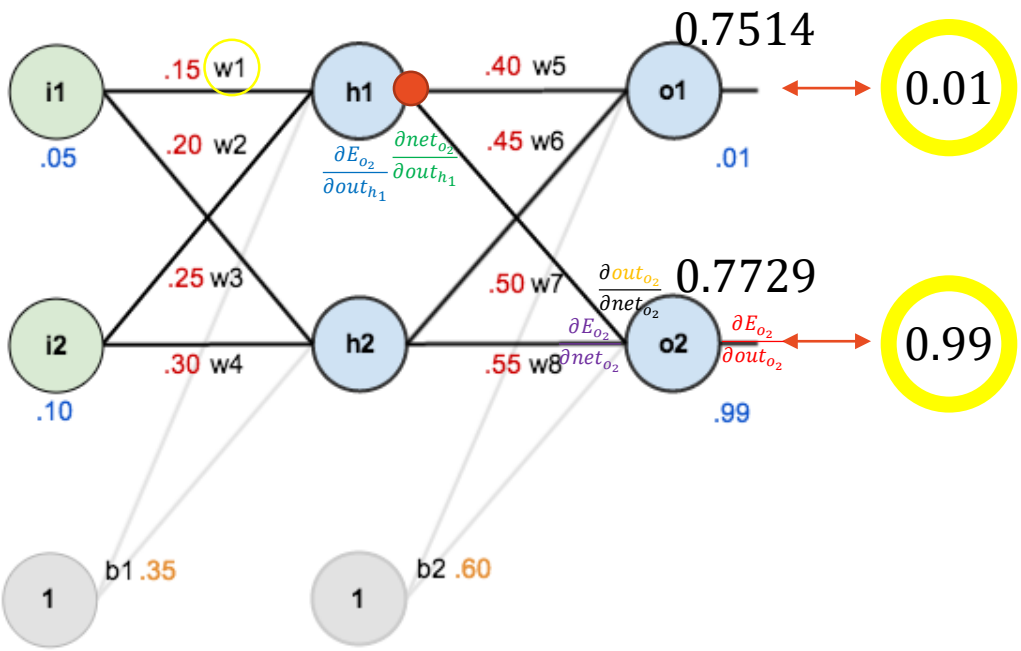
$$\frac{\partial E_{o_2}}{\partial net_{o_2}} = \frac{\partial E_{o_2}}{\partial out_{o_2}} * \frac{\partial out_{o_2}}{\partial net_{o_2}}$$

$$\frac{\partial E_{o_2}}{\partial out_{o_2}} = 2 * \frac{1}{2}\left(target_{o_2} - out_{o_2}\right)$$

$$= -(0.99 - 0.7729) = -0.2171$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = \frac{\partial E_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_2}}{\partial net_{o_2}} = -0.2171 * \frac{\partial out_{o_2}}{\partial net_{o_2}}$$

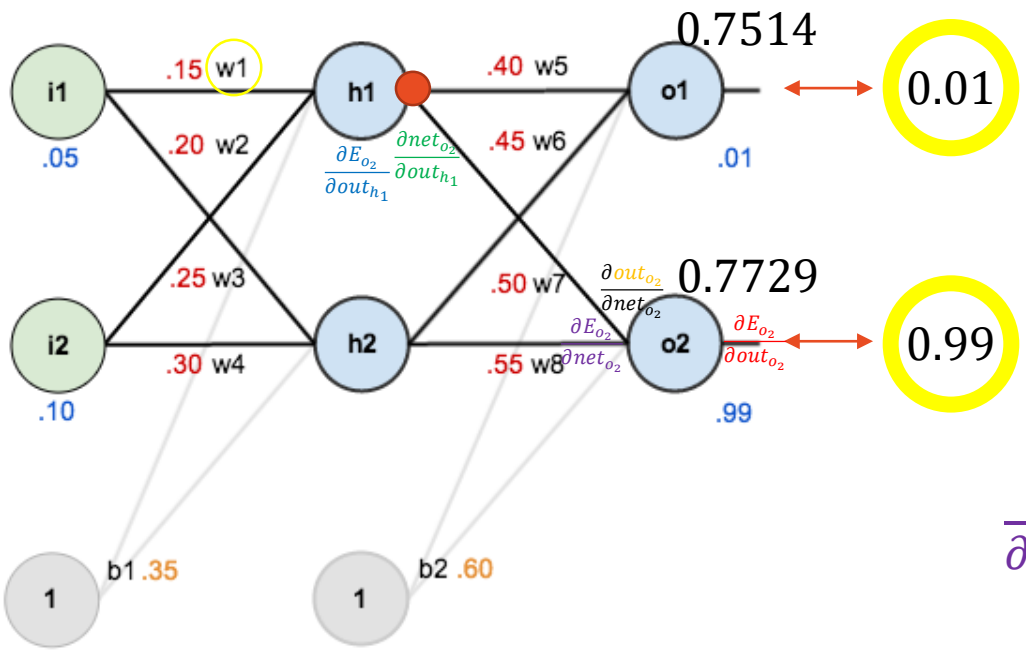
$$out_{o_2} = \frac{1}{1 + e^{-net_{o2}}}$$

$$\frac{\partial out_{o_2}}{\partial net_{o_2}} = out_{o_2}(1 - out_{o_2})$$

$$= 0.7729(1-0.7729) = 0.1755$$

Backpropagation

- An example
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

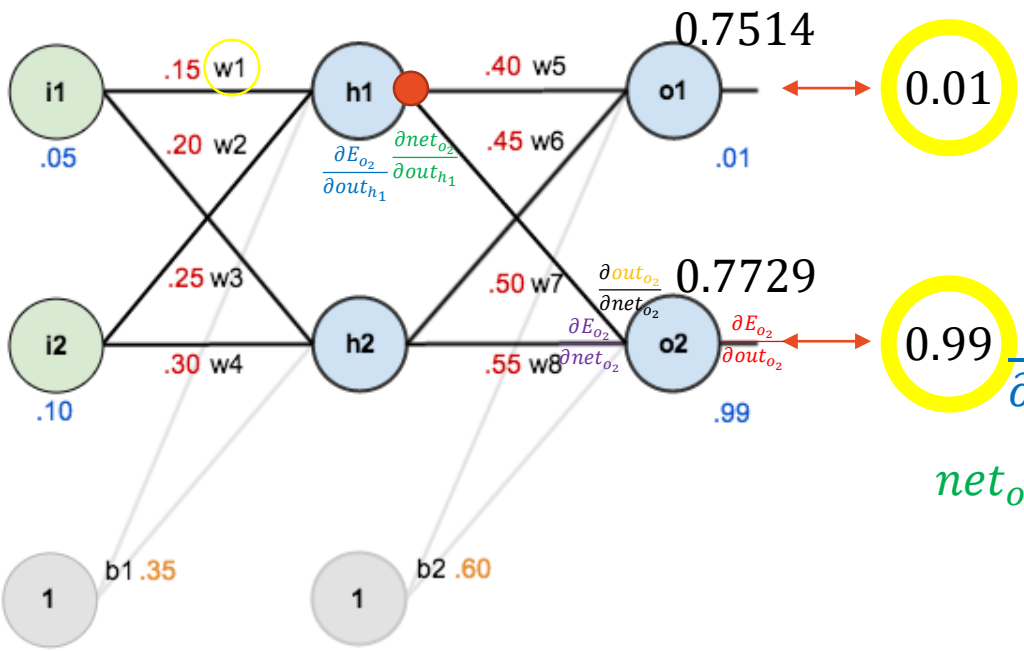
$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = \frac{\partial E_{o_2}}{\partial net_{o_2}} * \frac{\partial net_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_2}}{\partial net_{o_2}} = -0.2171 * 0.1755 = -0.0381$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

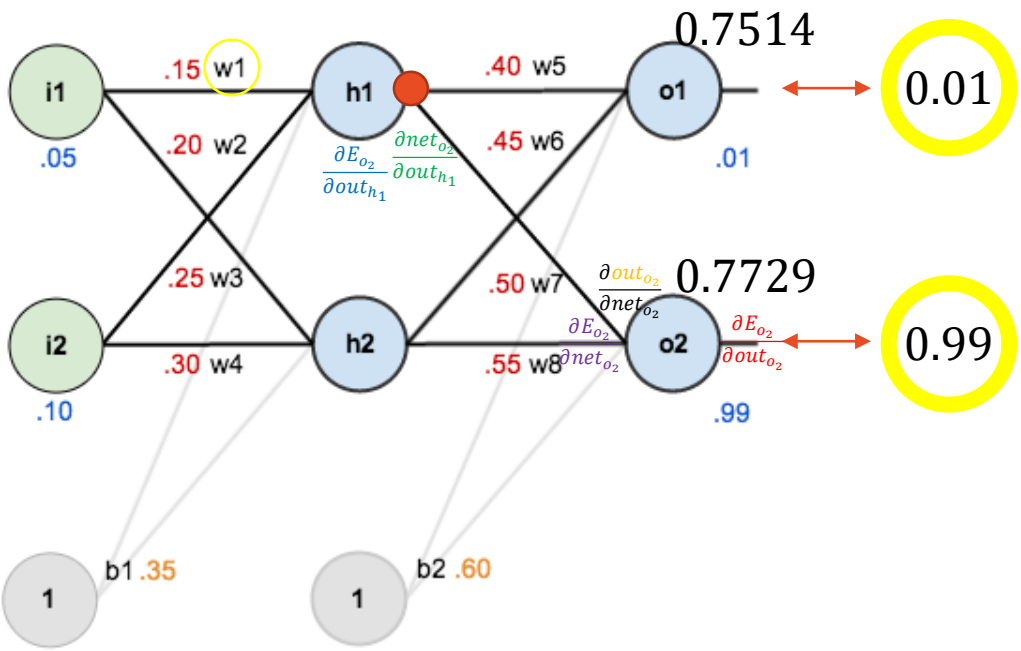
$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = -0.0381 * \frac{\partial net_{o_2}}{\partial out_{h_1}}$$

$$net_{o_2} = w_7 * out_{h1} + w_8 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o_2}}{\partial out_{h_1}} = w_7 = 0.5$$

Backpropagation

- **An example**
  - Step04) Backward Pass



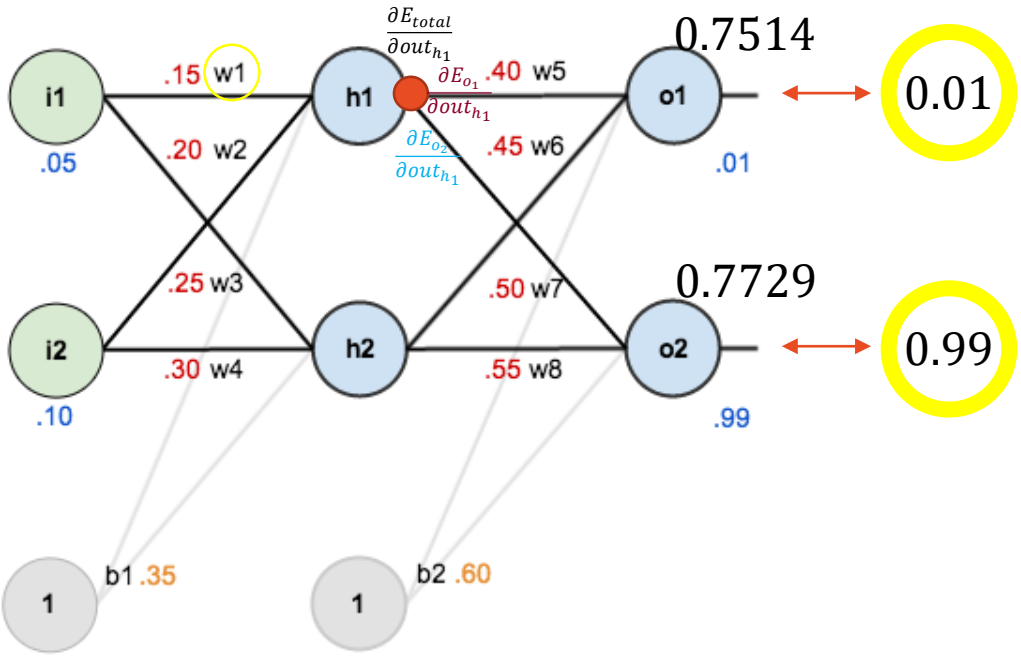
$$\frac{\partial E_{total}}{\partial out_{h_1}} = \frac{\partial E_{o_1}}{\partial out_{h_1}} + \frac{\partial E_{o_2}}{\partial out_{h_1}}$$

$$\frac{\partial E_{o_1}}{\partial out_{h_1}} = 0.055399$$

$$\frac{\partial E_{o_2}}{\partial out_{h_1}} = -0.0381 * 0.5 = -0.01905$$

Backpropagation

- An example
  - Step04) Backward Pass

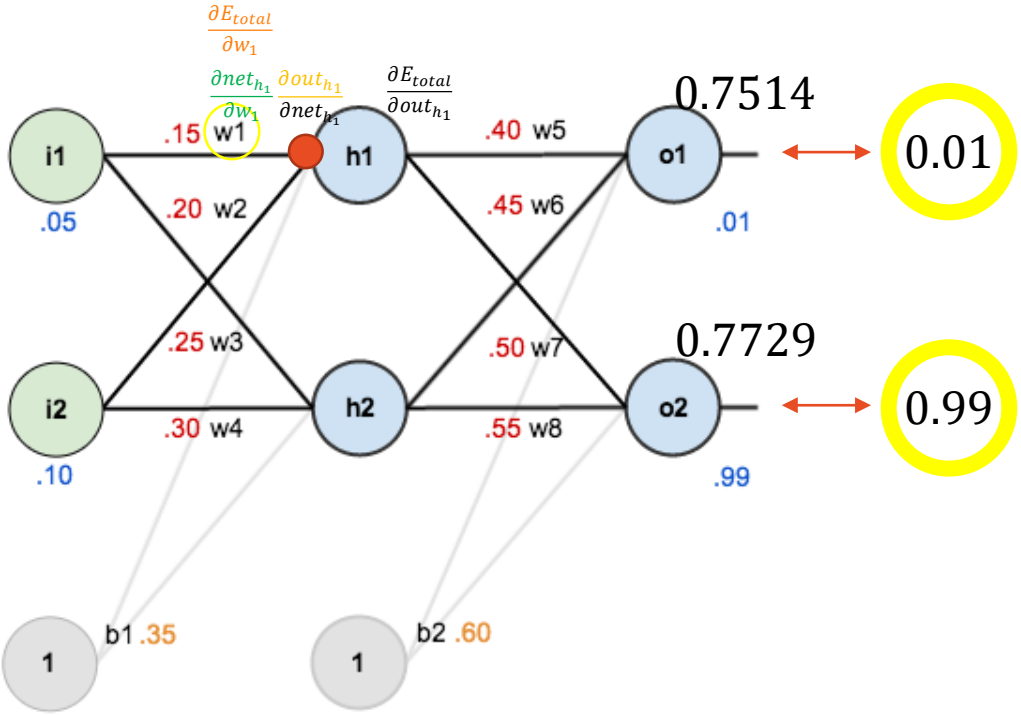


$$\frac{\partial E_{total}}{\partial out_{h_1}} = 0.055399 + -0.01905$$

$$= 0.03634$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = 0.055399 + -0.01905$$

$$= 0.03634$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h_1}} * \frac{\partial out_{h_1}}{\partial net_{h_1}} * \frac{\partial net_{h_1}}{\partial w_1}$$

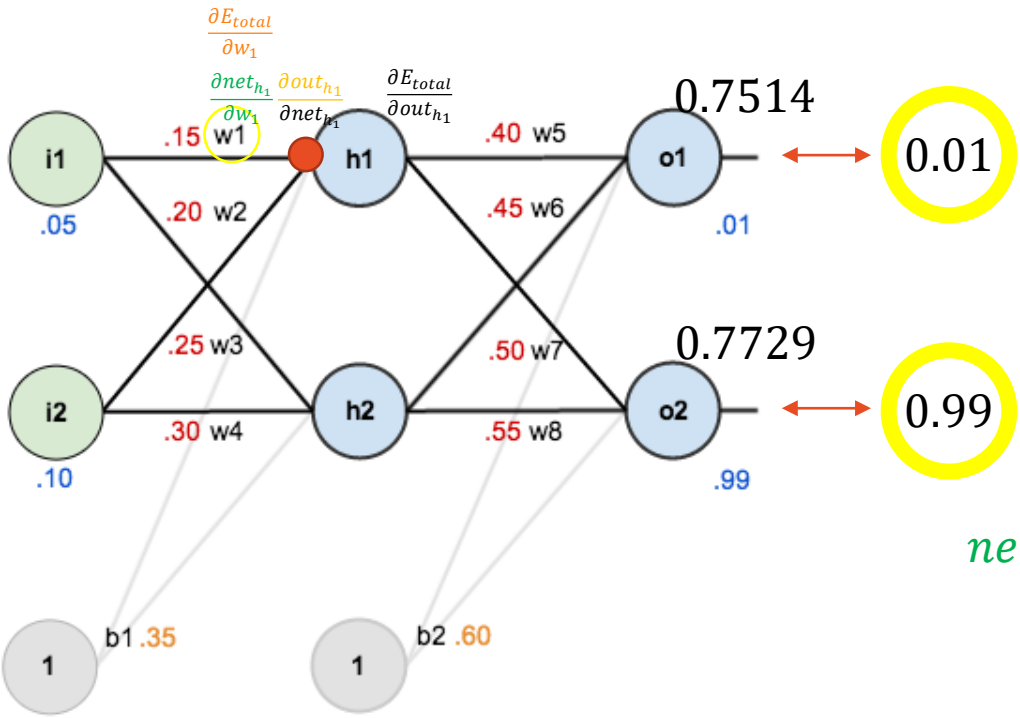
$$out_{h_1} = \frac{1}{1 + e^{-net_{h_1}}}$$

$$\frac{\partial out_{o_1}}{\partial net_{h_1}} = out_{h_1}(1 - out_{h_1})$$

$$= 0.5932(1-0.5932)=0.2413$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = 0.055399 + -0.01905$$

$$= 0.03634$$

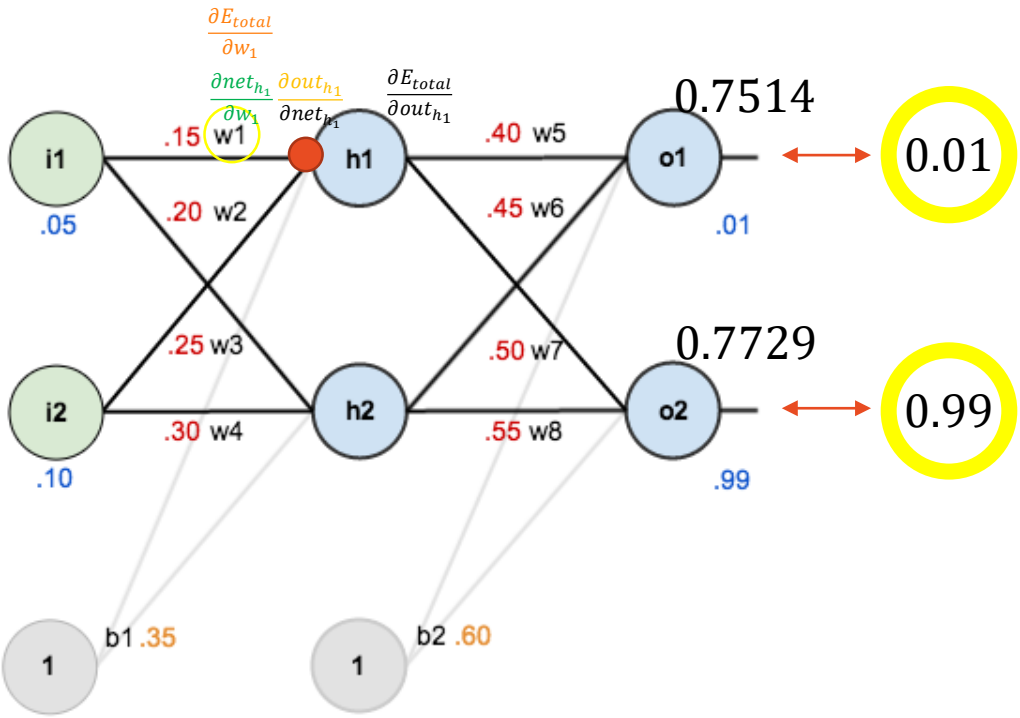
$$\frac{\partial E_{total}}{\partial w_1} = 0.03634 * 0.2413 * \frac{\partial net_{h_1}}{\partial w_1}$$

$$net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_2 * 1$$

$$\frac{\partial net_{h_1}}{\partial w_1} = i_1 = 0.05$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = 0.055399 + -0.01905$$

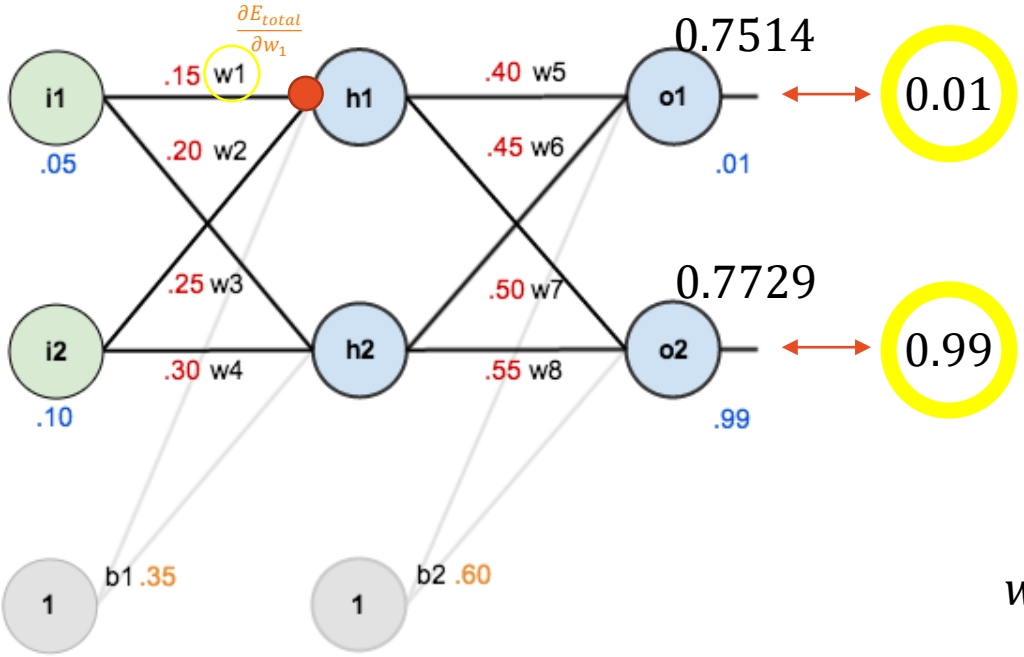
$$= 0.03634$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.03634 * 0.2413 * 0.05$$

$$= 0.00044$$

Backpropagation

- **An example**
  - Step04) Backward Pass



$$\frac{\partial E_{total}}{\partial out_{h_1}} = 0.055399 + -0.01905$$
$$= 0.03634$$

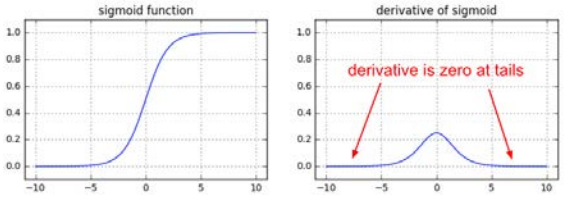
$$\frac{\partial E_{total}}{\partial w_1} = 0.03634 * 0.2413 * 0.05$$
$$= 0.00044$$

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1}$$
$$= 0.15 - 0.5 * 0.00044 = 0.1498$$

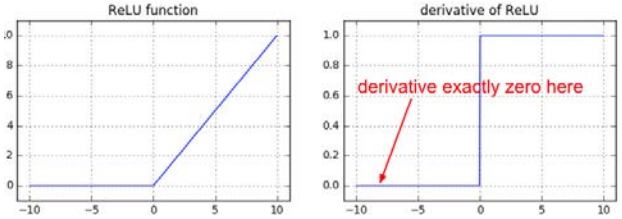
Backpropagation

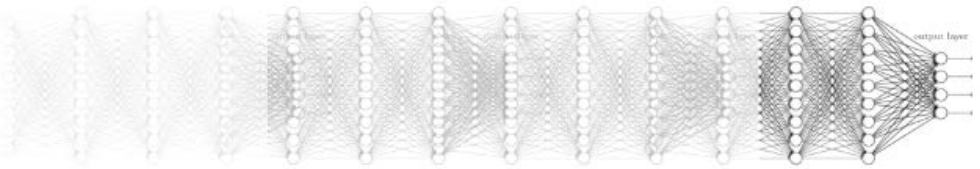
❖ **Vanishing gradient**

- Sigmoid: 미분값이 0이 되기 쉬움
- Relu: 0보다 큰 x에 대해서 x만큼의 미분값을 가짐



Vanishing gradient (NN winter2: 1986-2006)

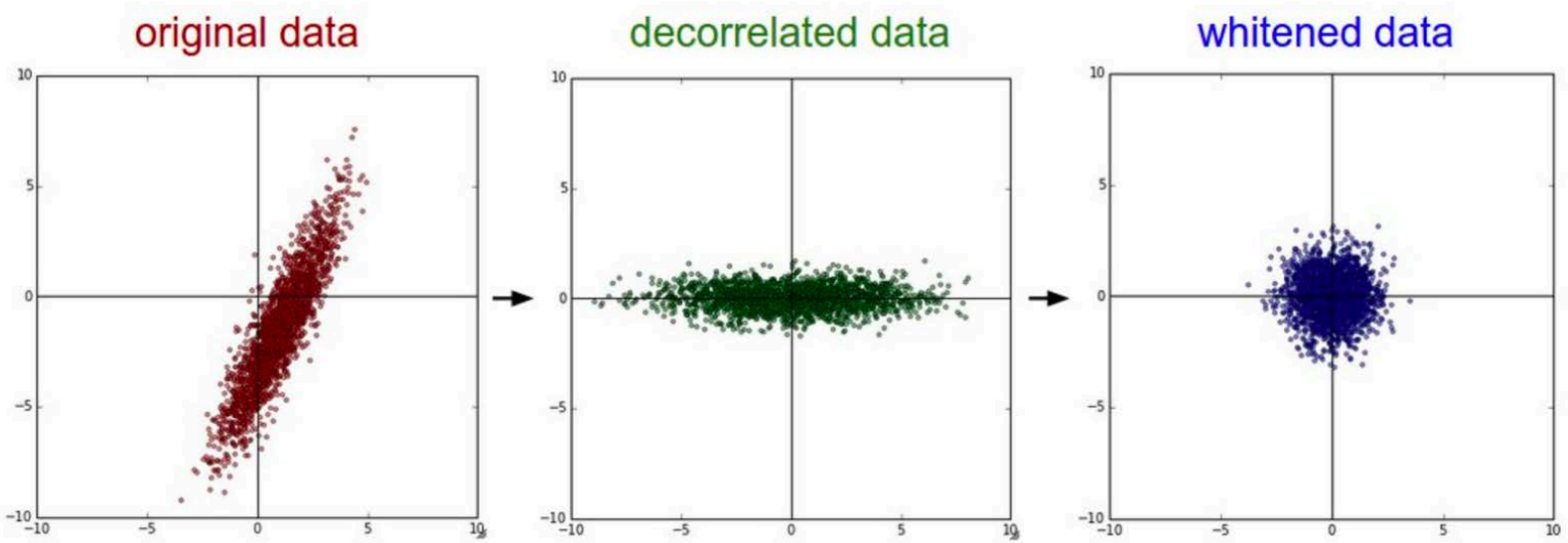


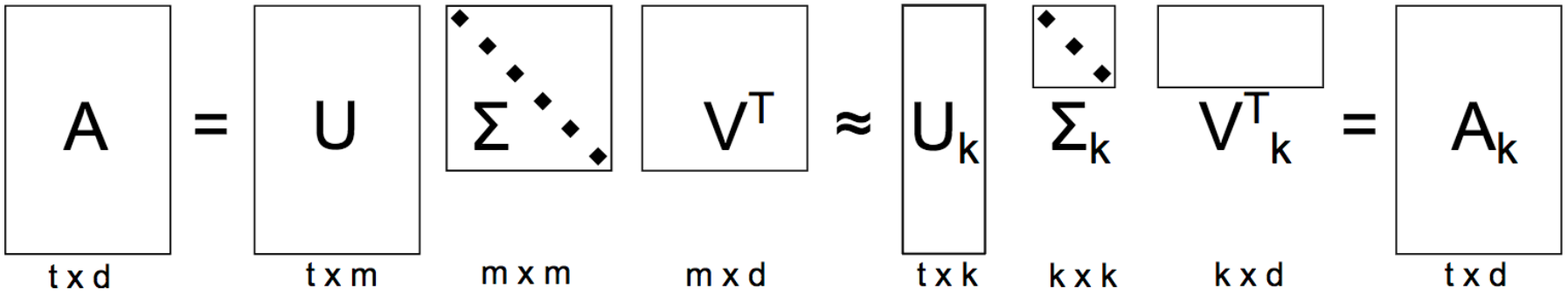


Contents

Data Preprocessing

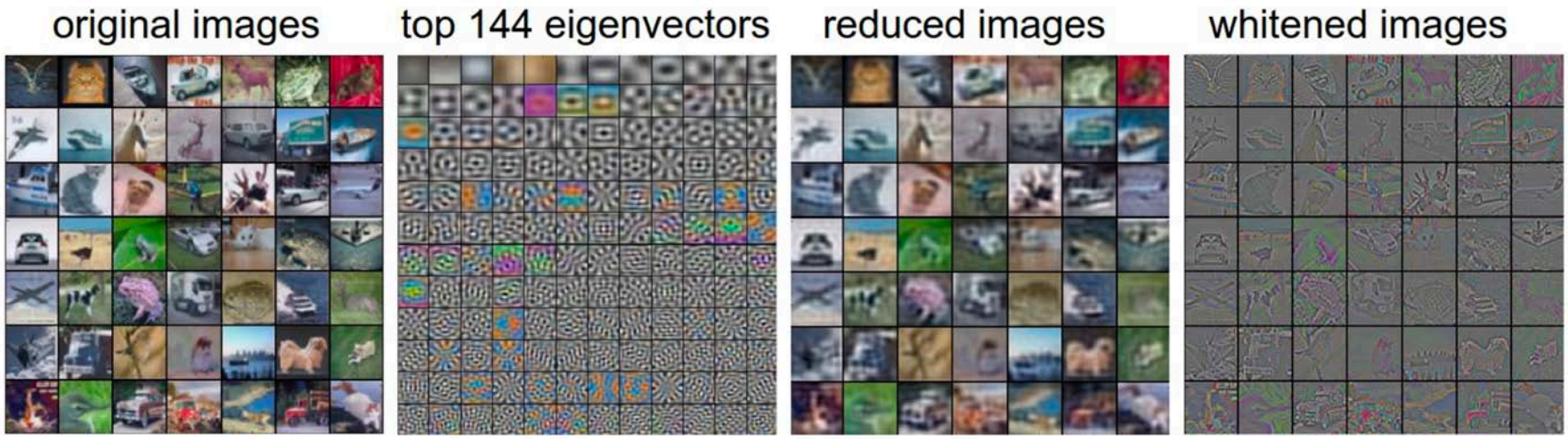
- Decorrelated data: $U^T X$
- Whitened data: $U^T X / \sqrt{S + \epsilon}$

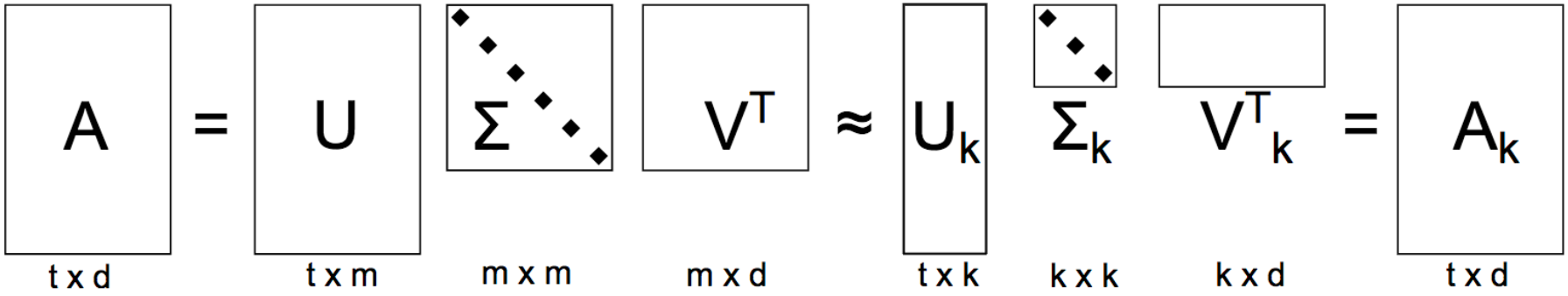




Data Preprocessing

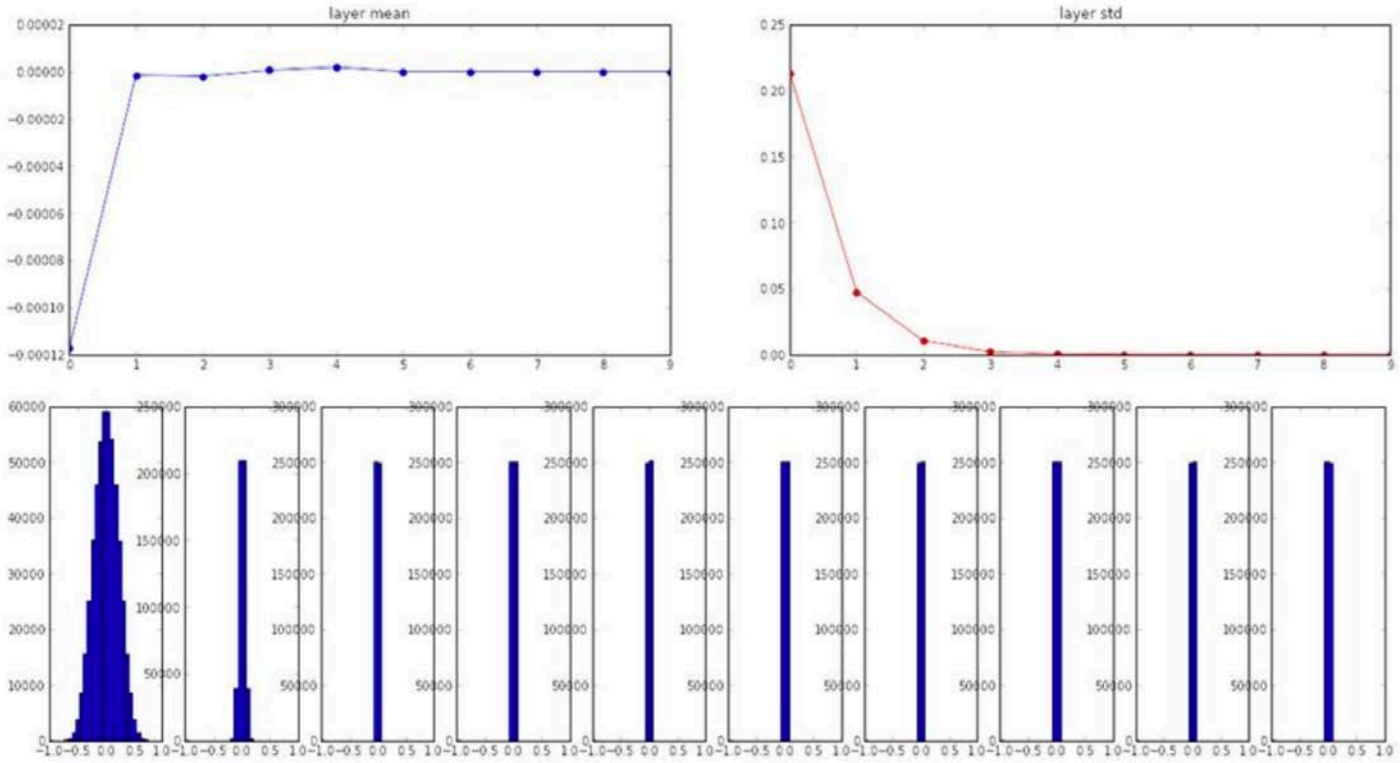
- Decorrelated data: $U^T X$
- Whitened data: $U^T X / \sqrt{S + \epsilon}$





Weight Initialization

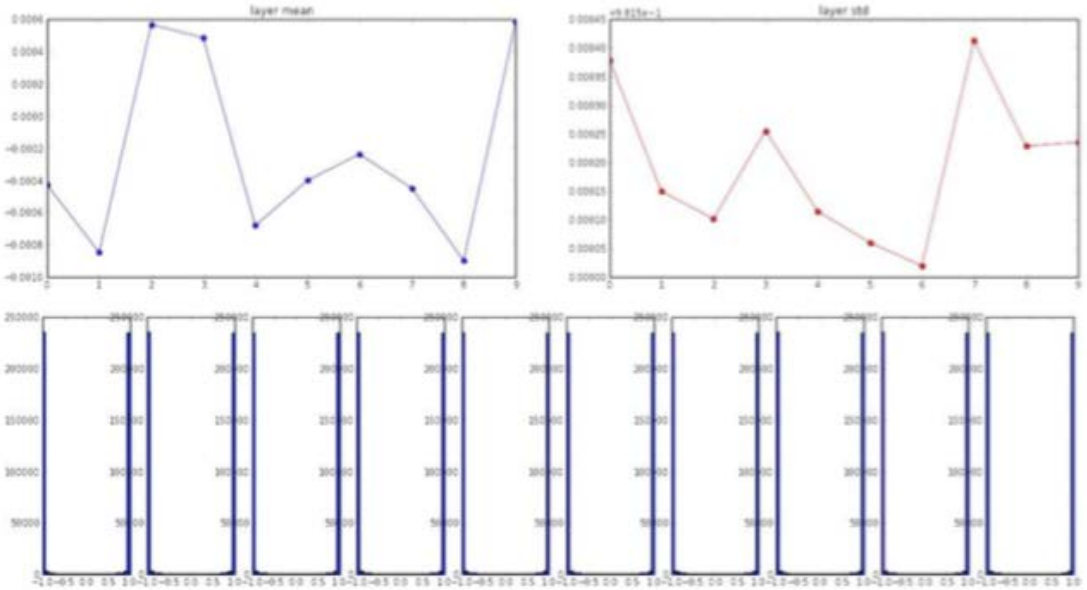
- **All zero initialization**
  - If you initialize your weights to zero, your gradient descent will never converge
- **Initialize with small values**
  - W = 0.01* np.random.randn(D,H)
  - Very small weights will during backpropagation compute very small gradients on its data



Weight Initialization

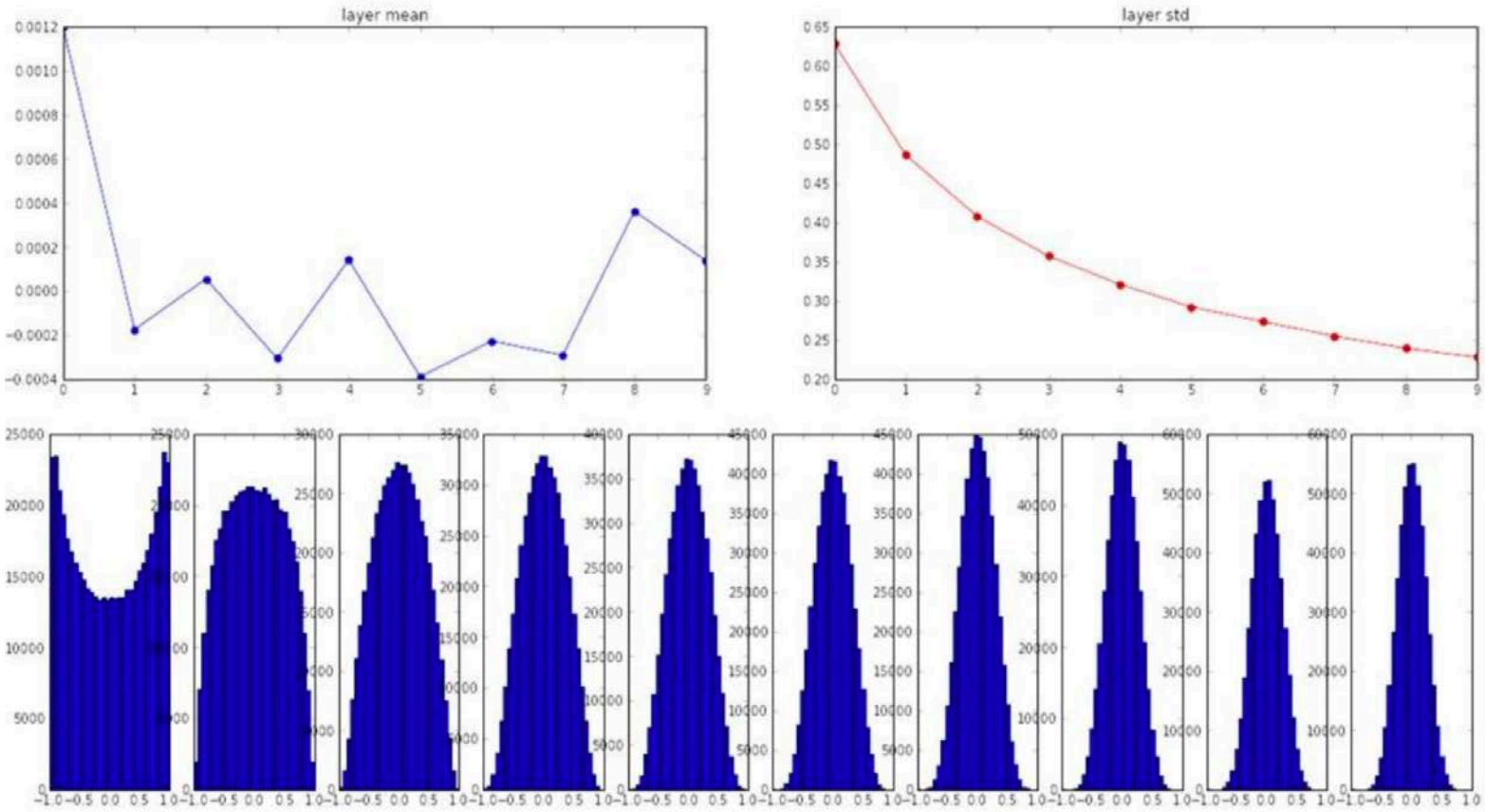
❖ **표준편차를 0.01에서 1.0로 바꿀 경우**

- pre-activation값이 너무 큼



Weight Initialization

- **Calibrating the variances with 1/sqrt(n)**
  - The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs
  - w = np.random.randn(n) / sqrt(n)



Weight Initialization

- **Calibrating the variances with 1/sqrt(n)**
  - The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs
  - w = np.random.randn(n) / sqrt(n)

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_{i}^{n} w_i x_i)$$

Weight Initialization

- **Calibrating the variances with 1/sqrt(n)**
  - The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs
  - w = np.random.randn(n) / sqrt(n)

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$\begin{aligned}\mathrm{Var}(X+Y) &= E[(X+Y)^2] - (E[X+Y])^2 \\ &= E[X^2 + 2XY + Y^2] - (E[X] + E[Y])^2\end{aligned}$$

$$\begin{aligned}\mathrm{Var}(X+Y) &= E[X^2] + 2E[XY] + E[Y^2] - (E[X]^2 + 2E[X]E[Y] + E[Y]^2) \\ &= E[X^2] + E[Y^2] - E[X]^2 - E[Y]^2 \\ &= \mathrm{Var}(X) + \mathrm{Var}(Y).\end{aligned}$$

Weight Initialization

- **Xavier initialization**
  - The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of input nodes
  - w = np.random.randn(n) / sqrt(n)

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$\begin{aligned}\mathrm{Var}(X) &= \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] \\ &= \mathrm{E}\left[X^2 - 2X\,\mathrm{E}[X] + \mathrm{E}[X]^2\right] \\ &= \mathrm{E}\left[X^2\right] - 2\,\mathrm{E}[X]\,\mathrm{E}[X] + \mathrm{E}[X]^2 \\ &= \mathrm{E}\left[X^2\right] - \mathrm{E}[X]^2\end{aligned}$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= (n\mathrm{Var}(w))\ \mathrm{Var}(x)$$

Weight Initialization

❖ **Glorot & Bengio's formula**

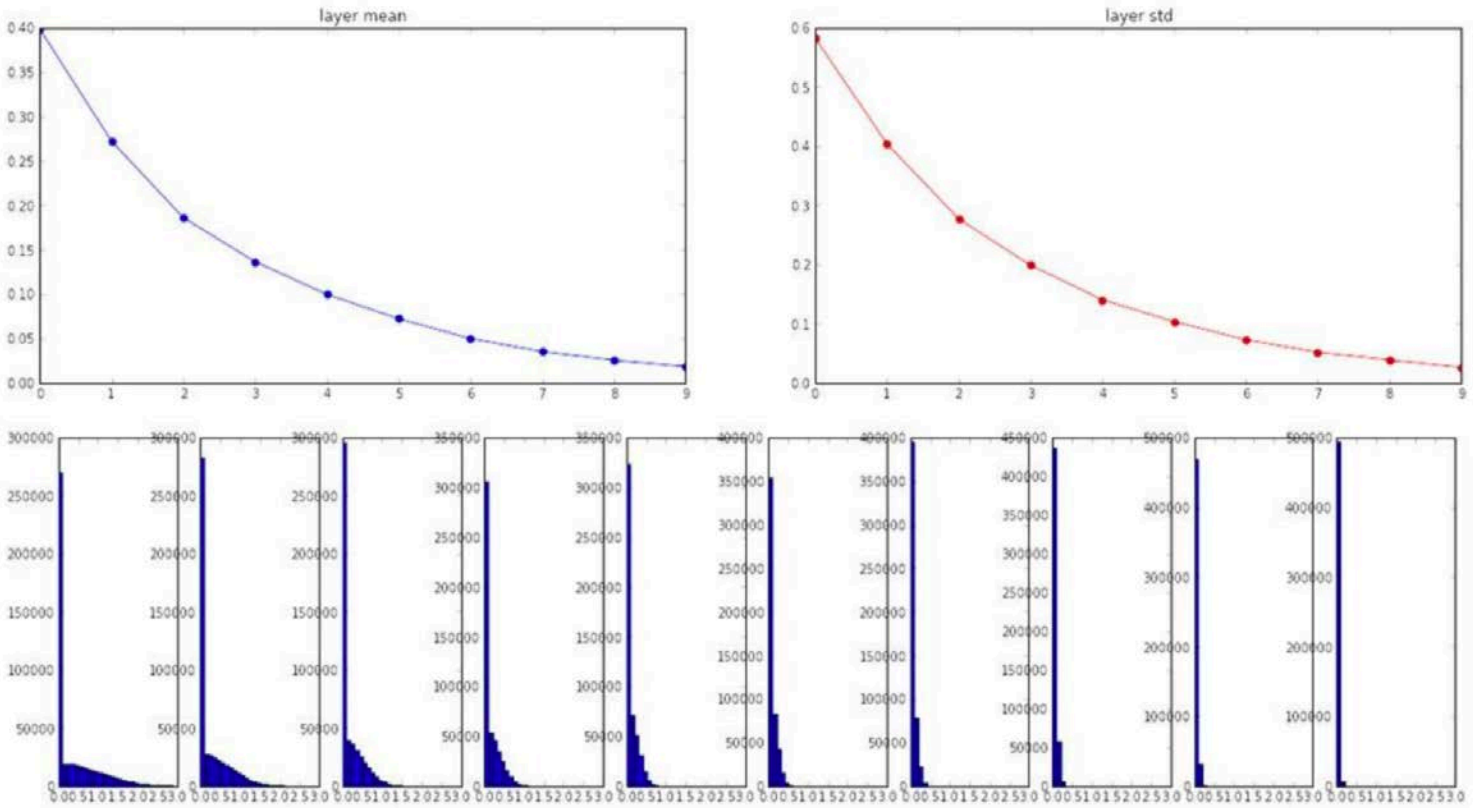
- To keep the variance of the input gradient & the output gradient the same
- Input 과 output의 평균의 수를 고려

$$\mathrm{Var}(W_i) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

Weight Initialization

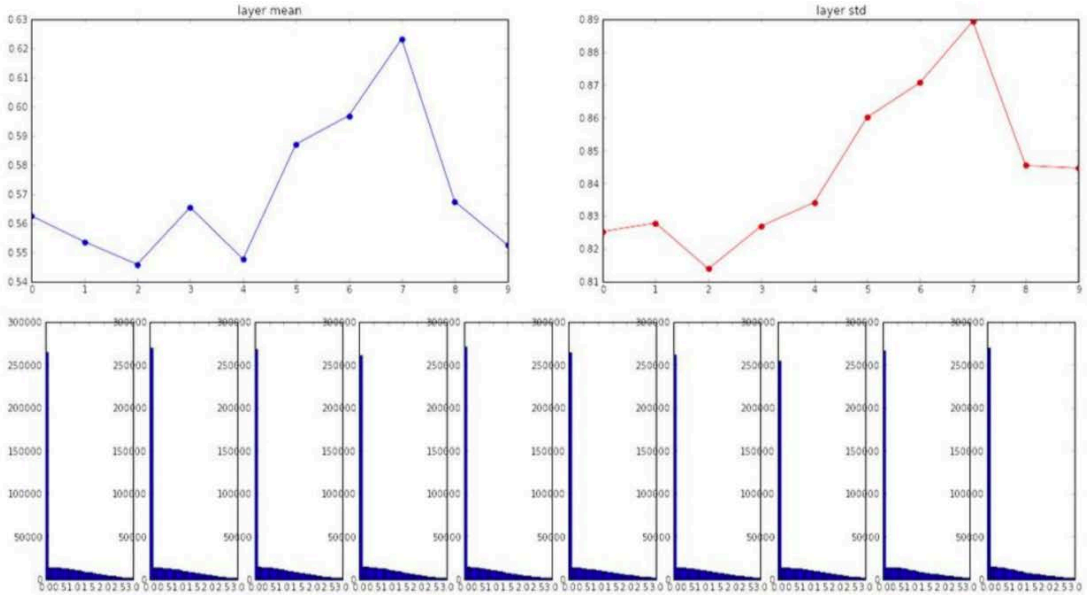
❖ **He initialization**

- a rectifying linear unit is zero for half of its input
- Xavier initialization은 activation(ReLU)을 고려하지 않았음

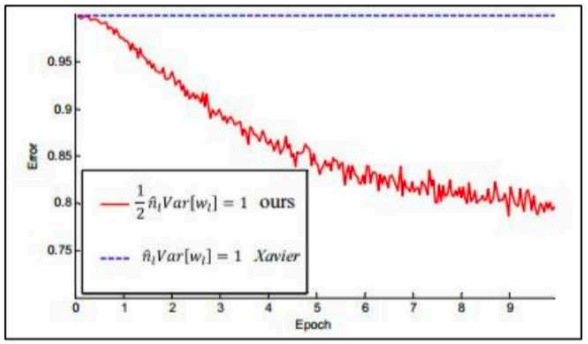


Weight Initialization

- **He initialization**
  - a rectifying linear unit is zero for half of its input



$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$



Learning rate

❖ **Overfit 되는지 확인해보기**

- 20개 정도의 적은 데이터를 먼저 학습해 봄
- Train accuracy가 1.0인지 확인
- Train accuracy는 증가, Loss는 줄어들지 않을 경우
  - 어떤 데이터는 잘 맞지만, 다른 데이터는 loss가 증가
- Learning rate [0.001, 0.000001]
- learning rate는 초기에는 높은 learning rate로 학습 시키다 점점 learning rate를 감소시키는 것이 가장 효과적

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Hyperparameter optimization

❖ Hyperparameter Search

- log-space 상에서 정해질 수 있도록 조절
- epoch를 몇 번만 돌려 (5번 정도) 대략적인 방향
- Random search가 뉴럴넷에 적절함

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6)

    trainer = ClassifierTrainer()
    model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
    trainer = ClassifierTrainer()
    best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                            model, two_layer_net,
                                            num_epochs=5, reg=reg,
                                            update='momentum', learning_rate_decay=0.9,
                                            sample_batches = True, batch_size = 100,
                                            learning_rate=lr, verbose=False)
```

note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

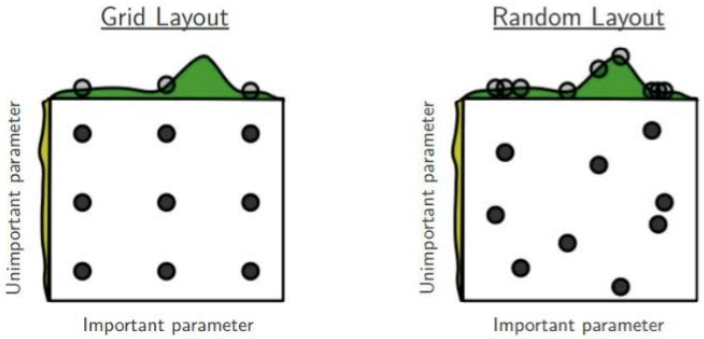
```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-5, 5)
    lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
    reg = 10**uniform(-4, 0)
    lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.



Contents

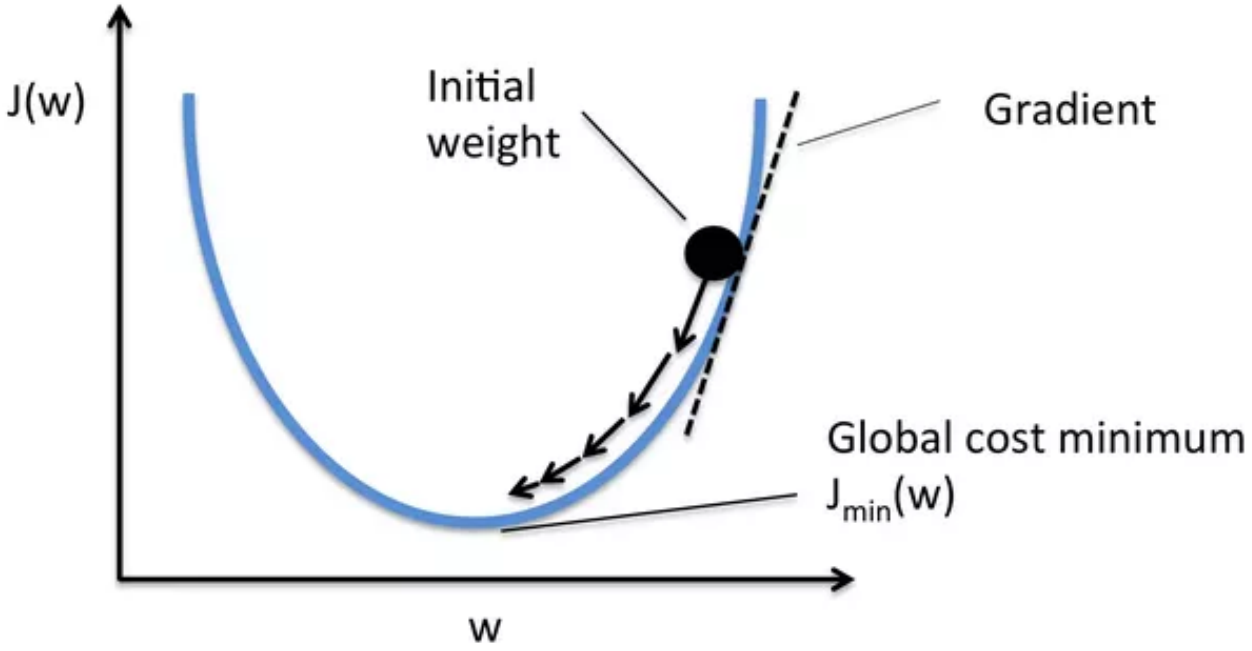
Faster Optimizers

- **Gradient Descent**
- **Momentum Method**
  - Momentum
  - Nesterov Accelerated Gradient Algorithm(NAG)
- **Adaptive Method**
  - Adagrad
  - RMSprop
  - Adam
  - Adamdelta

Gradient Descent

Gradient Descent

- **Gradient Descending: 목적함수를 최소화하는 문제**
  - 미분값(Gradient)가 양수일때 왼쪽으로 향함 (목적함수가 최소화 되는 방향)
  - 미분값이(Gradient)가 음수일때 오른쪽으로 향함 (목적함수가 최소화 되는 방향)
- **Gradient Ascending: 목적함수를 최대화하는 문제**



$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j}$$

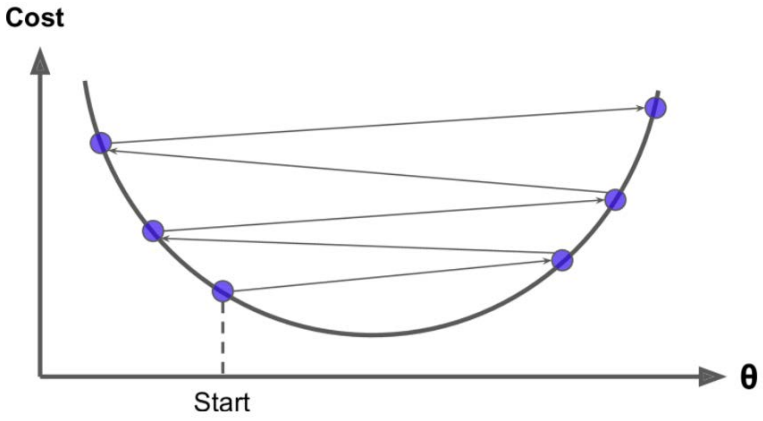
$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$$

Gradient Descent

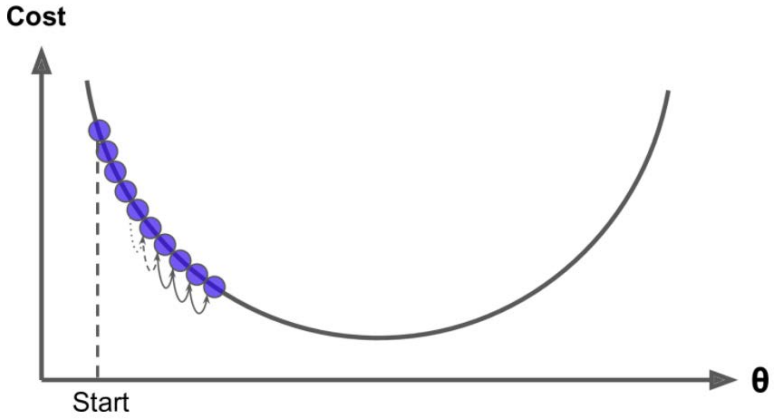
❖ **Learning rate: 학습된 그래디언트를 얼마나 반영할 것인가?**

- Too high learning rate: 수렴하지 못할 수도 있음
- Too low learning rate: 최적해에 도달하는데 학습속도가 느림

Too high learning rate

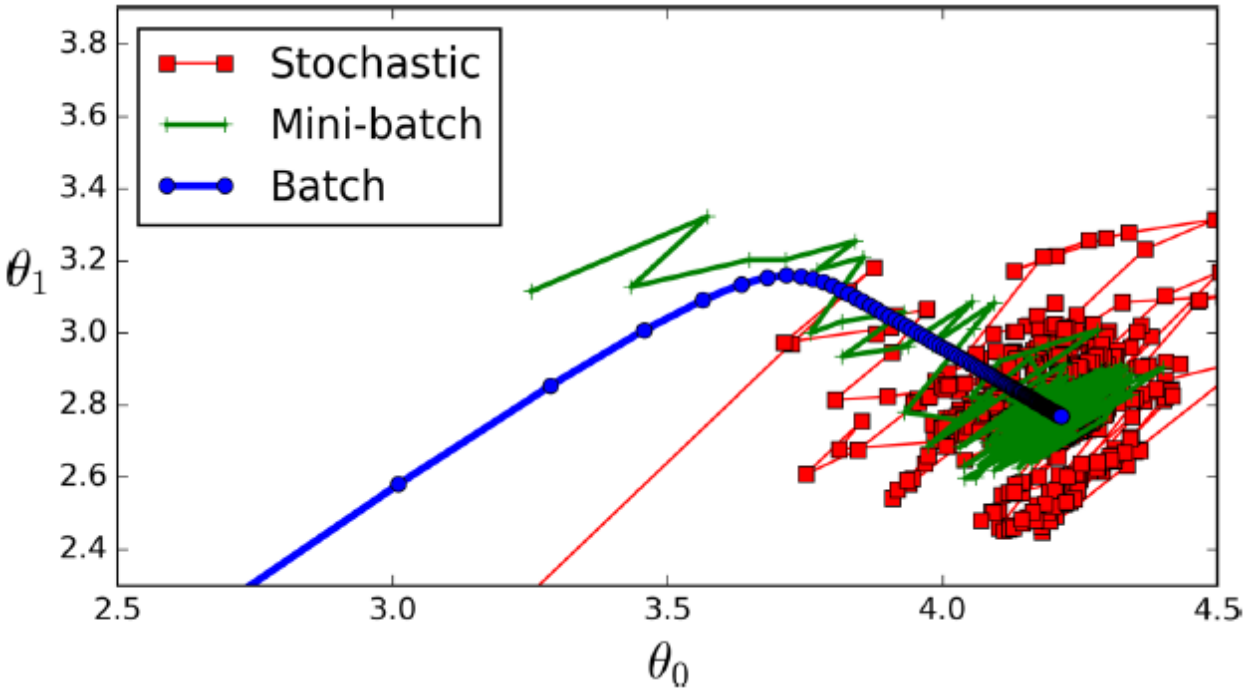


Too low learning rate



Gradient Descent

- GD : (Batch) Gradient Descent
  - 모든 학습데이터 사용
- SGD : Stochastic Gradient Descent
  - 오직 하나의 학습데이터(an example)를 학습
  - 실제 그래디언트의 확률적 추정(approximation)
- Mini-Batch GD : Mini-Batch Gradient Descent

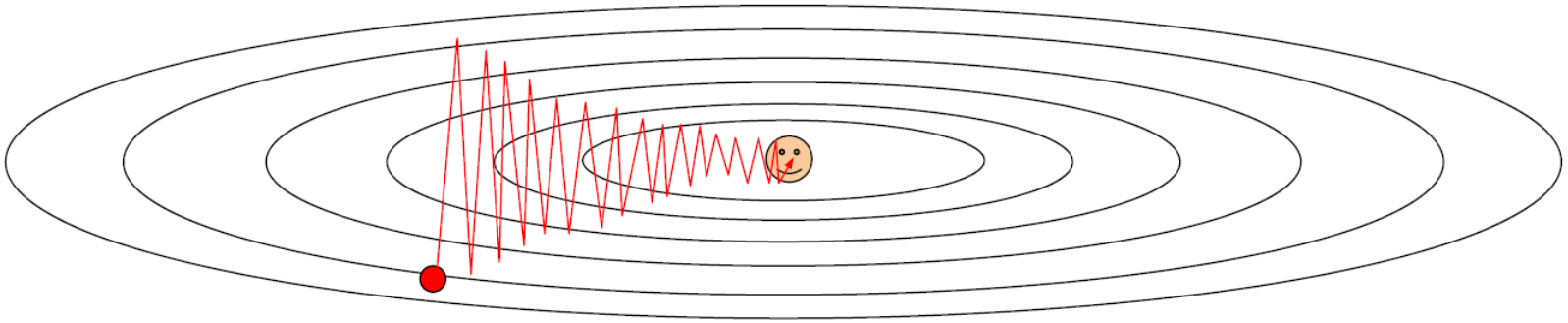


Momentum Method

Momentum Method

- **Gradient Desent**
  - Slow: 매우 작은 스텝으로 규칙적으로 움직이기 때문에 너무 느림
  - Local gradient: 이전 그래디언트 값을 고려하지 않음
- **Shallow horizontally loss function?**
  - Very slow progress

Suppose loss function is steep vertically but shallow horizontally:



GD: 가파른 쪽으로 수직적으로 움직이는 경향

Momentum

- **얕은 곡면의 목적함수에서도 빠르게 찾을 수 있을까?**
  - Momentum을 사용
- β: 이전 그래디언트를 얼마나 고려할 것인가?

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

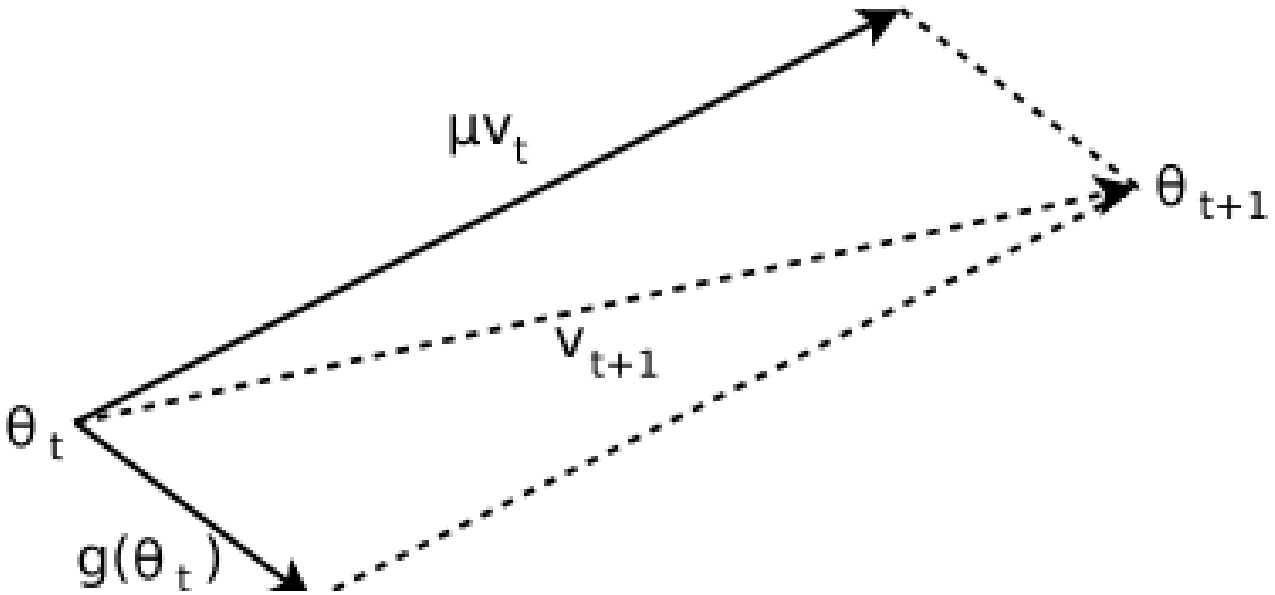
η: Learning rate
β: a weight of previous gradients(0.9)
m: like velocity
Terminal velocity: $\frac{1}{1-\beta}$

Momentum

❖ Momentum

- 가속도(이전 그래디언트)와 현재 그래디언트의 벡터 합으로 표현됨

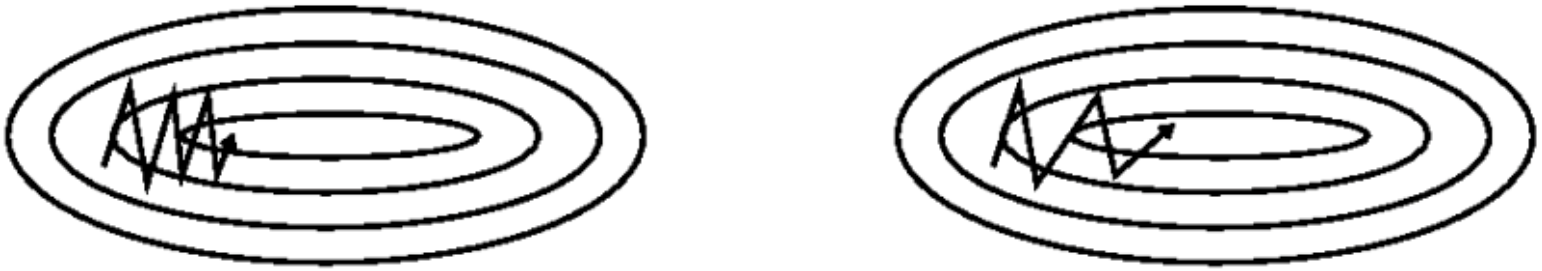
$$\mathbf{m} \leftarrow \beta \mathbf{m} + \eta \nabla_\theta J(\theta)$$



Momentum

❖ **Momentum의 방향성**

- 수직하강하는 GD보다 대각방향으로 최적해를 찾아감



(a) SGD without momentum

(b) SGD with momentum

Figure 2: Source: Genevieve B. Orr

Momentum

❖ Momentum Code

- 텐서플로우에 제공되는 함수로 쉽게 사용 가능
- Learning rate = η
- momemtum = β

Implementing Momentum optimization in TensorFlow is a no-brainer: just replace the `GradientDescentOptimizer` with the `MomentumOptimizer`, then lie back and profit!

```
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
                                       momentum=0.9)
```

Nesterov Accelerated Gradient Algorithm(NAG)

- Momentum은 Local position 에서 시작함
- NAG
  - 가속도벡터를 시작점으로 움직이는 그래디언트

Equation 11-5. Nesterov Accelerated Gradient algorithm

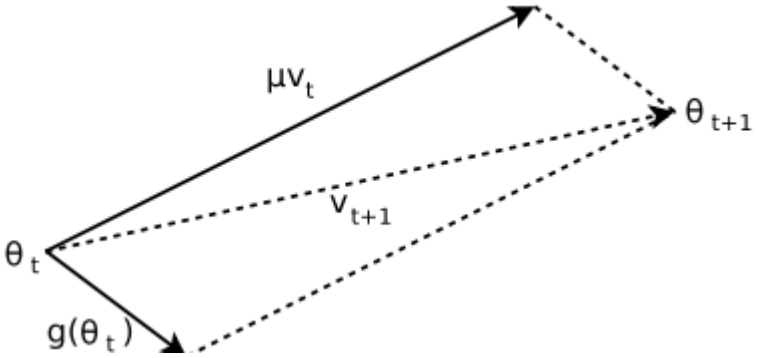
1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta + \beta\mathbf{m})$
2. $\theta \leftarrow \theta - \mathbf{m}$

Nesterov Accelerated Gradient Algorithm(NAG)

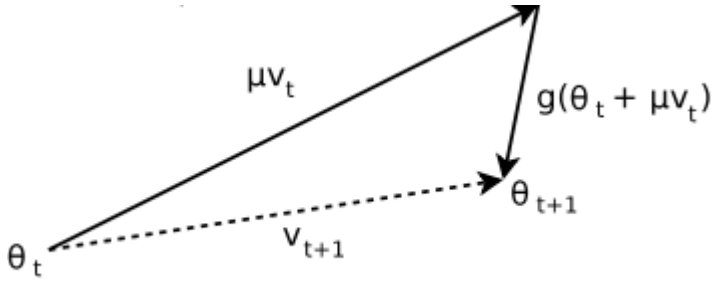
❖ NAG

- 가속도벡터를 시작점으로 움직이는 그래디언트

Momentum



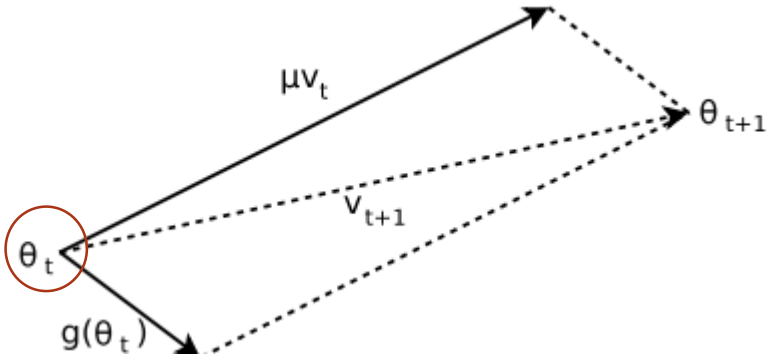
NAG



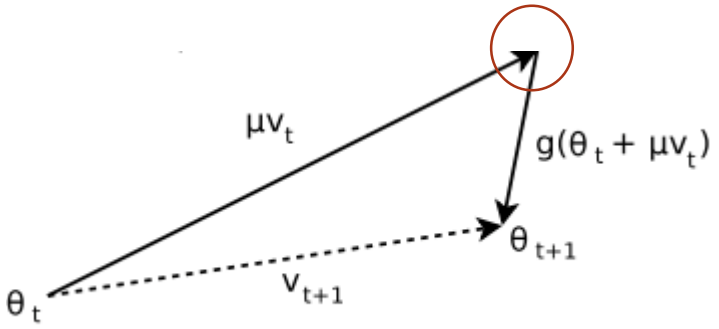
Nesterov Accelerated Gradient Algorithm(NAG)

❖ NAG
- 가속도벡터를 시작점으로 움직이는 그래디언트

Momentum



NAG

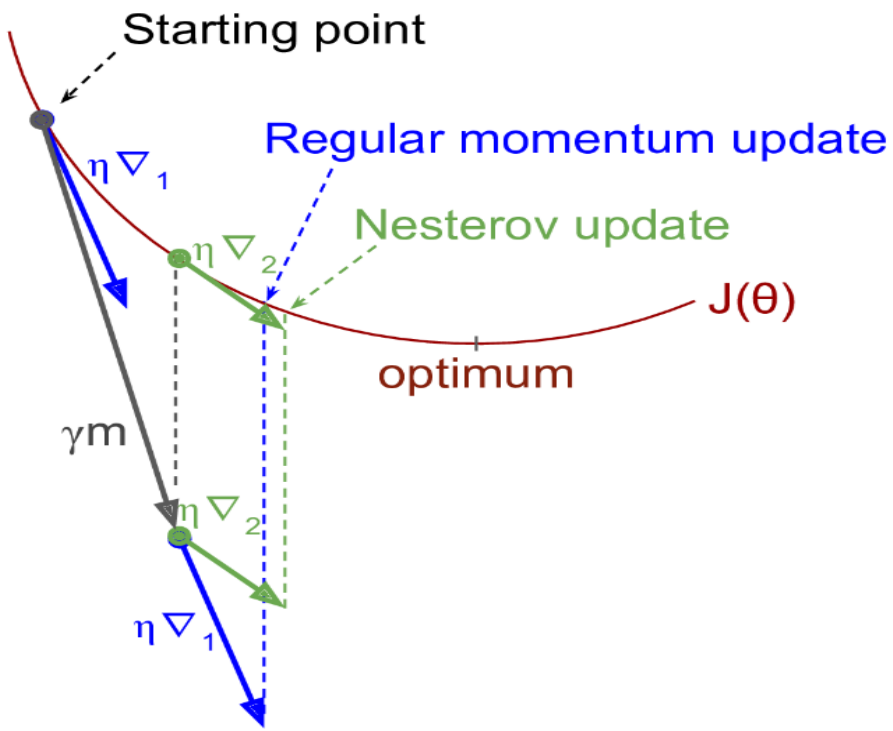


Current gradient
(jittering 이미 많이 벗어난 상태)

Nesterov Accelerated Gradient Algorithm(NAG)

❖ NAD

- 원래 위치의 그래디언트 구한 값(momentum)보다 약간 더 수렴 속도가 빠름



Adaptive Method

Adagrad

- **Adagrad**
  - By scaling down the gradient vector
  - This algorithm decays the learning rate

Equation 11-6. AdaGrad algorithm

1. $\mathbf{s} \leftarrow \mathbf{s} + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

Adagrad

- **Adagrad**
  - By scaling down the gradient vector
  - This algorithm decays the learning rate

Equation 11-6. AdaGrad algorithm

1. $\mathbf{s} \leftarrow \mathbf{s} + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

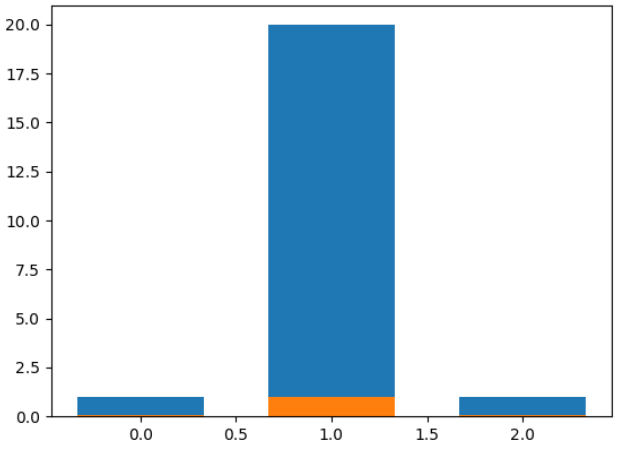
Avoids division by zero: 1e−8

Adagrad

- **Adagrad**
  - By scaling down the gradient vector
  - This algorithm **decays** the learning rate (Too much decrease)

Equation 11-6. AdaGrad algorithm

1. $\mathbf{s} \leftarrow \mathbf{s} + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

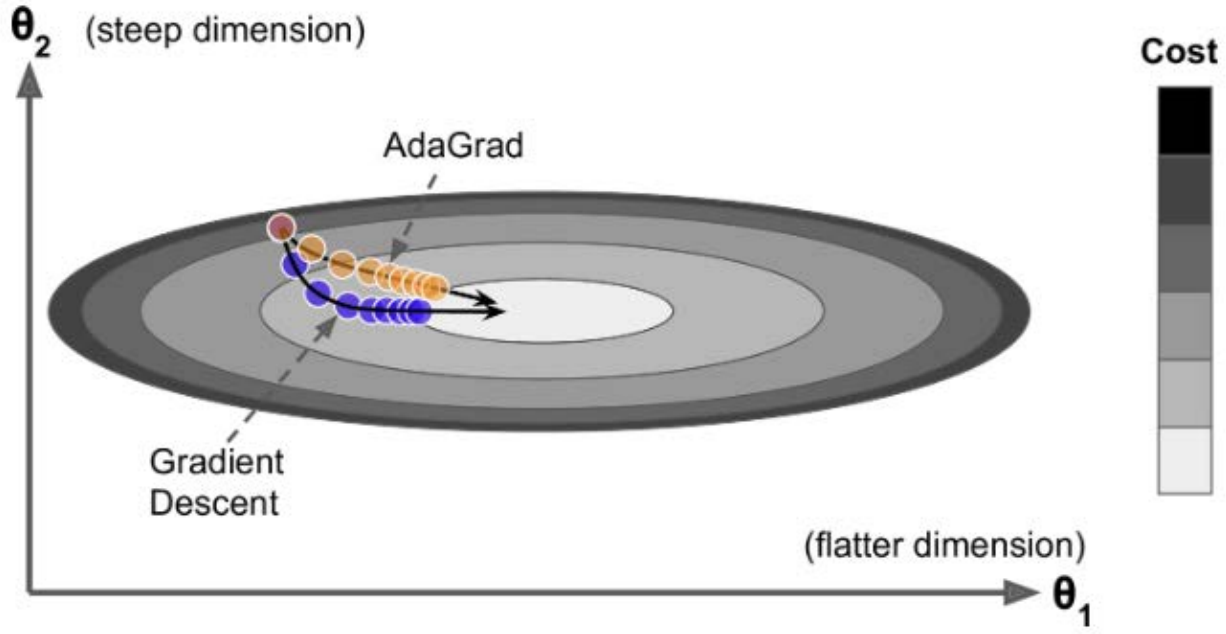


L2 – Normalization effect

```
import matplotlib.pyplot as plt
examples = np.array([1,20,1])
N = len(examples)
x = range(N)
width = 1/1.5
plt.bar(x, examples,width)
plt.bar(x, examples/np.sqrt(sum(np.power(examples, 2))), width)
```

Adagrad

- **Adagrad**
  - Detect this **early** on and correct its direction



RMSProp Method

RMSProp

❖ **Adagrad 문제점**

- 단순한 2차항 문제에는 잘 수행되지만, DNN학습에서 너무 일찍 학습을 멈추게 하는 경향이 있음

❖ **RMSProp**

- by using exponential decay in the first step
- 최근 그래디언트($1-\beta$)와 과거 그래디언트 (β) 를 얼마나 반영할 것인가?

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta\,\nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s}+\epsilon}$

RMSProp

❖ **Adagrad 문제점**

- 단순한 2차항 문제에는 잘 수행되지만, DNN학습에서 너무 일찍 학습을 멈추게 하는 경향이 있음

❖ **RMSProp**

- by using exponential decay in the first step
- 최근 그래디언트($1-\beta$)와 과거 그래디언트 (β) 를 얼마나 반영할 것인가?

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta\,\nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s}+\epsilon}$

Adaptive moment estimation

Adam

- **Adam**
  - Momentum: exponentially decaying sum
  - RMSProp: exponentially decaying average

Momentum

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

decaying sum of past gradients

RMSProp

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta\nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

decaying average of past squared gradients

Adam

- Step 1 : Momentum
  - 차이점: exponentially decaying sum –〉 exponentially decaying average
- Step 2: RMSProp
- Step 5: Momentum+RMSProp

Equation 11-8. Adam algorithm

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_\theta J(\theta)$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
3. $\mathbf{m} \leftarrow \frac{\mathbf{m}}{1 - {\beta_1}^T}$
4. $\mathbf{s} \leftarrow \frac{\mathbf{s}}{1 - {\beta_2}^T}$
5. $\theta \leftarrow \theta - \eta \, \mathbf{m} \oslash \sqrt{\mathbf{s} + \epsilon}$

- T represents the iteration number (starting at 1).

Adam

❖ **Technical Skill**

- 반복학습이 이루어지면서 β^T가 작아지며, m,s값이 작아짐 (과거정보 감소)
- beta1: 0.9 beta2: 0.999

Equation 11-8. Adam algorithm

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_\theta J(\theta)$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
3. $\mathbf{m} \leftarrow \dfrac{\mathbf{m}}{1 - \beta_1^T}$
4. $\mathbf{s} \leftarrow \dfrac{\mathbf{s}}{1 - \beta_2^T}$
5. $\theta \leftarrow \theta - \eta \, \mathbf{m} \oslash \sqrt{\mathbf{s} + \epsilon}$

- T represents the iteration number (starting at 1).

AdaDelta

❖ **AdaDelta**

- RMSProp에서 Momentum으로 확장된 형태
- 그래디언트의 RMS, 모멘텀의 RMS를 고려한 형태

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta\,\nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

AdaDelta

❖ **AdaDelta**

- RMSProp에서 Momentum으로 확장된 형태
- 그래디언트의 RMS, 모멘텀의 RMS를 고려한 형태

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

momentum

AdaDelta

❖ **AdaDelta**

- RMSProp에서 Momentum으로 확장된 형태
- 그래디언트의 RMS, 모멘텀의 RMS를 고려한 형태

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta\,\nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

Momentum의 Mean Square

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

이전 Momentum의 RMS

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

AdaDelta

❖ AdaDelta

- Learning rate 를 고려하지 않고 이전 가속도들의 RMS를 사용
- 텐서플로우에서 learning rate가 존재
  - Learning rate = 1로 설정

Equation 11-7. RMSProp algorithm

1. $\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \eta\,\nabla_\theta J(\theta) \oslash \sqrt{\mathbf{s} + \epsilon}$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

Equation 11-4. Momentum algorithm

1. $\mathbf{m} \leftarrow \beta\mathbf{m} + \eta\nabla_\theta J(\theta)$
2. $\theta \leftarrow \theta - \mathbf{m}$

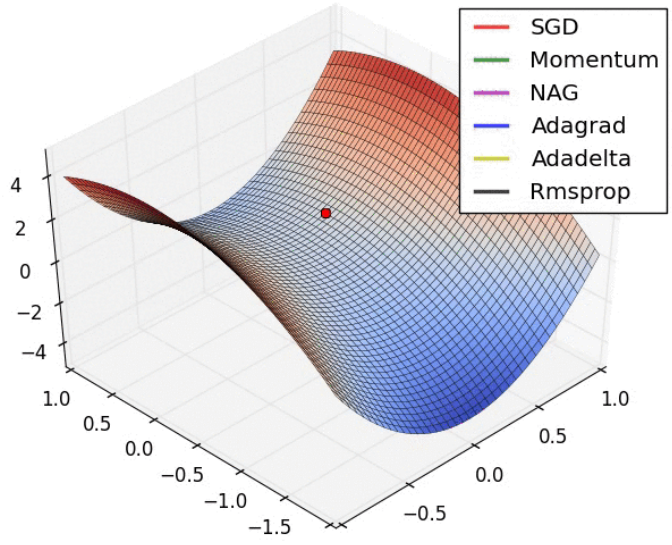
이전 Momentum의 RMS

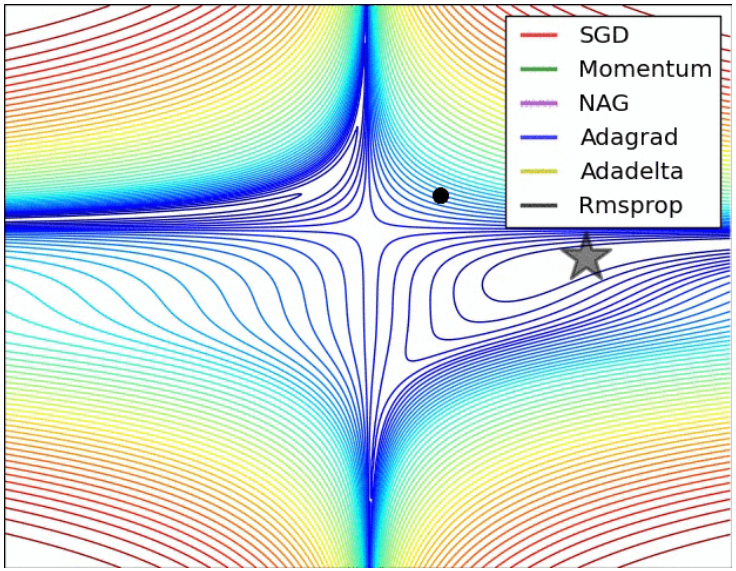
$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Faster Optimizers

- Visualization





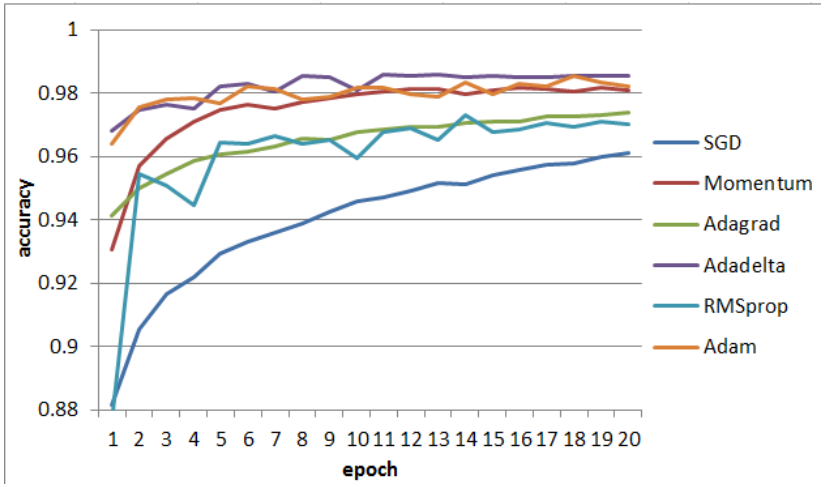
Adam vs Adadelta

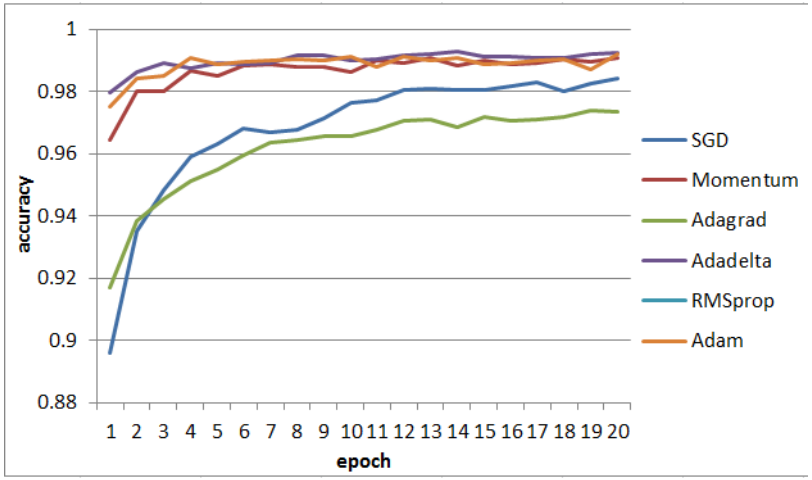
[–] **mr_robot_elliot** 3 점 1 년 전에

Adam has a stepsize to be tuned but adadelta doesn't need any. However i found Adam to be converging faster in my experiments. Yeah even i find it surprising that they didnt include Adadelta probably they want to show the methods which has stepsize and i think Adam is more robust to stepsize compared to other methods. And theoretically Adam is more structured but in Adadelta there is no convergence or regret guarantees, its like we just have to believe it from empirical results!. However Adadelta raises some of the serious issues with first order methods that the units of updates and parameters are imbalanced.(Check section 3.2 in Adadelta paper). In Adam there is bias correction towards the moments which might probably result in faster convergence as you would be having bigger gradients in the beginning where bias correction really plays a major role.So you get close to optima fast and after some steps you slow down due to the first order moment and here bias correction term might have already nullified i.e (has become 1). I am still working on intuition for Adadelta it would be great if someone can mention here.

[–] **siblbombs** 2 점 1 년 전에

I've found adadelta to be faster than adam the times I've tested it, although it certainly wasn't an exhaustive evaluation. The one major exception is if you are training an embedding matrix, then adam is much better than adadelta in my experience.

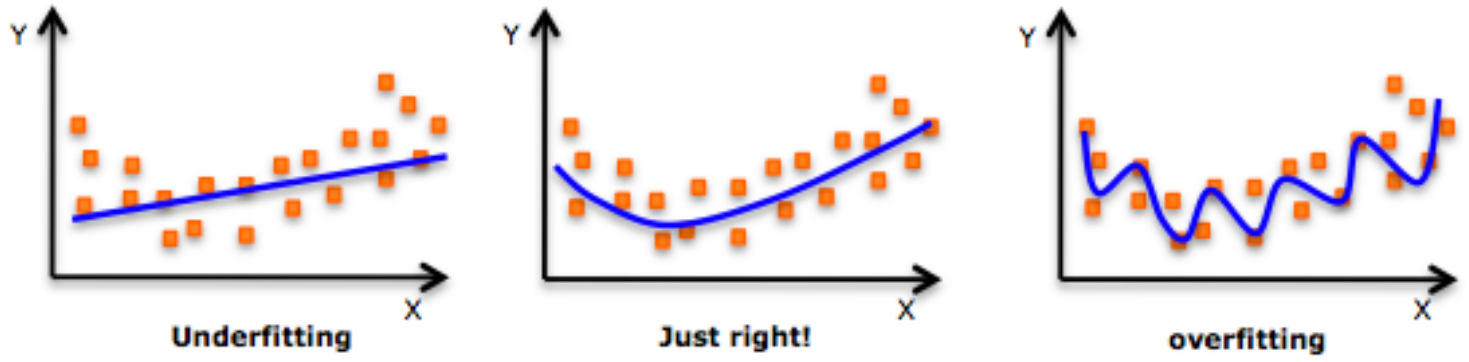




Contents

Avoiding Overfitting Through Regularization

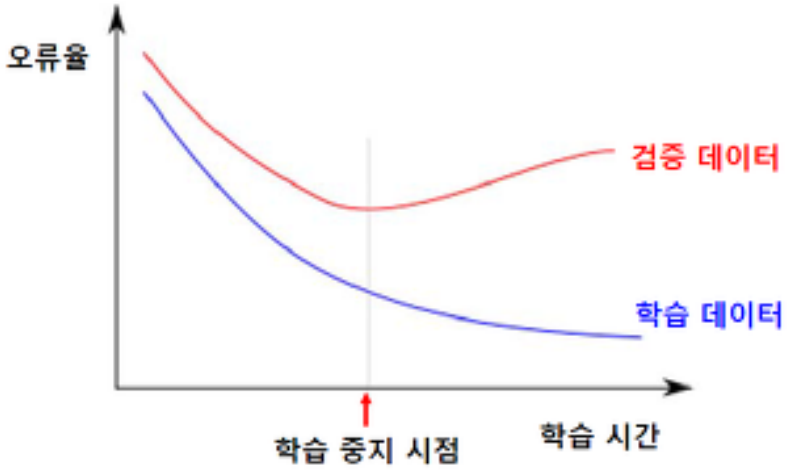
- Early stopping
- $L1$, $L2$ Regularization
- Dropout
- Max-norm Regularization
- Data augmentation



Early stopping

- 검증데이터에 대한 성능이 떨어질 때 학습을 멈추는 방법
- 텐서플로우는 Step 50마다 검증데이터를 평가후 저장
  - if 'Winner' snapshot (t) 〉 'Winner' snapshot ($t-1$)
  - Step t를 기록해 둠
- 다른 Regularization와 함께 쓰임

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

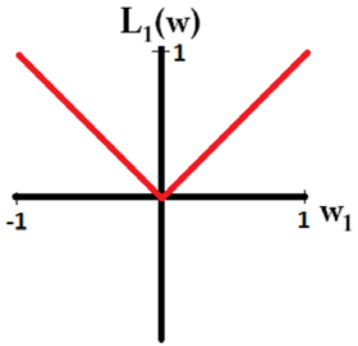


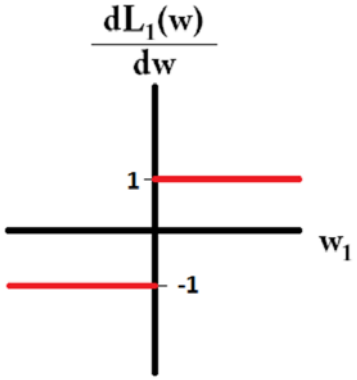
$L1$, $L2$ Regularization (1)

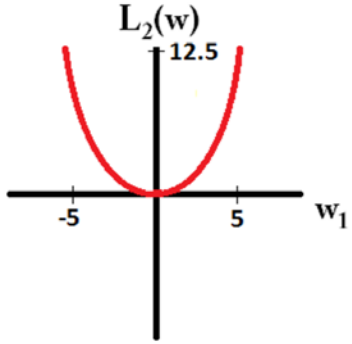
- Sparse model: 많은 가중치(W)들이 0으로 수렴
- L1-regularization is more likely to create 0-weights
  - Weights $(w_1, w_2, \dots, w_m)$

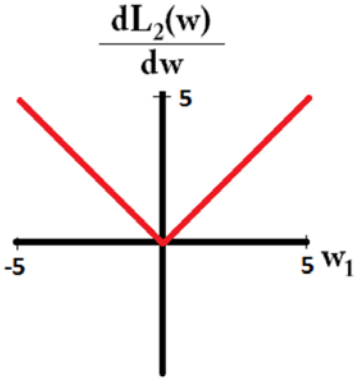
$$L_1(w) = \Sigma_i |w_i| \qquad \frac{L_1(w)}{dw} = sign(w)$$

$$L_2(w) = \frac{1}{2}\Sigma_i w_i^2 \qquad \frac{L_2(w)}{dw} = w$$



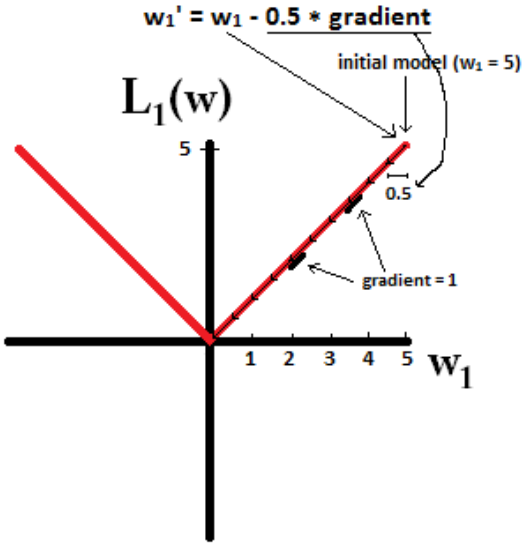






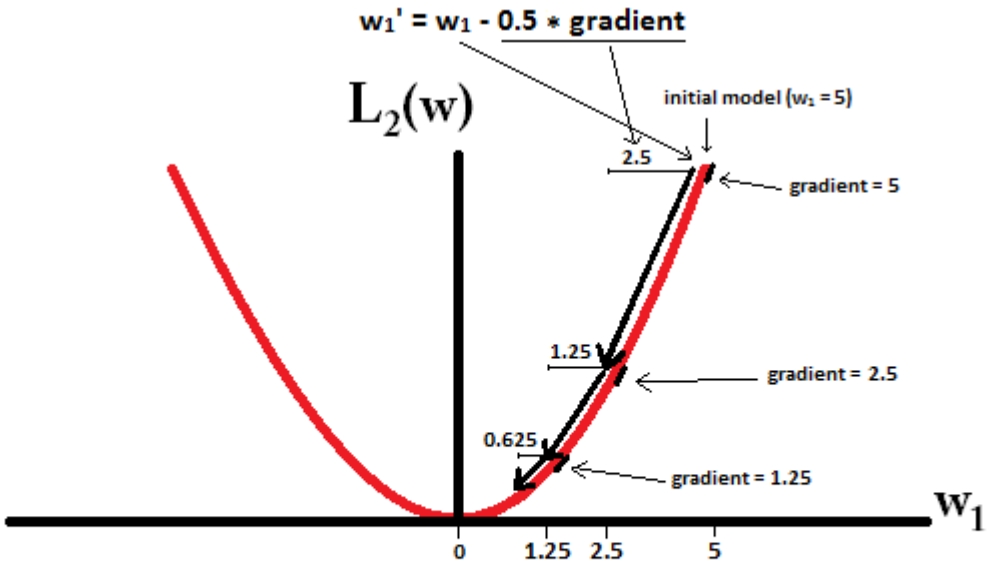
$L1$, $L2$ Regularization (2)

- L1 Regularization (a single parameter)
  - Move any weight towards 0, with the same step size, regardless the weight's value
  - e.g. $w_1 = 5$, $\eta = \frac{1}{2}$
  - $w_1 := w_1 - \eta * \frac{dL_1(w)}{dw} = w_1 - 0.5 * 1$



$L1$, $L2$ Regularization (3)

- **L2 Regularization (a single parameter)**
  - Move any weight towards 0, with the halfway step size, depended on the weight's value
  - e.g. $w_1 = 5$, $\eta = \frac{1}{2}$
  - $w_1 := w_1 - \eta * \frac{dL_1(w)}{dw} = w_1 - 0.5 * w_1$



Dropout (1)

❖ 회사직원이 무슨 일을 할지 동전으로 결정한다면 그 경영은 효과적일까?

Single Case

Which task?

Robust

Multiple Case

Which task?

Dropout (2)

❖ **각 학습단계마다 유일한 하나의 뉴런네트워크를 생산한다면?**

- 10,000 training steps에서 10,000 다른 뉴런네트워크들을 생성
- 그 뉴런네트워크들은 상호 의존적인 관계지만 다른 것들 임
- 각 단계에서 P = 0.5 확률로 뉴런들을 제거

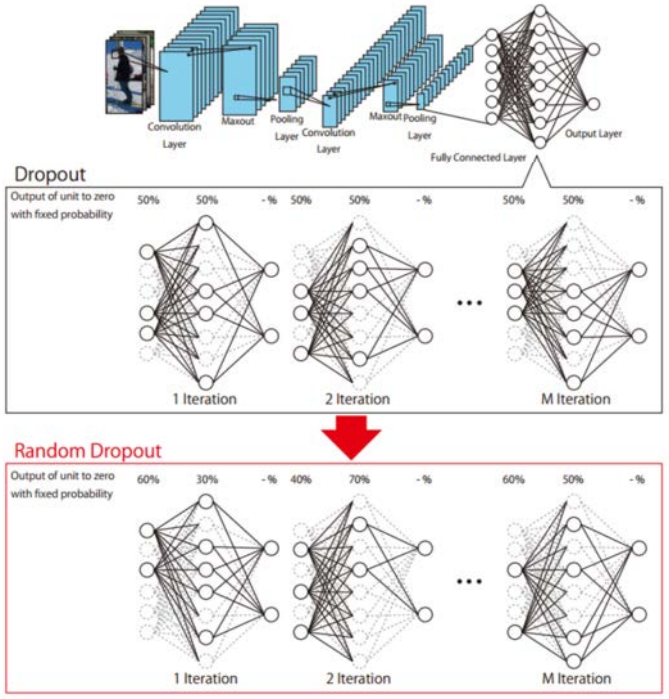
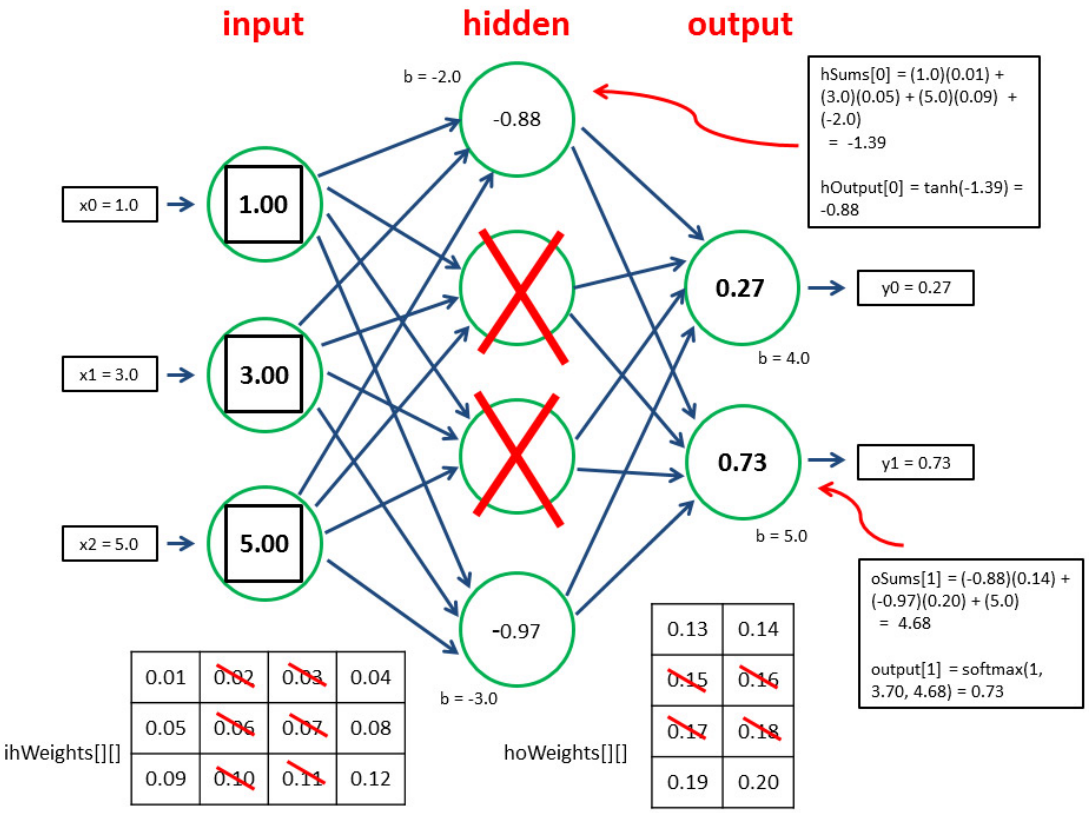


Fig. 2. Algorithm of conventional Dropout and Random Dropout.

Dropout (3)

❖ **Testing 뉴런들은 입력 뉴런만큼(no dropout) 많은 연결들로 이루어져 있음**

- 학습 후에 $w \times (1-p)$로 조정 # $(1-p)$:생존확률
- 학습 중에 뉴런의 아웃풋 $\div (1-p)$로 조정



Data augmentation

❖ **Random crops and scales**

- Training:
  - [256, 480]범위의 이미지를 랜덤으로 추출
  - 길이가 짧은 쪽으로 학습 이미지를 Resize
  - 224x224 patch in ResNet
- Testing:
  - 5 가지 크기로 이미지 크기 조정 {224, 256, 384, 480, 640}
  - 각 이미지마다 4 corners+ center, +flips 을 224x224로 crop하여 사용