

# Introduction to neural network

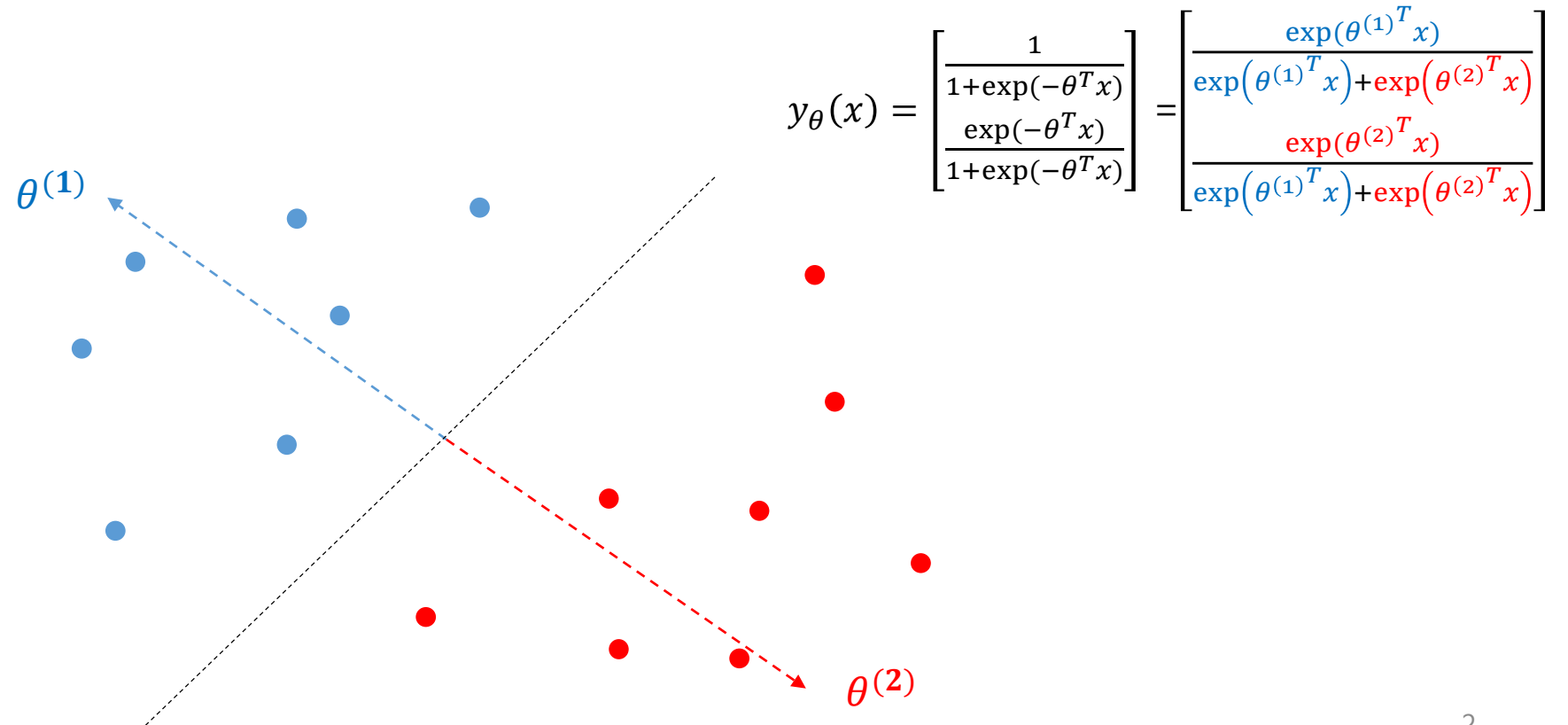
Hyunjoong Kim

[soy.lovit@gmail.com](mailto:soy.lovit@gmail.com)

[github.com/lovit](https://github.com/lovit)

# XOR problem

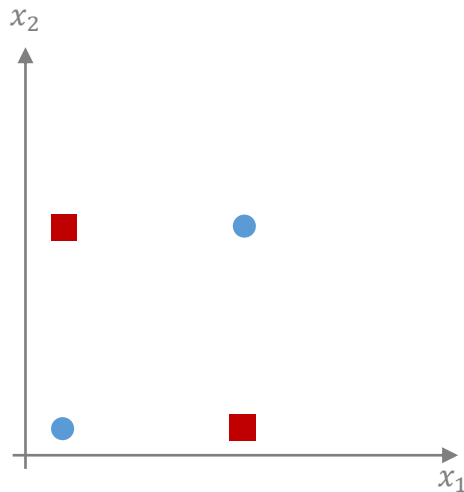
- Softmax regression 은 클래스별 대표벡터  $\theta^{(j)}$  를 이용하여 데이터를 분류합니다.
- 하나의 직선 경계면으로 데이터가 분류될 때 "**linear separable**" 하다고 표현합니다.



# XOR problem

---

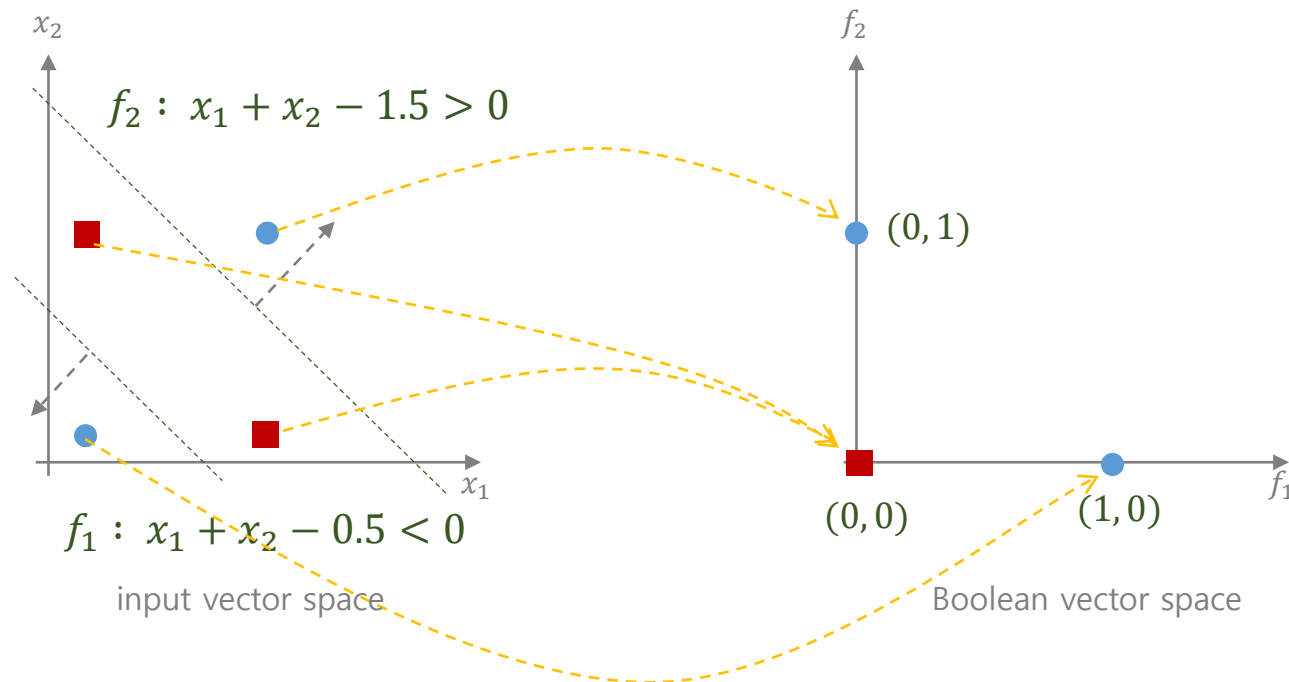
- 하나의 직선 경계면으로 분류되지 않는 데이터를 “linear inseparable” 이라 합니다.
  - 곡선 경계면을 이용하던지 (nonlinear classifiers)
  - 데이터의 벡터 표현 (representation) 을 선형으로 변환해야 합니다 (neural network).



input vector space

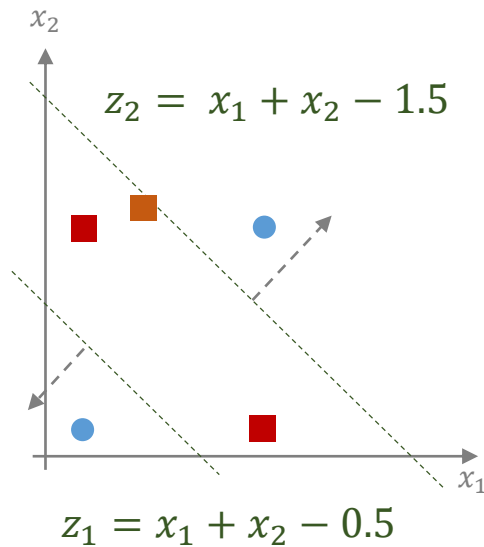
# XOR problem

- 두 개의 직선을 이용하면 아래 데이터를 구분할 수 있습니다.
  - (0, 0) 은 빨간색, 그 외는 파란색으로 데이터를 구분할 수 있습니다.
  - Perceptron 은 Boolean 함수를 이용하여 데이터의 representation 을 변형합니다.

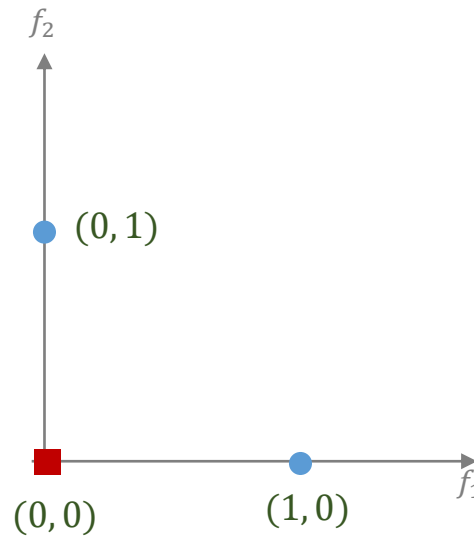


# XOR problem

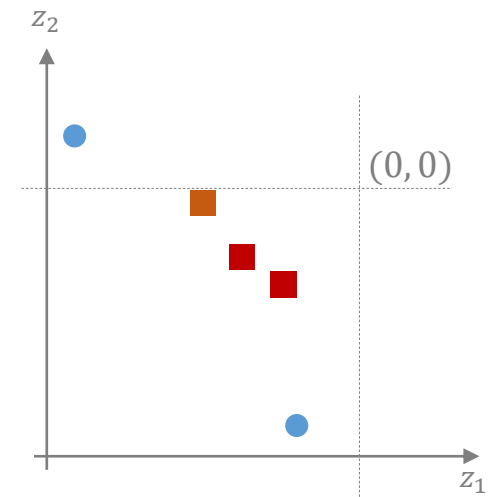
- 두 개의 직선을 이용하면 아래 데이터를 구분할 수 있습니다.
  - Boolean 함수는 지나치게 많은 정보를 잃어버립니다.
  - 선형 결합 ( $z_2$ ) 은 더 풍부한 정보를 표현하지만, 범위가  $(-\infty, \infty)$  이며, 복잡한 비선형 공간을 표현하기 어렵습니다.



input vector space



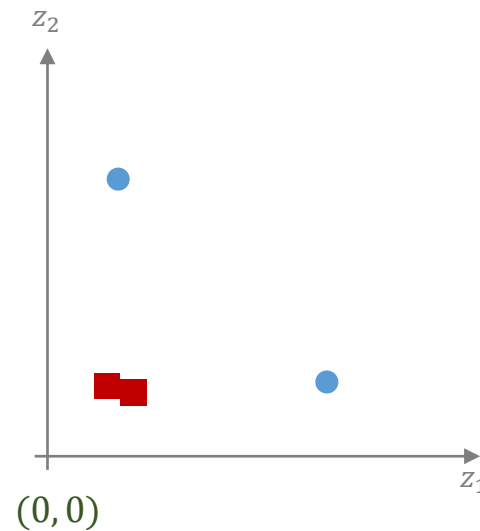
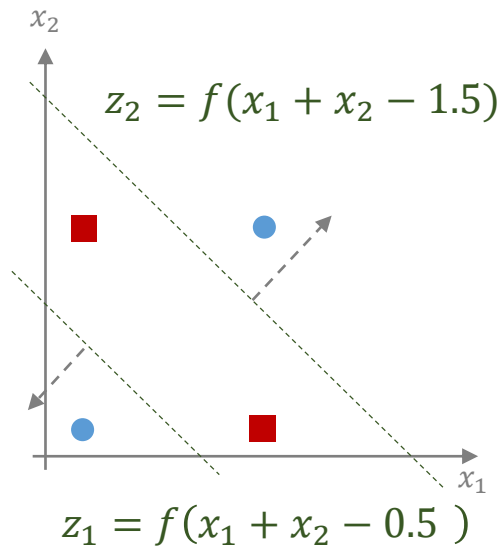
Boolean vector space



continuous vector space

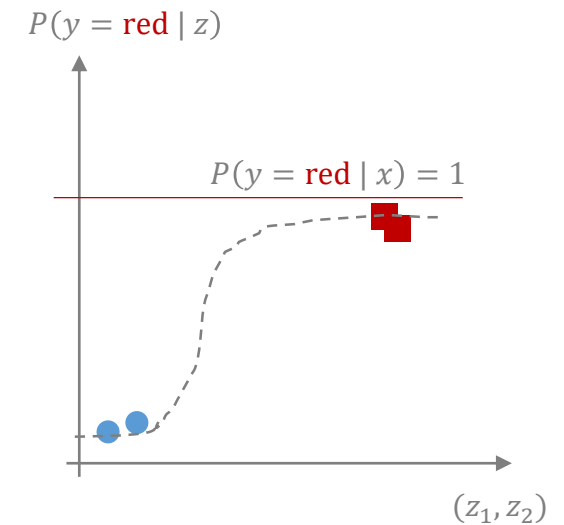
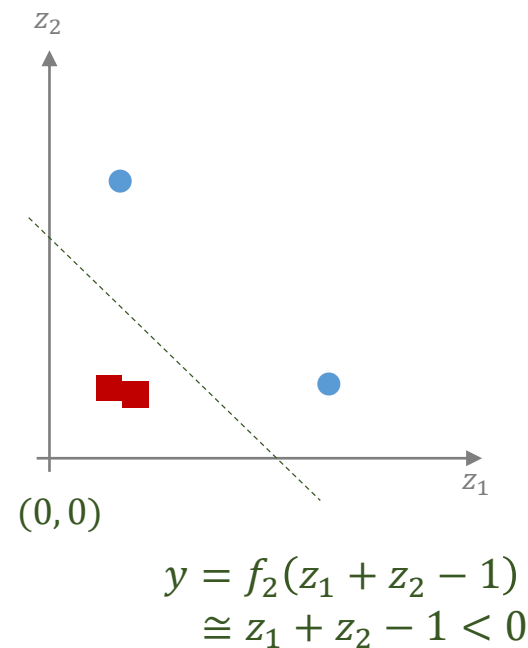
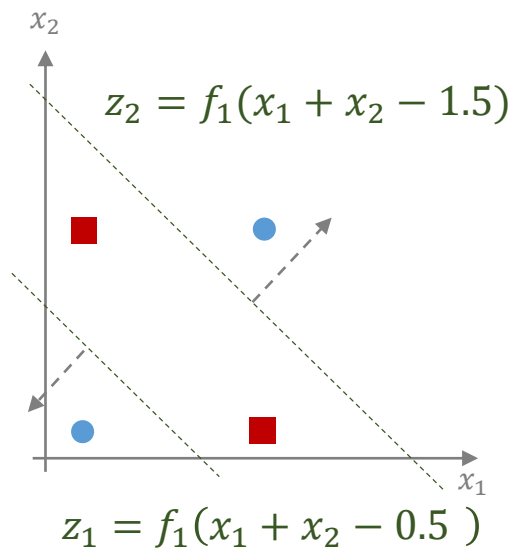
# XOR problem

- 비선형 함수 (activation function,  $f$ ) 를 이용하면 제한된 범위의 공간에서, 비선형 공간을 표현하는 새로운 representation 을 얻을 수 있습니다.
- Sigmoid,  $f(x) = \frac{1}{1+\exp(-x)}$  함수가 이용될 수 있습니다.



# XOR problem

- $f_1$  을 이용하여  $x \rightarrow z$  로 representation 을 변경한 뒤,  $z$  에 의한 함수  $f_2(z)$  를 이용하여 데이터를 분류 합니다.



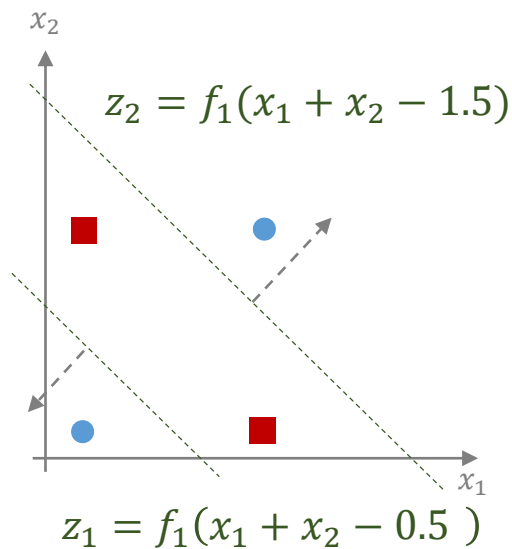
# XOR problem

- 이러한 판별 모델을 feed-forward neural network 라 합니다.

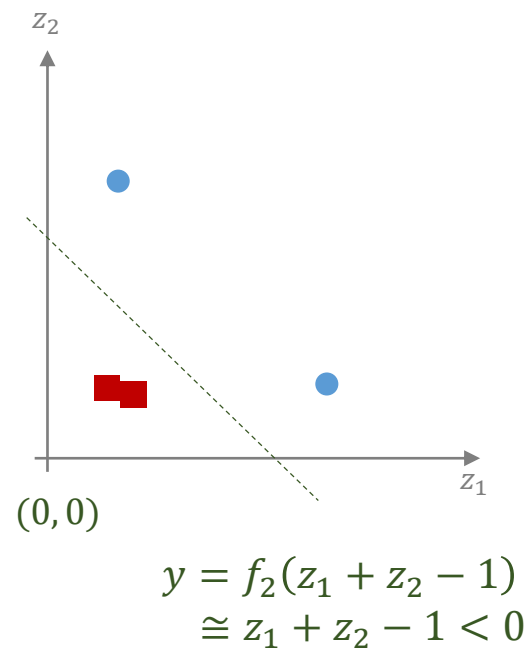
- 패러미터들을 weights 라 부르기도 합니다.

- $h_1 = f_1(A_1x + b_1)$

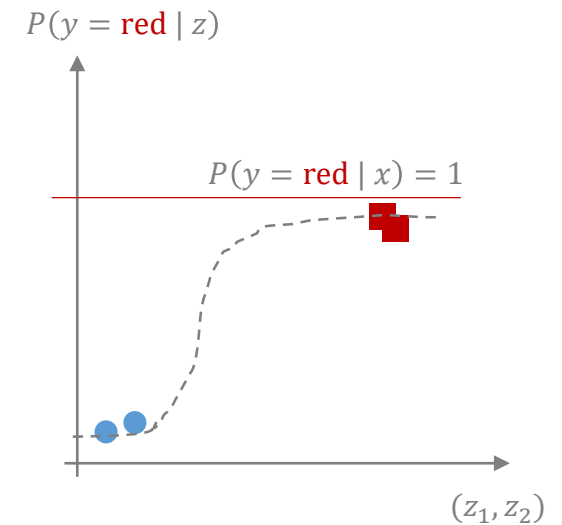
$$\hat{y} = f_2(A_2h_1 + b_2) = f_2(A_2f_1(A_1x + b_1) + b_2)$$



input layer



hidden layer

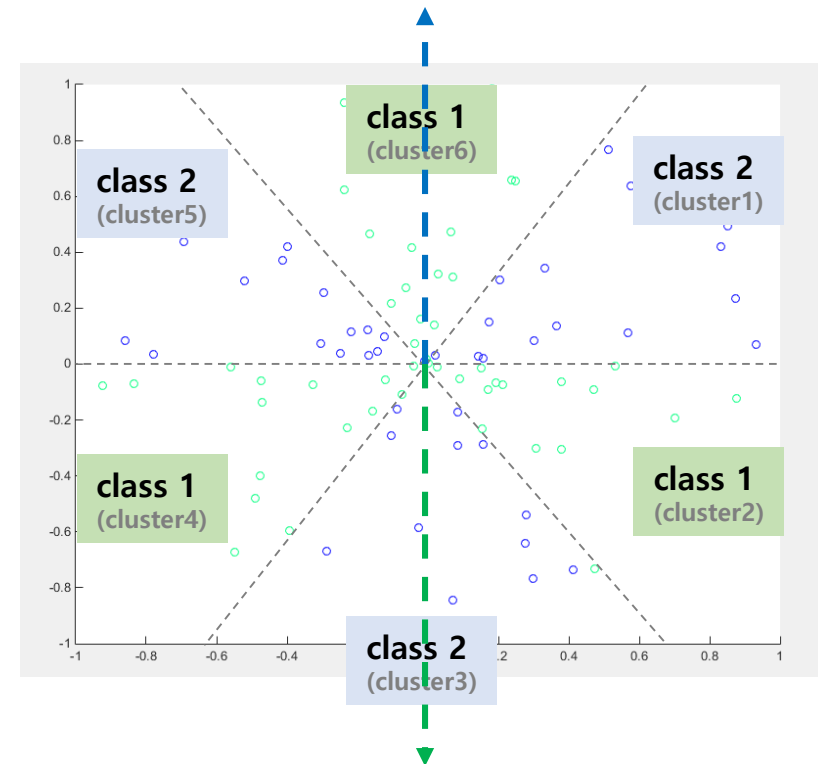
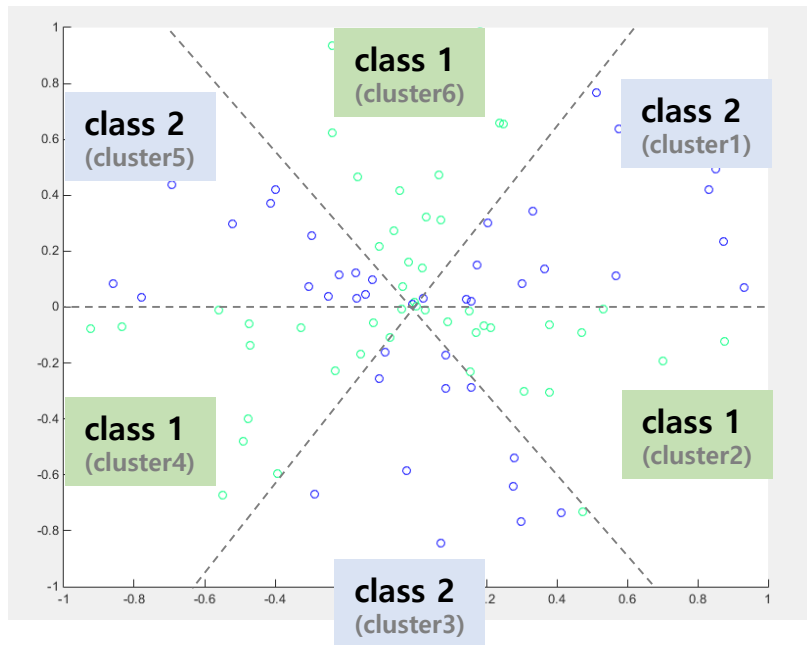


output layer



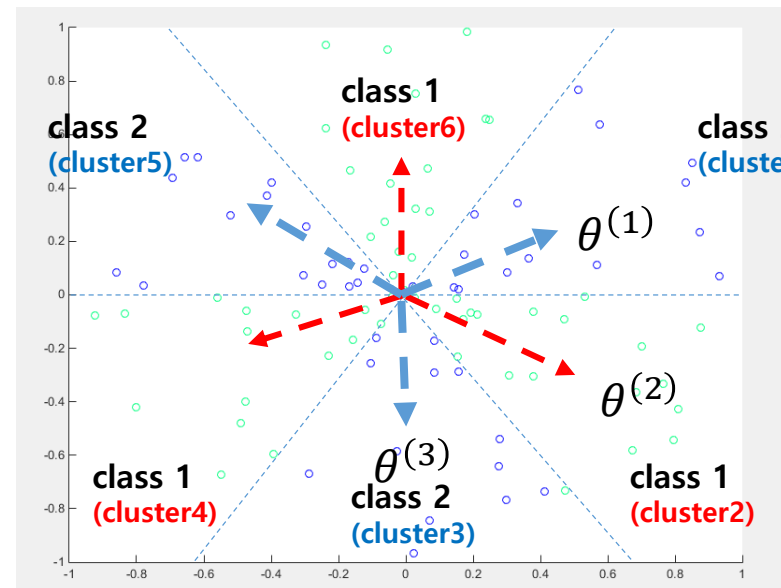
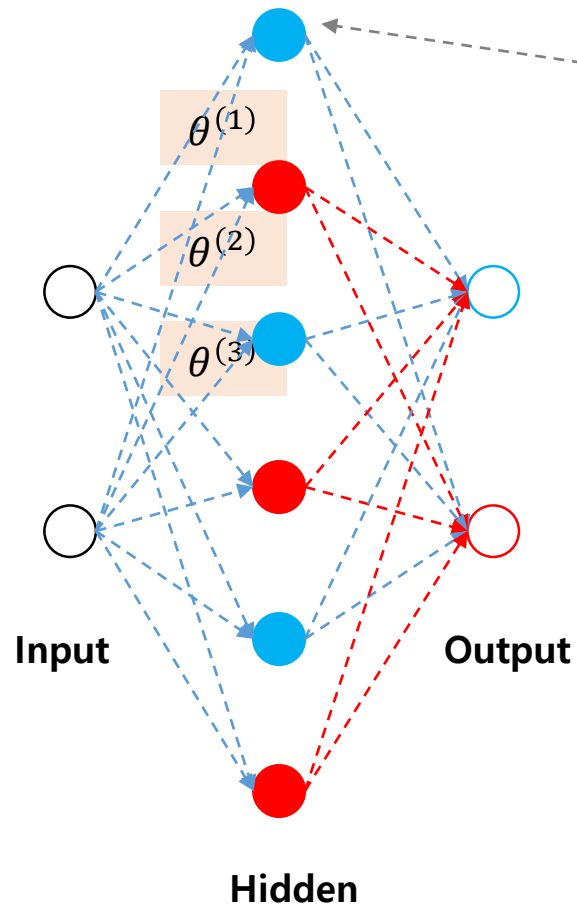
# Feed-forward Neural Network

- 이러한 방법은 공간이 복잡하여도 잘 작동합니다. 2 개의 클래스에 속한 데이터가 6개의 지역으로 나뉘어 분포한 데이터입니다.
- Logistic regression 은 두 개의 대표벡터만 학습하기 때문에 분류를 할 수 없습니다.



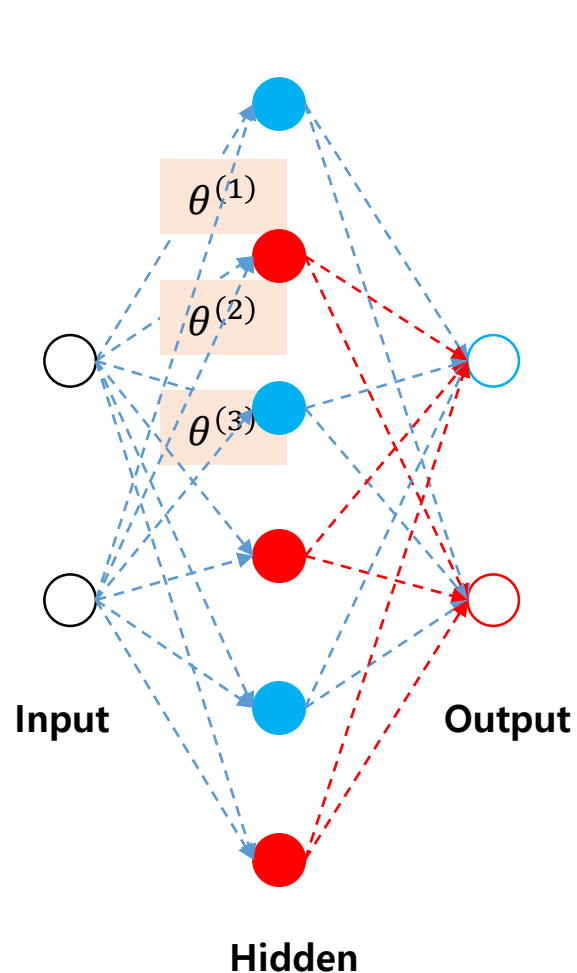
# Feed-forward Neural Network

- 각 지역별로 하나의 대표벡터를 두고,  $z_j = \frac{1}{1+\exp(-x^T\theta_j)}$  로 변형해 봅니다.



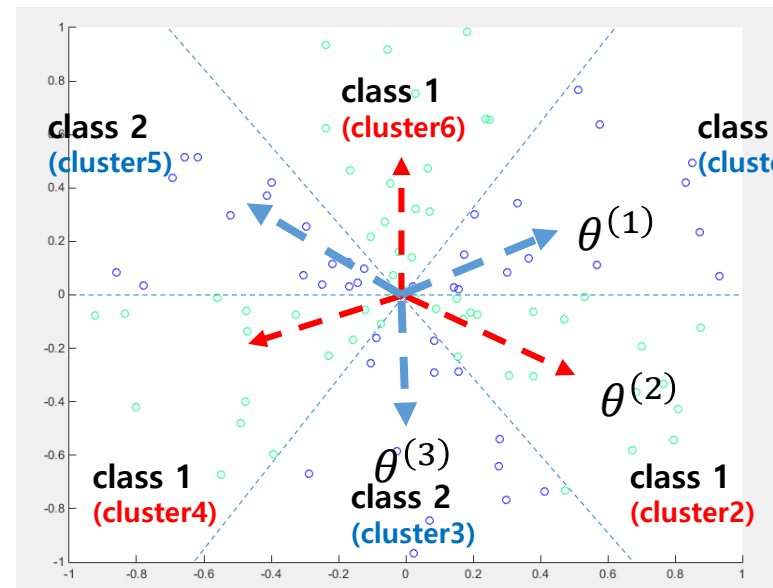
# Feed-forward Neural Network

- Sigmoid,  $f = \frac{1}{1+\exp(-x)}$  는 데이터를 Boolean vector 처럼 변형시킵니다.



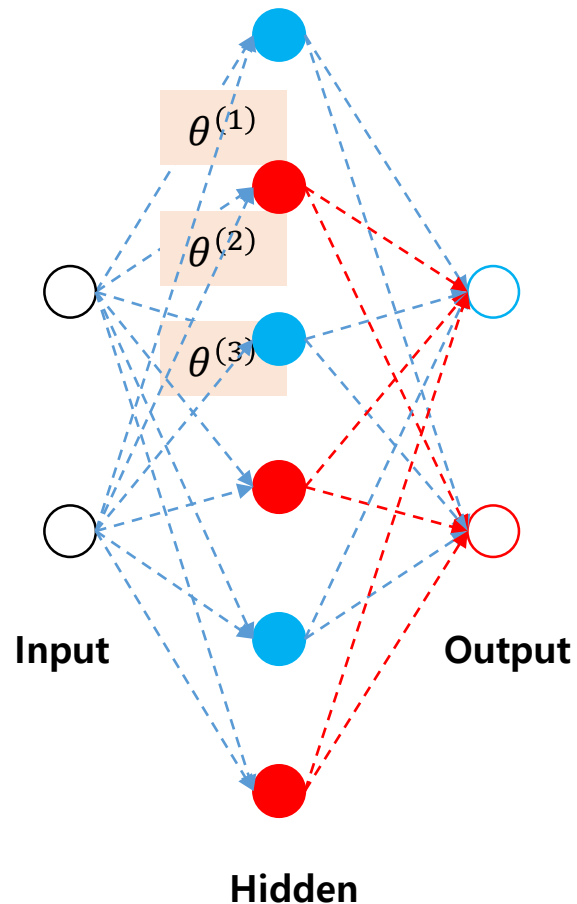
Hidden to output parameters

$$h^{(1)} = [1, \quad 0.3, \quad 0.1, \quad 0.03, \quad 0.1, \quad 0.3]$$



# Feed-forward Neural Network

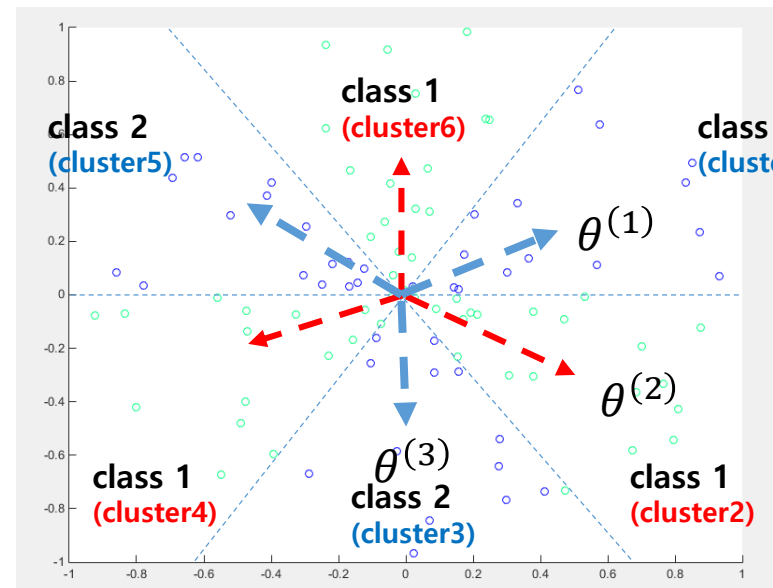
- 6 차원으로 변한 데이터는 linear separable 입니다.



Hidden to output parameters

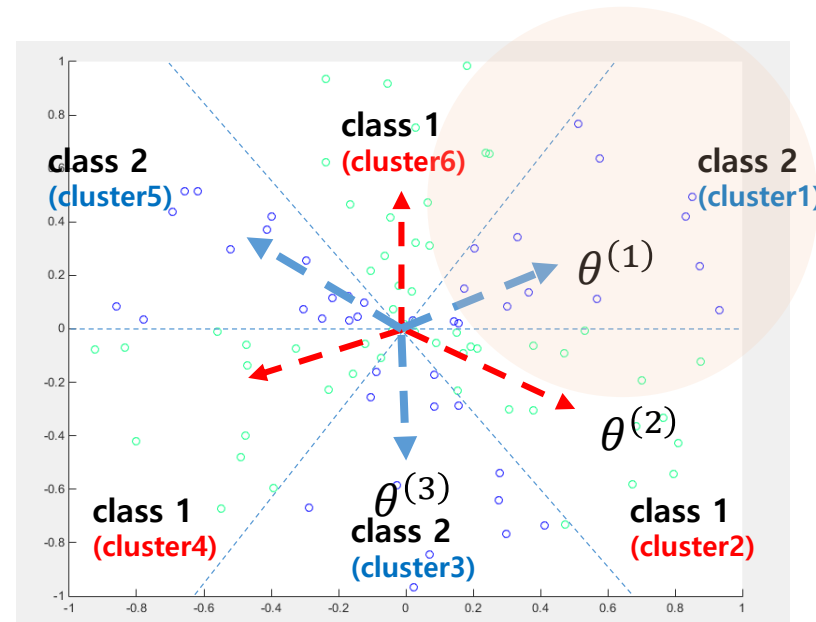
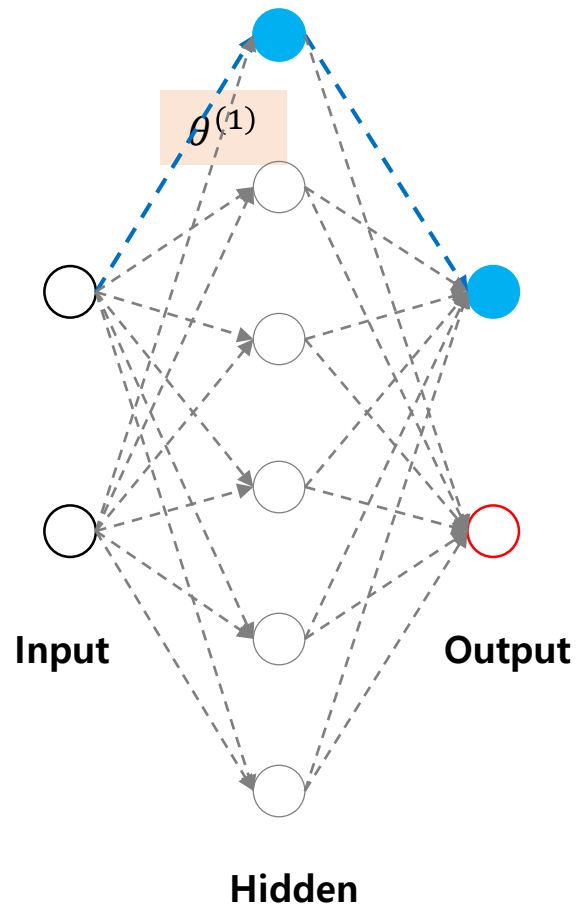
$$h^{(1)} = [1, \quad 0.3, \quad 0.1, \quad 0.03, \quad 0.1, \quad 0.3]$$

$$\theta^{h \rightarrow o_1} = [1, \quad -1, \quad 1, \quad -1, \quad 1, \quad -1]$$



# Feed-forward Neural Network

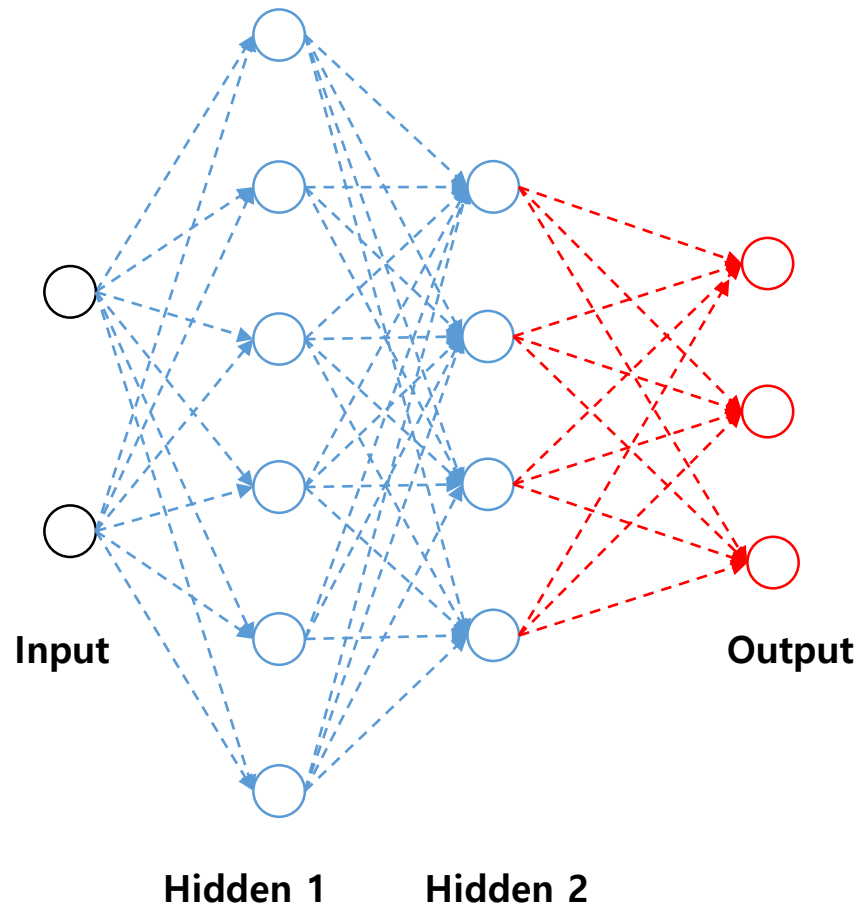
- 하나의 hidden unit 은 한 지역을 표현합니다. 군집화와 비슷합니다.
  - Cluster 1에 속하면 파란색 클래스일 가능성이 높습니다.



# Feed-forward Neural Network

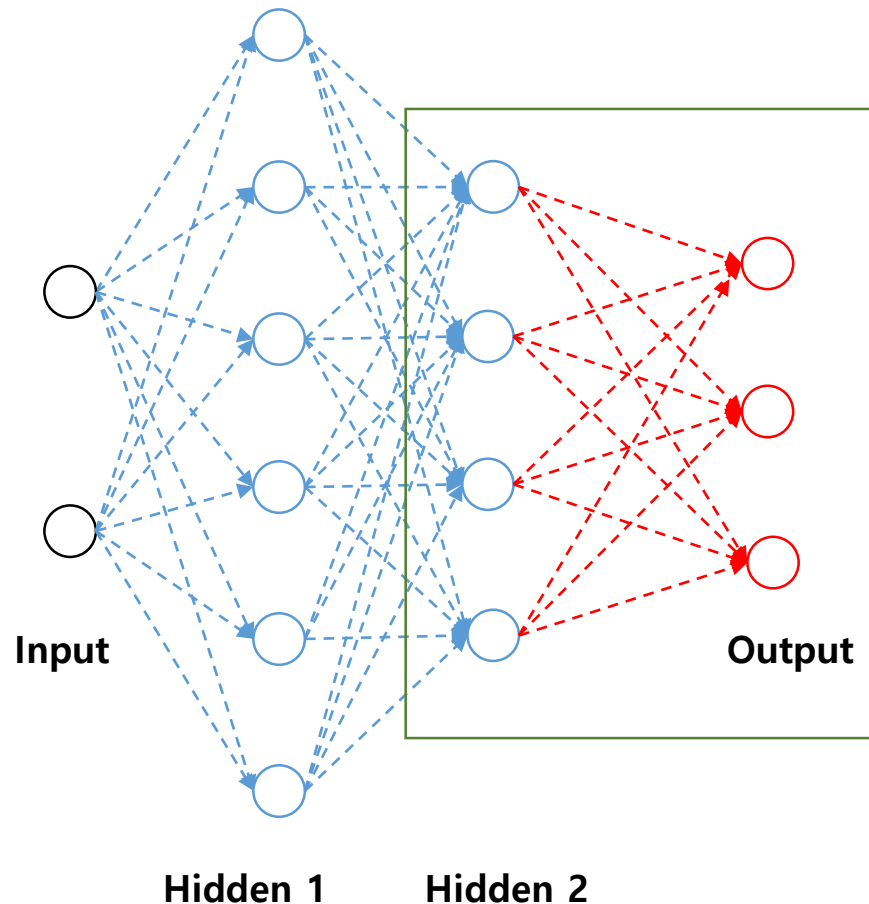
---

- 여러 개의 hidden layers 는 여러번의 공간 변형을 통하여 문제를 linear separable 하게 변형하는 것입니다. (2차원 데이터  $\rightarrow$  6  $\rightarrow$  4 차원 hidden, 3 classes 분류)



# Feed-forward Neural Network

- 여러 개의 hidden layers 는 여러번의 공간 변형을 통하여 문제를 linear separable 하게 변형하는 것입니다. (2차원 데이터  $\rightarrow$  6  $\rightarrow$  4 차원 hidden, 3 classes 분류)



Hidden 2 의 4차원이 linear separable 하다면,  
Softmax (hidden2  $\rightarrow$  output) 은 잘 작동합니다.

# Components

---

- 뉴럴 네트워크를 구성할 때에는 대표적으로 다음을 결정해야 합니다.
  - loss (objective) : 모델의 성능 측정 기준
  - structure : hidden layers 개수, units 크기, activation functions
  - optimizer : 모델 학습 방법
  - initializer : 패러미터 초기화 방법
  - training style



# Loss function (objective)

---

- 문제에 따라 다양한 “모델 품질 평가 기준”이 존재합니다.
  - 앞서 살펴본 기준들이 이용될 수 있습니다.
  - Regression 문제는 다음의 기준들이 있습니다.
    - Mean Squared Error :  $\frac{1}{2n} \sum (y - \hat{y})^2$
    - Mean Absolute Percentage Error :  $\frac{1}{n} \sum_i \frac{|\hat{y}_i - y_i|}{|y_i|}$
    - etc ...

# Loss function (objective)

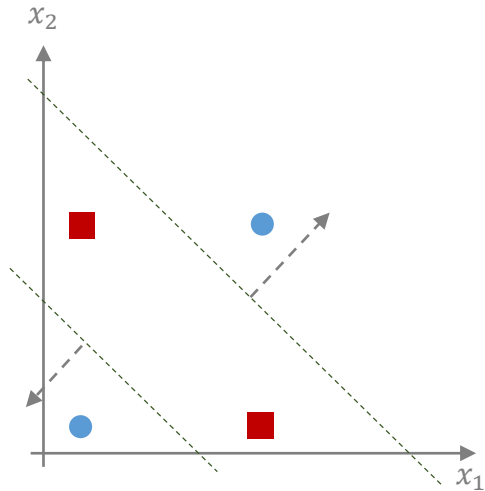
---

- 문제에 따라 다양한 “모델 품질 평가 기준”이 존재합니다.
  - Classification 문제는 다음의 기준들이 있습니다.
    - Negative Log-Likelihood :  $\sum_{i=1}^n \sum_{j=1}^K I(y_i = j) \left( -x_i^T \beta_j + \log \sum_{j=1}^K \exp(x_i^T \beta_j) \right)$ 
      - cross entropy loss 라 불립니다.
    - Hinge loss
    - Kullback-Leibler Divergence (KL divergence)
    - etc ...

# Structure

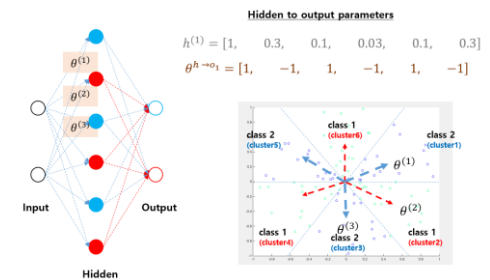
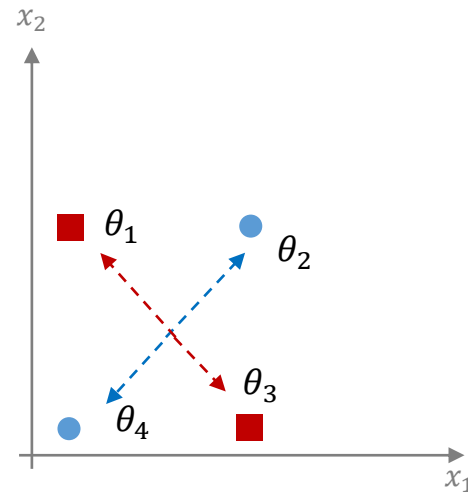
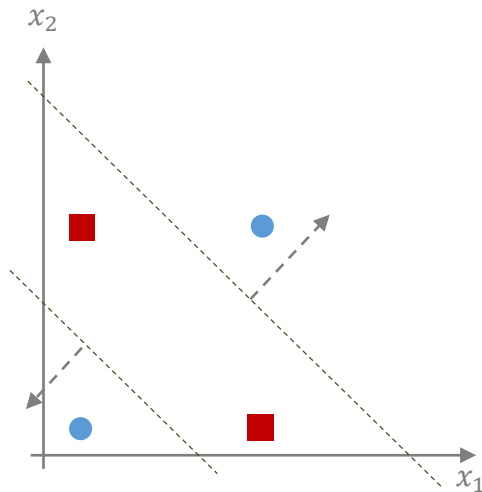
---

- 예시 데이터는 2 차원이기 때문에 적절한 hidden layers/units 을 설정할 수 있지만, 실제 데이터는 고차원이기 때문에 이를 정의하기 어렵습니다.



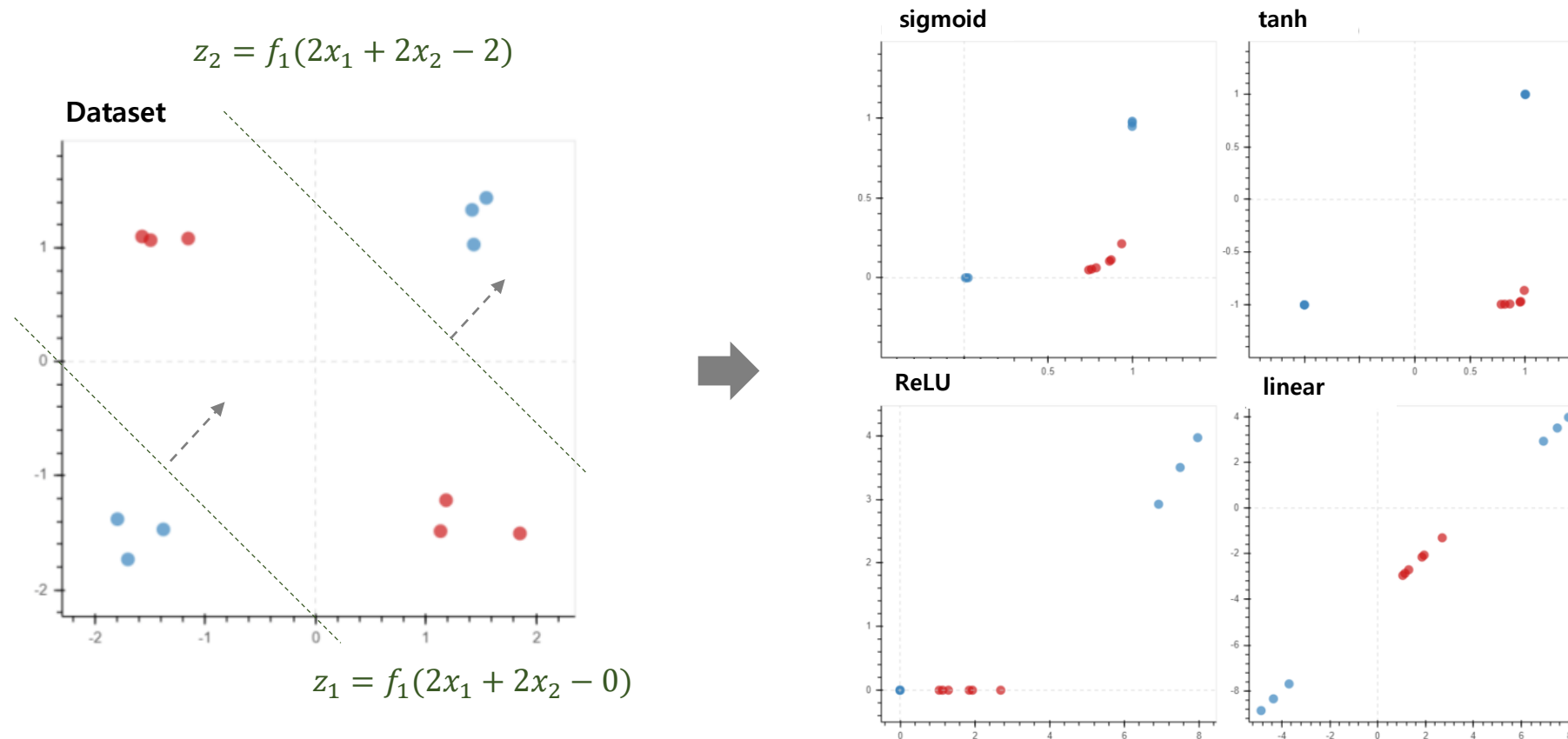
# Structure

- 4 차원 hidden 이 2 차원 hidden 보다 학습이 잘 이뤄질 수 있습니다.
  - 하지만 hidden layers/units 이 지나치게 크면 과적합이 일어날 수 있습니다.
  - 데이터가 충분할 경우, 성능 평가를 통하여 모델 구조를 설정해야 합니다.



# Activation functions

- 활성함수 (activation functions) 에 따라서 hidden vector 의 분포가 달라집니다.



# Activation functions

- 대표적으로 많이 이용되는 활성화함수입니다.

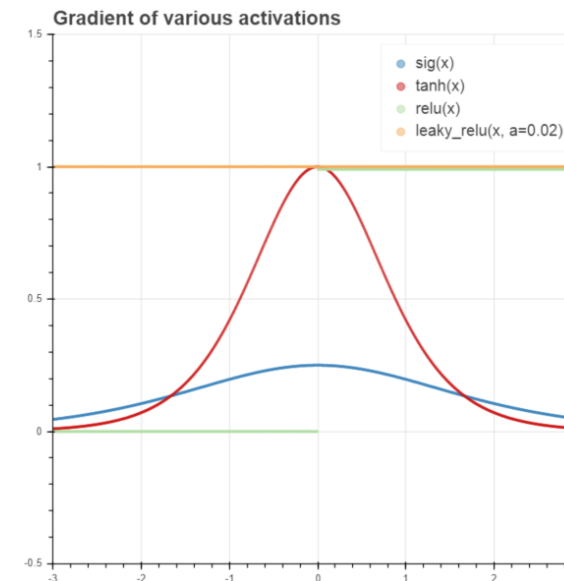
Name	$f(x)$	$f(x)'$	range
Sigmoid	$f(x) = \frac{1}{1 + \exp(-x)}$	$f(x) \times (1 - f(x))$	$(0, 1)$
Hyper-tangent (tanh)	$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$	$1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit (ReLU)	$f(x) = \max(0, x)$	$f(x)' = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$	$(0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} \max(0, x) & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases}$	$f(x)' = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{if } x \leq 0 \end{cases}$	$(-\infty, \infty)$

# Activation functions

- 처음에는 sigmoid  $f(x) = \frac{1}{1+\exp(-x)}$  함수가 자주 이용되었습니다. 그러나 gradient 가 잘 전달되지 않아 여러 층의 hidden layers 가 쌓이면 학습이 어려웠습니다.

- 이를 해결하기 위하여  $\tanh f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ , ReLU  $f(x) = \max(0, x)$ ,

Leaky-ReLU  $f(x) = \begin{cases} \max(0, x) & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases}$  등이 이용되었습니다.



# Activation functions

---

- 처음에는 sigmoid  $f(x) = \frac{1}{1+\exp(-x)}$  함수가 자주 이용되었습니다. 그러나 gradient 가 잘 전달되지 않아 여러 층의 hidden layers 가 쌓이면 학습이 어려웠습니다.
- 이를 해결하기 위하여  $\tanh f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ ,  $\text{ReLU } f(x) = \max(0, x)$ ,  
 $\text{Leaky-ReLU } f(x) = \begin{cases} \max(0, x) & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases}$  등이 이용되었습니다.
- 최근에는  $\tanh$ , (leaky) ReLU 들이 자주 이용되며, 특정 네트워크에서 잘 작동하는 활성화함수가 있기도 합니다 (CNN + ReLU).



# Activation functions

Name	Plot	Equation	Derivative (with respect to $x$ )	Range	Order of continuity
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	$C^\infty$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	$C^{-1}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ <sup>[1]</sup>	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	$C^\infty$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	$C^\infty$
SQLN <sup>[10]</sup>		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$f'(x) = 1 \mp \frac{x}{2}$	$(-1, 1)$	$C^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$(-\frac{\pi}{2}, \frac{\pi}{2})$	$C^\infty$
ArSinH		$f(x) = \sinh^{-1}(x) = \ln(x + \sqrt{x^2 + 1})$	$f'(x) = \frac{1}{\sqrt{x^2 + 1}}$	$(-\infty, \infty)$	$C^\infty$
ElliotSig <sup>[11][12]</sup> Softsign <sup>[13][14]</sup>		$f(x) = \frac{x}{1 +  x }$	$f'(x) = \frac{1}{(1 +  x )^2}$	$(-1, 1)$	$C^1$
Inverse square root unit (ISRU) <sup>[15]</sup>		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left( \frac{1}{\sqrt{1 + \alpha x^2}} \right)^3$	$(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}})$	$C^\infty$
Inverse square root linear unit (ISRLU) <sup>[15]</sup>		$f(x) = \begin{cases} \frac{x}{\sqrt{1 + \alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \left( \frac{1}{\sqrt{1 + \alpha x^2}} \right)^3 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\frac{1}{\sqrt{\alpha}}, \infty)$	$C^2$
Rectified linear unit (ReLU) <sup>[16]</sup>		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	$[0, \infty)$	$C^0$
Bipolar rectified linear unit (BReLU) <sup>[17]</sup>		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} ReLU'(x_i) & \text{if } i \bmod 2 = 0 \\ ReLU'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$(-\infty, \infty)$	$C^0$

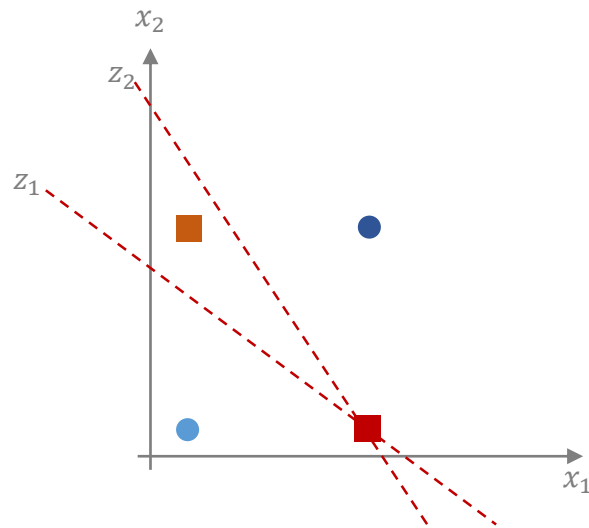
더 많은 활성화함수는 위키피디아를 참고하세요

[https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function)

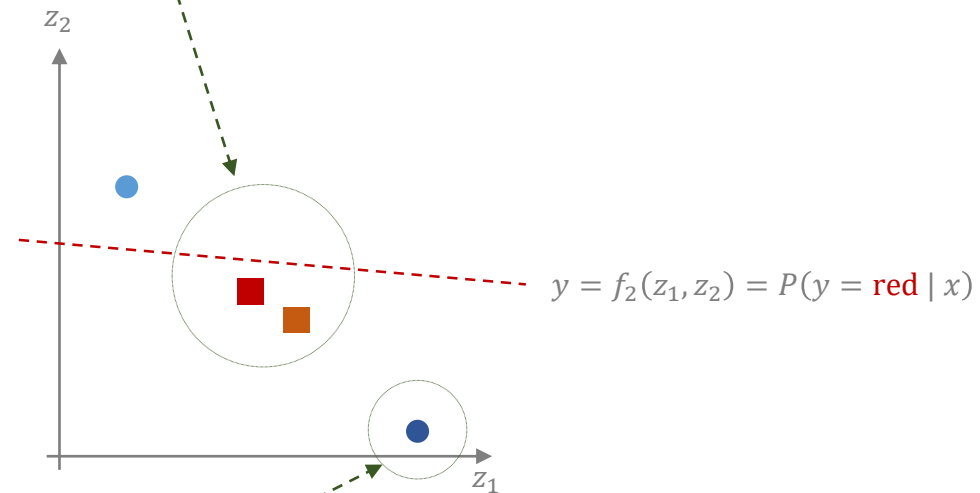
# Training

$$z = f_1(A_1x + b_1)$$

$$\hat{y} = f_2(A_2z + b_2) = f_2(A_2f_1(A_1x + b_1) + b_2)$$



점들이 'red' 일 확률이 확실하지 않습니다  
0.5 보다 조금 큼니다.



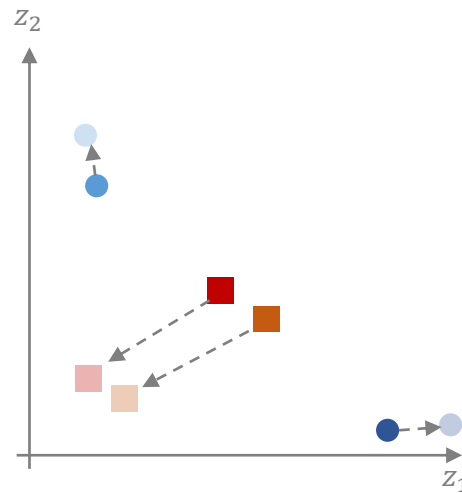
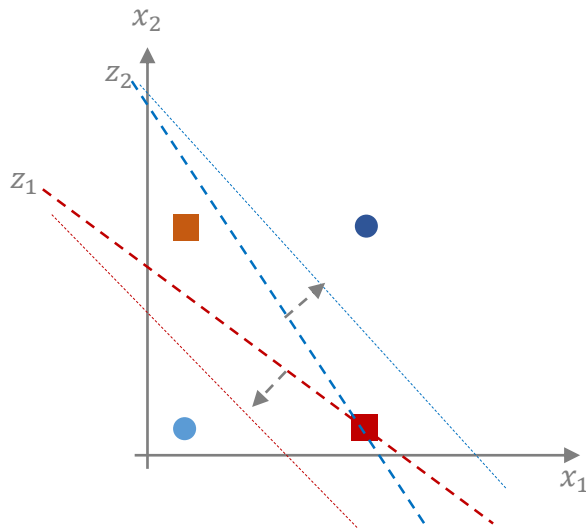
잘못 분류된 점이 존재합니다.

# Training

$$z = f_1(\mathbf{A}_1\mathbf{x} + \mathbf{b}_1)$$

$$\hat{y} = f_2(\mathbf{A}_2\mathbf{z} + \mathbf{b}_2) = f_2(\mathbf{A}_2f_1(\mathbf{A}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

각 점의  $z$  좌표가 수정되도록  $\mathbf{A}_1, \mathbf{b}_1$  을 업데이트 합니다.

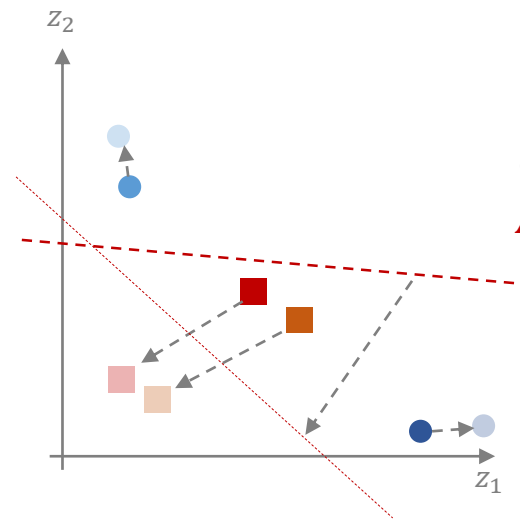
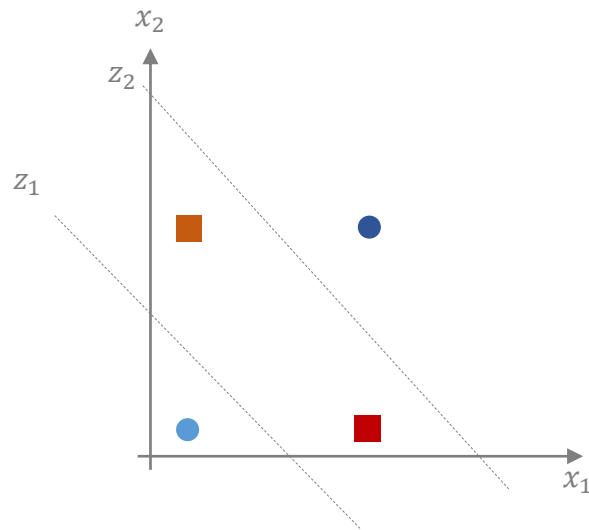


분류가 잘되도록 각 점의  $z$  좌표를 수정합니다  
(representation 을 수정합니다)

# Training

$$z = f_1(A_1x + b_1)$$

$$\hat{y} = f_2(\mathbf{A}_2z + \mathbf{b}_2) = f_2(A_2f_1(A_1x + b_1) + b_2)$$



업데이트 된  $z$  좌표를 이용하여 분류가 잘 되도록  $\mathbf{A}_2, \mathbf{b}_2$  를 업데이트 합니다.

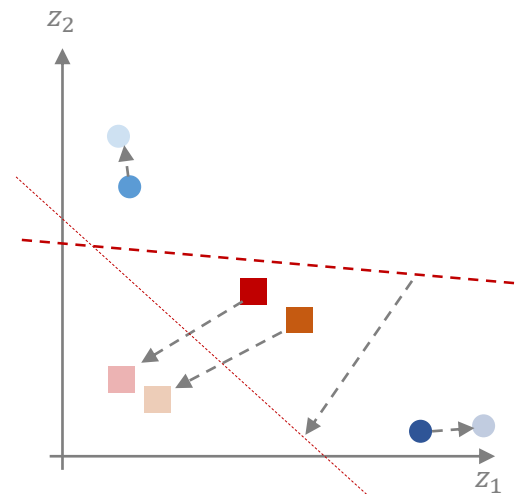
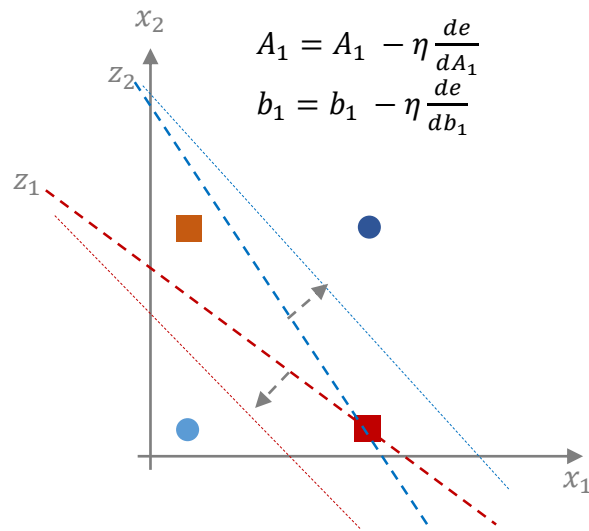
# Training

$$z = f_1(\mathbf{A}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\hat{y} = f_2(\mathbf{A}_2 \mathbf{z} + \mathbf{b}_2) = f_2(A_2 f_1(A_1 x + b_1) + b_2)$$

(e.g.) loss  $e = NLL(y, \hat{y})$

Loss 함수를 기준으로 품질이 더 좋아지는 방향으로 모든 패러미터를 동시에 학습합니다.



# Training

---

- 수학적으로 이해하려면,
  - Chain rule 을 이용하여 gradient 를 앞쪽 레이어의 패러미터로 전달합니다.

$$e = f(g(x)), \frac{de}{dg(x)} = f'(g(x))$$

$$\rightarrow e = f(h), \frac{de}{dh} = f'(h)$$

$$\rightarrow \frac{de}{dx} = \frac{de}{dg(x)} \times \frac{dg(x)}{x} = f'(g(x)) \times g'(x)$$

# Training

---

$$h_1 = f_1(A_1x + b_1)$$

$$\hat{y} = f_2(A_2h_1 + b_2) = f_2(A_2f_1(A_1x + b_1) + b_2)$$

$$e = NLL(y, \hat{y}) = -\log(\hat{y})$$

$$\frac{de}{dA_2} = \frac{de}{d\hat{y}} \times \frac{d\hat{y}}{dA_2} = -\frac{1}{\hat{y}} \times h_1 f_2'(A_2h_1 + b_2)$$

$$\frac{de}{dA_1} = \frac{de}{d\hat{y}} \times \frac{d\hat{y}}{dh_1} \times \frac{dh_1}{dA_1} = -\frac{1}{\hat{y}} \times f_2'(A_2h_1 + b_2)A_2 \times f_1'(A_1x + b_1)x$$

# Training

---

$$h_1 = f_1(A_1x + b_1)$$

$$\hat{y} = f_2(A_2h_1 + b_2) = f_2(A_2f_1(A_1x + b_1) + b_2)$$

$$e = NLL(y, \hat{y}) = -\log(\hat{y})$$

$$\frac{de}{dA_2} = \frac{de}{d\hat{y}} \times \frac{d\hat{y}}{dA_2} = -\frac{1}{\hat{y}} \times h_1 f_2'(A_2h_1 + b_2)$$

$$\frac{de}{dA_1} = \frac{de}{d\hat{y}} \times \frac{d\hat{y}}{dh_1} \times \frac{dh_1}{dA_1} = -\frac{1}{\hat{y}} \times \overbrace{f_2'(A_2h_1 + b_2)A_2}^{\text{활성함수의 미분값, 사용하는 활성화함수에 따라 다르게 정의}} \times f_1'(A_1x + b_1)x$$

활성함수의 미분값, 사용하는 활성화함수에 따라 다르게 정의



# Training

$$h_1 = f_1(A_1x + b_1)$$

$$\hat{y} = f_2(A_2h_1 + b_2) = f_2(A_2f_1(A_1x + b_1) + b_2)$$

$$e = NLL(y, \hat{y}) = -\log(\hat{y})$$

- $\frac{d\text{sigmoid}(x)}{dx} = f(x) \times (1 - f(x))$
- Sigmoid 와 같은 특정 활성화함수 미분값의 절대값 크기가 1 보다 작은 경우에는, 이 값의 누적값이 0 에 가까워질 수 있습니다.
- 활성화함수 미분값의 누적값이 지나치게 작아 앞에 위치한 레이어에 gradient 0 에 가까운 값이 전달되는 현상을 gradient vanishing problem 이라하며, 초기에 뉴럴넷 레이어를 깊게 쌓지 못한 원인이었습니다.

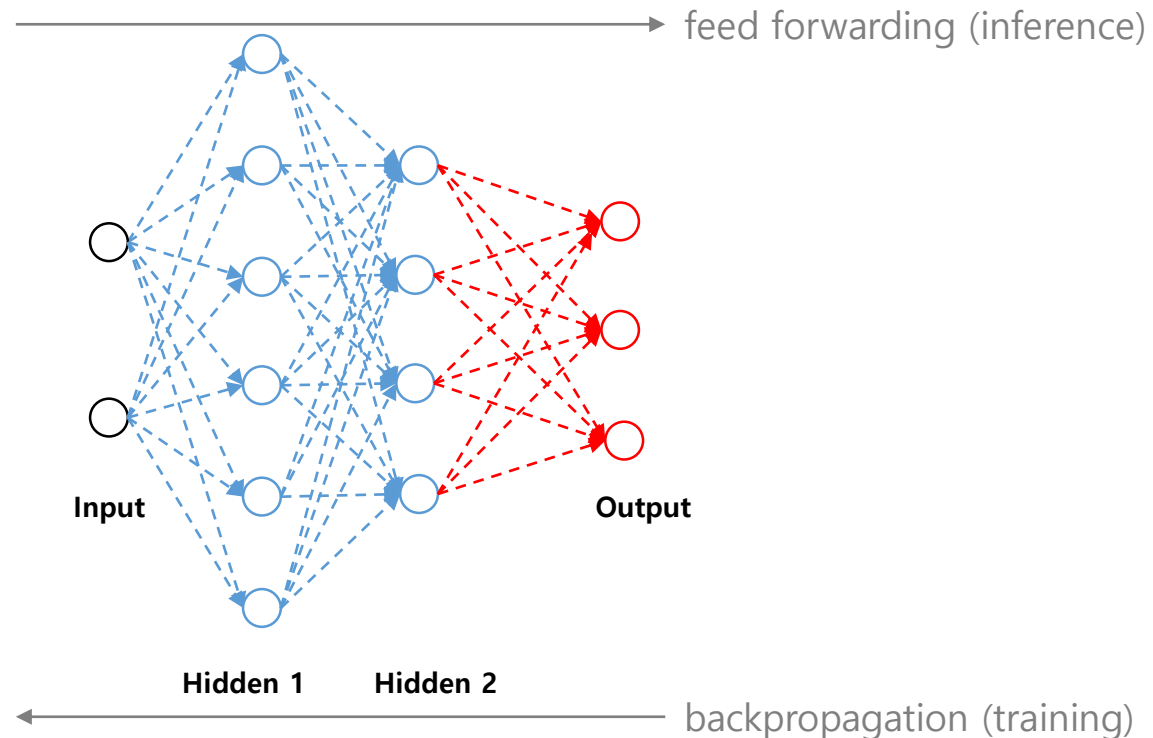
$$\frac{de}{dA_2} = \frac{de}{d\hat{y}} \times \frac{d\hat{y}}{dA_2} = -\frac{1}{\hat{y}} \times h_1 f'_2(A_2h_1 + b_2)$$

$$\frac{de}{dA_1} = \frac{de}{d\hat{y}} \times \frac{d\hat{y}}{dh_1} \times \frac{dh_1}{dA_1} = -\frac{1}{\hat{y}} \times \underbrace{f'_2(A_2h_1 + b_2)A_2}_{\text{활성함수의 미분값}} \times f'_1(A_1x + b_1)x$$

$$-\frac{1}{\hat{y}} \times \underbrace{f(A_2h_1 + b_2)}_{\leq 1} \underbrace{(1 - f(A_2h_1 + b_2))}_{\leq 1} A_2 \times \underbrace{f_1(A_1x + b_1)}_{\leq 1} \underbrace{(1 - f_1(A_1x + b_1))}_{\leq 1} x$$

# Training

- 네트워크를 통하여  $x \rightarrow h_1 \rightarrow h_2 \rightarrow \dots \rightarrow \hat{y}$  를 만드는 과정을 forwarding 이라 합니다.
- Loss 로부터 패러미터 변화량을 계산하는 과정을 backpropagation 이라 합니다.



# Training

---

- 수식으로 gradient 를 계산하면 복잡하지만, loss 로부터 모든 패러매터가 output 의 품질이 좋아지는 방향으로 동시에 학습된다는 점만 기억해도 됩니다.
- 앞서 gradient 를 계산했으니 logistic regression 을 학습한 것처럼 learning rate 를 곱하여 조금씩 패러매터를 개선합니다 (gradient descent)

# Optimizers

---

- Gradient descent 는
  - 패러매터  $\theta$  를
  - loss function 으로부터의 gradient 인  $-\nabla_{\theta}J(\theta)$
  - Learning rate  $\eta$  를 곱한 값을 더하여 업데이트 합니다.
- Optimizers 는  $\eta$  와  $\nabla_{\theta}J(\theta)$  를 다르게 정의하여 안정적이고 빠른 학습이 되도록 만듭니다.

$$\theta = \theta - \eta \nabla_{\theta}J(\theta)$$

# Optimizers

---

- Batch style 은 모든 데이터의 gradient 의 평균을 취하여 학습합니다.
  - 데이터의 개수가 적을 때 안정적이지만, 데이터의 개수가 많으면 매우 오래걸립니다.

$$\theta = \theta - \eta \nabla_{\theta} J(\theta, X, Y)$$

$$\nabla_{\theta} J(\theta, X, Y) = \frac{1}{n} \sum \nabla_{\theta} J(\theta, x^{(i)}, y^{(i)})$$

# Optimizers

---

- Stochastic Gradient Descent (SGD) 은 각 데이터 별로 업데이트를 합니다.
  - 데이터의 개수가 많고, 패턴이 반복될 때 잘 작동합니다.
  - 반복되는 데이터가 존재한다면 모든 데이터의 gradients 의 평균이 각 데이터의 gradient 와 비슷할 가능성이 높기 때문입니다.
  - 대신 (노이즈) 각 데이터에 민감하지 않도록 learning rate 는 작게 설정합니다.

$$\theta = \theta - \eta \nabla_{\theta} J(\theta, x^{(i)}, y^{(i)})$$

# Optimizers

---

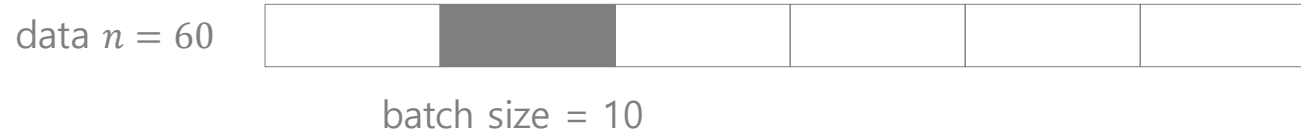
- Mini-batch 는 SGD 와 batch 의 중간입니다.
  - 적은 양의 데이터로 여러 번 패러매터를 학습함으로써 빠르게 모델을 학습시키고,
  - SGD 보다 일부 데이터에 덜 민감합니다.

$$\theta = \theta - \eta \nabla_{\theta} J(\theta, x^{(i:i+b)}, y^{(i:i+b)})$$

# Optimizers

---

- Mini-batch 는 SGD 와 batch 의 중간입니다.



**epoch** 은 모든 데이터가 모델 학습에 이용되는 단위입니다.

**iteration** 은 패러매터가 학습되는 (mini batch 가 이용되는) 횟수입니다.

**batch size** 는 한 iteration 에 이용되는 데이터의 개수입니다.

$$\theta = \theta - \eta \nabla_{\theta} J(\theta, x^{(i:i+b)}, y^{(i:i+b)})$$



# Optimizers

---

- Mini-batch 는 shuffling 이 필요합니다.
  - 데이터의 순서에 따라 특정한 패턴 변화가 있을 수 있습니다. 데이터의 순서를 임의로 섞은 뒤, batch size 만큼 선택하여 이러한 위험성을 줄입니다.

data  $n = 60$

1	4	6	2	3	5
---	---	---	---	---	---

batch size = 10

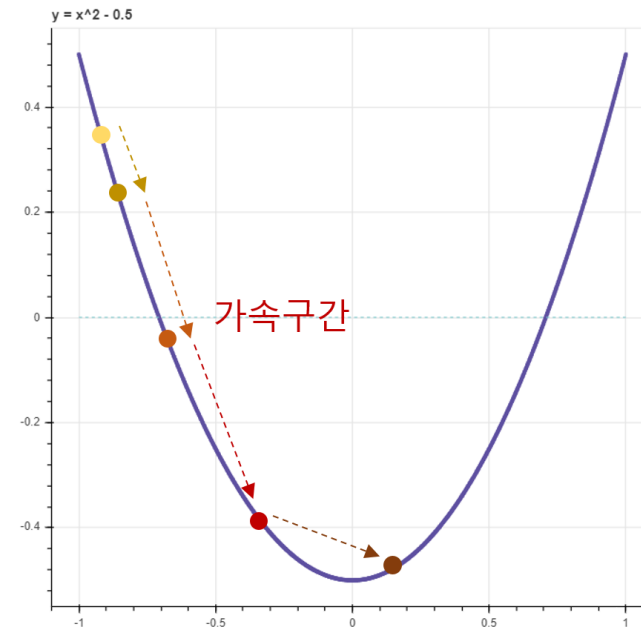
$$\theta = \theta - \eta \nabla_{\theta} J(\theta, x^{(i:i+b)}, y^{(i:i+b)})$$

# Optimizers

- Batch, SGD, mini-batch 모두 한 번에 작은 크기의 gradient 가 계산됩니다.  
효율적인 학습을 위하여 momentum 을 이용할 수 있습니다.
- 그러나 momentum 은 loss 의 최저점을 지나칠 가능성이 있습니다.

$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

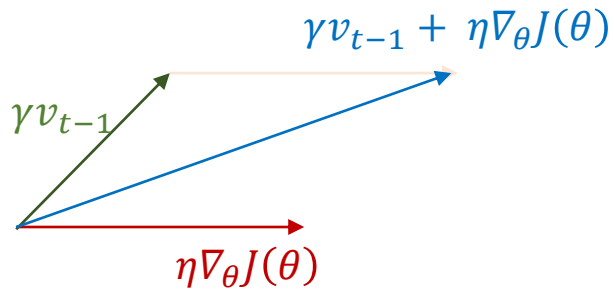


# Optimizers

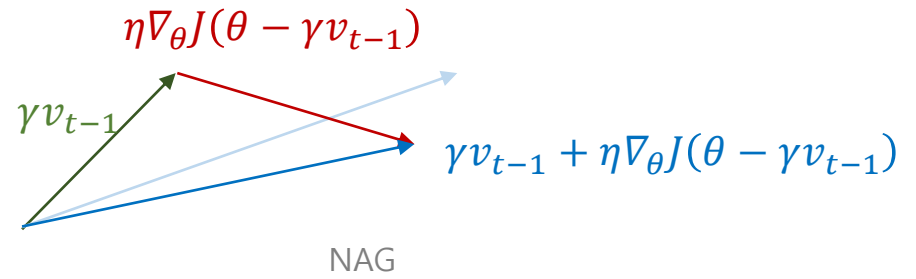
- Nesterov Accelerated Gradient (NAG) 는 momentum 에 의하여 최저점을 지나치는 경우를 방지하기 위한 optimizer 입니다.
  - Momentum 만큼 패러매터를 미리 이동한 뒤, gradient 를 계산합니다.

$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$



gradient with momentum



# Optimizers

---

- Adaptive Gradient (Adagrad) 는 매 변수마다 골고루 업데이트를 합니다.
  - 초기화 값에 따라 hidden weights 의 일부만 학습될 수 있습니다.
  - 시점별로 각 weights 의 변화량  $(\nabla_{\theta}J(\theta_t))^2$  을 누적하여 저장한 뒤, 변화량이 적은 부분에 더 큰 learning rate 를 부여합니다.
  - $\epsilon \cong 10^{-4}$  은 분모가 0 인 경우를 방지합니다.

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_{\theta}J(\theta)$$

$$G_t = G_{t-1} + (\nabla_{\theta}J(\theta_t))^2$$

# Optimizers

---

- RMSprop 는 AdaGrad 의  $G_t$  가 무한정 커지는 것을 방지합니다.
  - $0 < \gamma < 1$  에 의하여 이전  $G_{t-1}$  의 크기를 감소함으로써,  $G_t$  가 지속적으로 증가하는 것을 방지하고, 최근에 업데이트가 적은 부분에 집중하여 학습이 이뤄지도록 합니다.

$$\theta = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_{\theta} J(\theta)$$

$$G_t = \gamma G_{t-1} + (1 - \gamma) (\nabla_{\theta} J(\theta_t))^2$$

# Optimizers

---

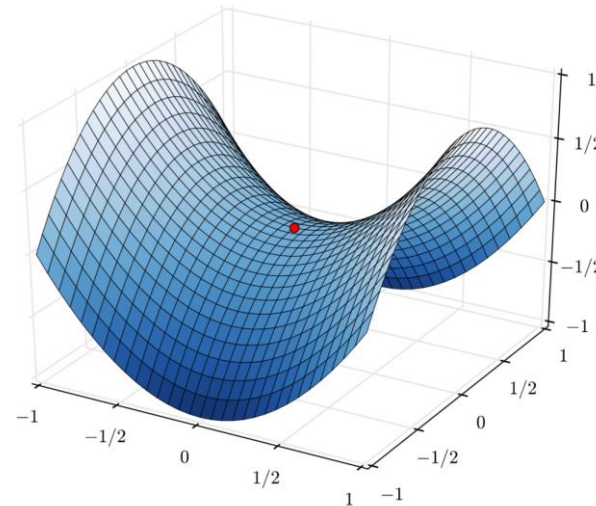
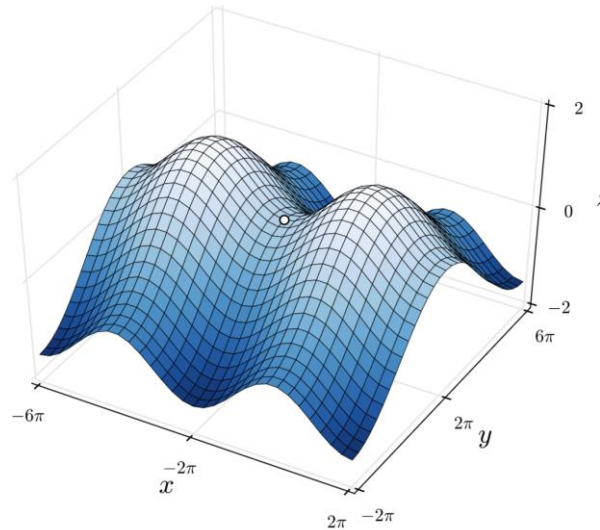
- Adam 은 RMSProp 와 momentum 을 모두 이용합니다.
  - $m_t$  를 이용하여 업데이트를 가속화하며,  $v_t$  를 이용하여 업데이트가 느린 weights 에 더 큰 learning rate 를 부여합니다.

$$\theta = \theta - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} J(\theta))^2$$

- SGD 는 saddle points\* 에서 멈춰버릴 가능성이 다른 optimizers 보다 상대적으로 높다고 (하지만 매우 작은 확률로 발생) 알려져 있습니다.



[image from Wikipedia] Saddle points (안장점) 은 한 방향에서는 최소점이며 다른 방향에서는 최대점인 지점으로, 미분값이 0 이기 때문에 gradient 가 발생하지 않아 업데이트가 되지 않을 가능성이 있다.

# Optimizers

---

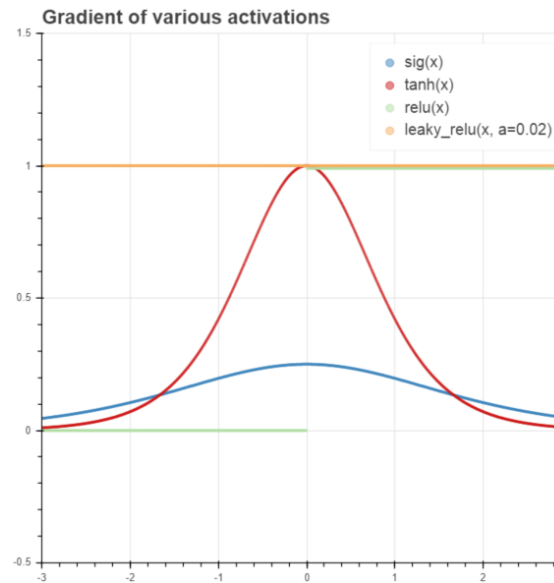
- Which one is best?
  - 다양한 optimizers 들이 있지만, 최근에도 SGD 를 이용하는 논문들이 지속적으로 등장하고 있습니다. 수렴까지 오래걸리지만, 하드웨어 성능으로 이를 극복합니다.
  - 데이터가 sparse 하면 features 마다 다른 속도로 학습되는 RMSprop, 학습이 빠르게 수렴하기를 원한다면 대체로 adaptive 한 방법들이 좋다고 알려져 있습니다.
- 그 외에도 다양한 optimizers 가 제안되었습니다. 더 자세히 공부하실 때는 꼭 ruder<sup>^1</sup> 의 블로그를 읽어보시기 바랍니다.

<sup>^1</sup>: <https://ruder.io/optimizing-gradient-descent/>



# Initializers

- Weights 를 0 에 가까운 임의의 숫자로 초기화 할 수 있습니다.
  - Sigmoid, tanh 과 같은 활성화함수는 0 근처에서 큰 gradient 가 발생합니다.
  - Weights 와 input 의 내적값 ( $A_j h_{j-1} + b_j$ ) 이 크면 학습이 잘 이뤄지지 않습니다.



# Initializers

---

- Xavier (Glorot) initializer 는 매 hidden layers 에서의 값들의 분포를 고르게 만드는 것이 목적입니다.
  - Hidden space 에서 한 영역에만 벡터값이 몰려있다면 학습이 잘 이뤄지지 않습니다.
  - Hidden layers 의 input, output 크기를 고려하여 weights 를 초기화 합니다.
  - Sigmoid, tanh 를 이용할 경우 효과적이라 알려져 있습니다.
- $W \sim N\left(0, \sqrt{\frac{2}{n_{in}+n_{out}}}\right)$  or  $W \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}}\right)$

# Initializers

---

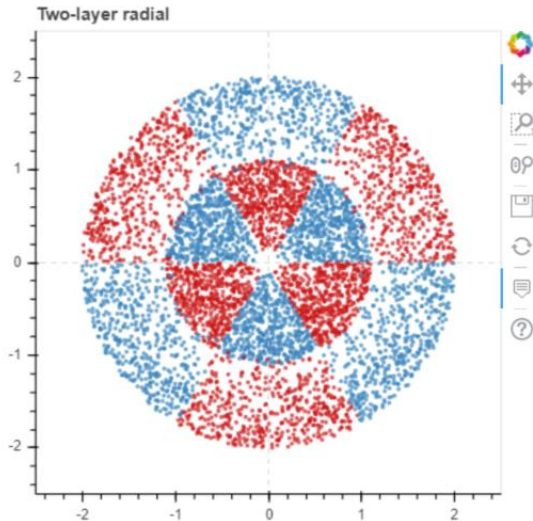
- He initializer 는 ReLU 를 이용하기 위한 초기화 방법입니다.
  - Xavier + ReLU 를 이용하면 학습이 비효율적이기 때문에 이를 보완하기 위해 더 큰 분산으로 random numbers 를 생성합니다.
- $W \sim N\left(0, \sqrt{\frac{2}{n_{in}}}\right)$  or  $W \sim U\left(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}}\right)$

- 
- 최근의 뉴럴네트워크 패키지들은 다양한 activation functions, optimizers, initializers 를 제공합니다. 각각의 원리까지 자세히 알아야 하는 것은 아니지만, 제안 배경과 목적을 알수록 효율적인 모델 학습이 가능합니다.

## Demo: two layers radial dataset, $h=(6,6)$

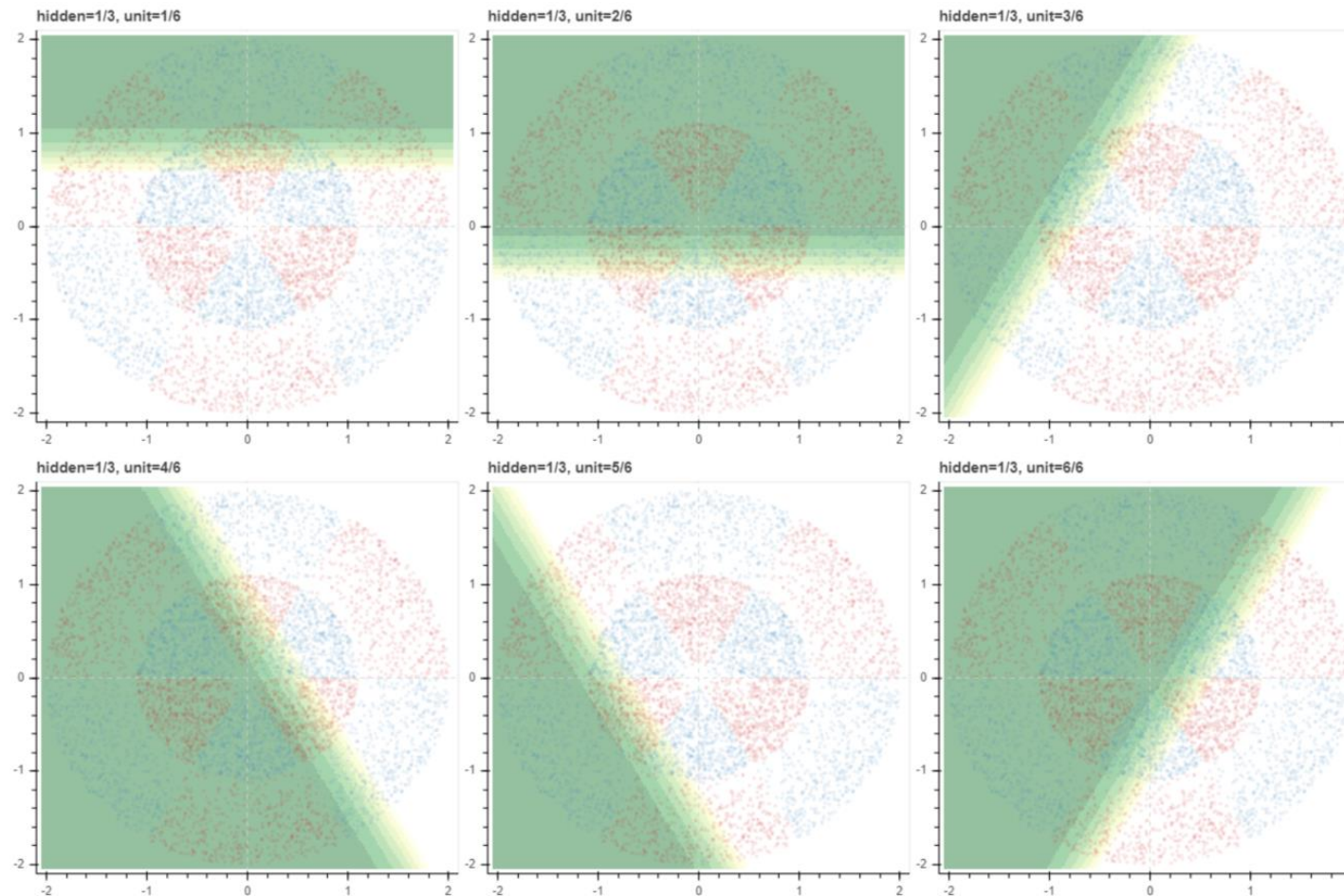
---

- 2 개의 클래스로 구성된 2 층의 방사형 데이터입니다.
- 이를 구분하기 위한 2 layers feed forward 모델을 학습해 봅니다.



## Demo: two layers radial dataset, $h=(6,6)$

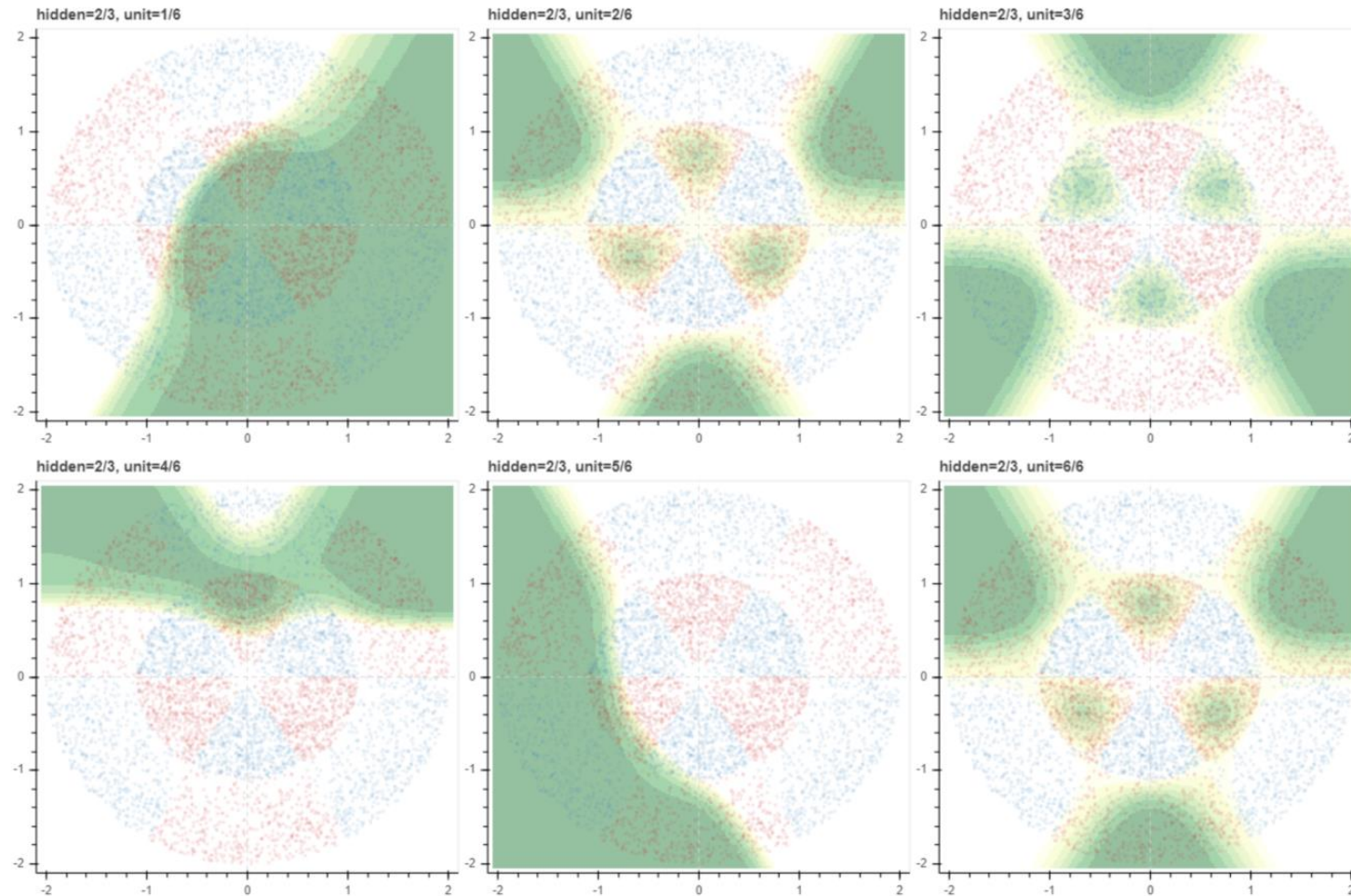
- 층을 나누기 위한 방식으로 첫번째 hidden layer 를 학습합니다.



1st hidden, 6 units

## Demo: two layers radial dataset, $h=(6,6)$

- 두 개의 hidden layers + tanh 만으로도 비선형의 공간을 학습할 수 있습니다.

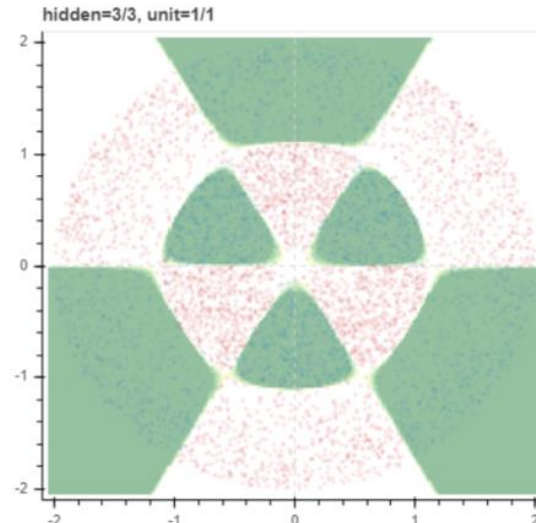
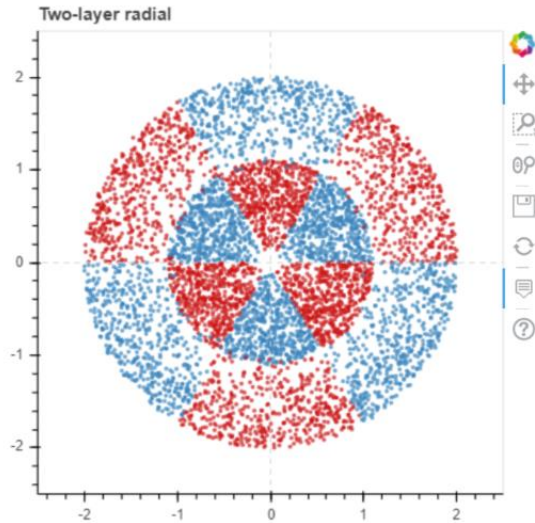


2nd hidden, 6 units



## Demo: two layers radial dataset, $h=(6,6)$

- 방사형으로 퍼지는 데이터라 hidden units 만 충분하면 쉽게 클래스를 구분할 수 있습니다.

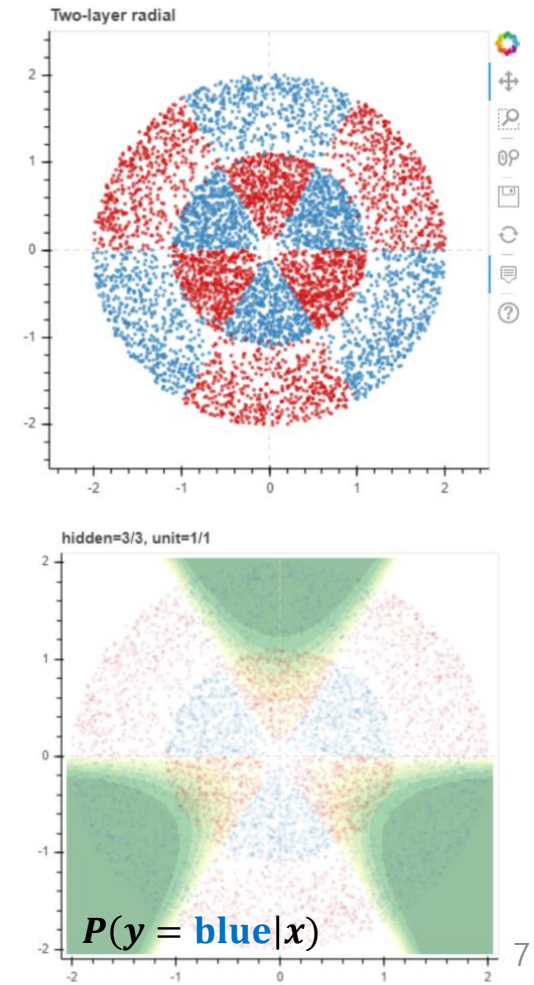
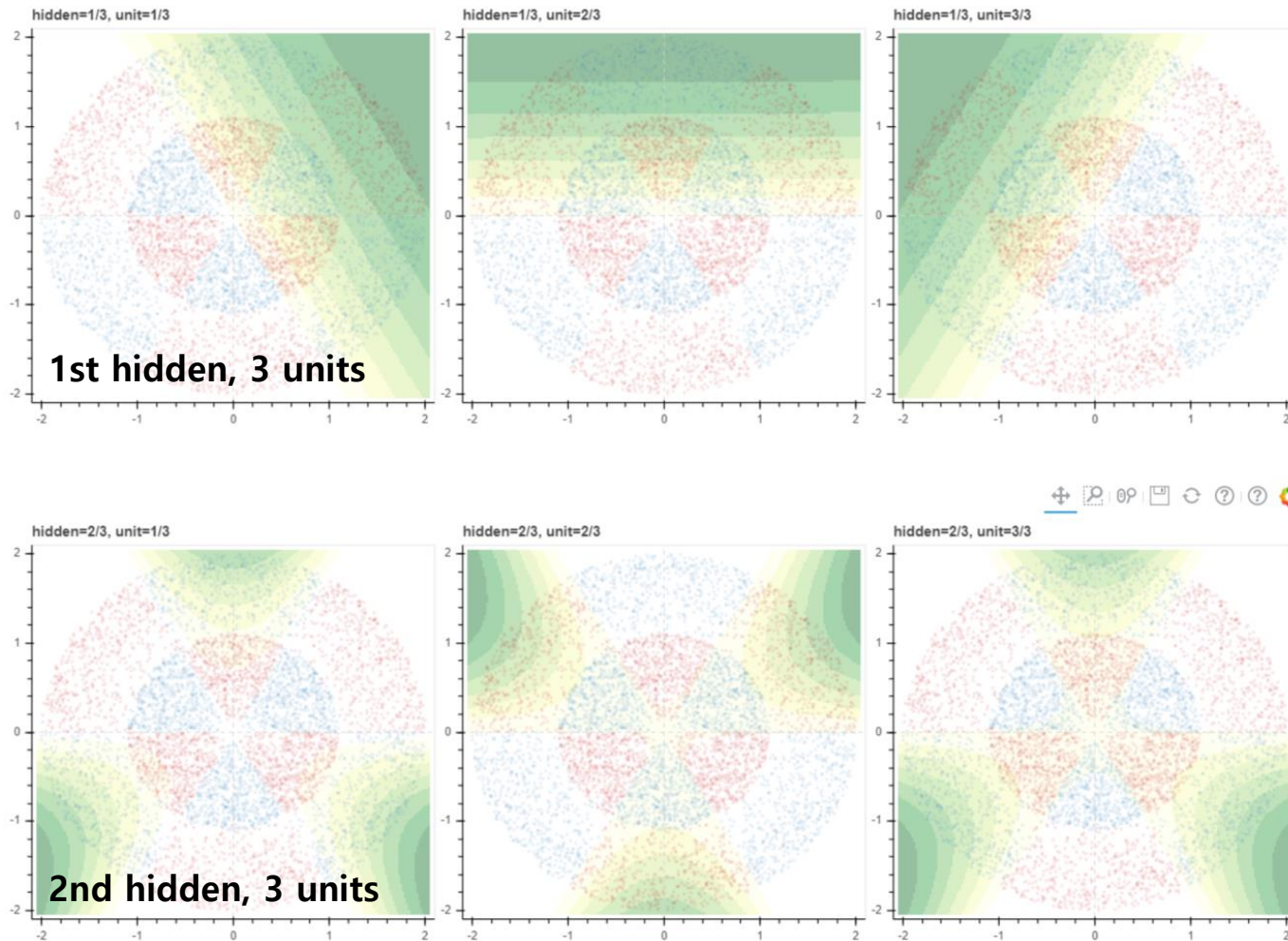


$$P(y = \text{blue} | x)$$



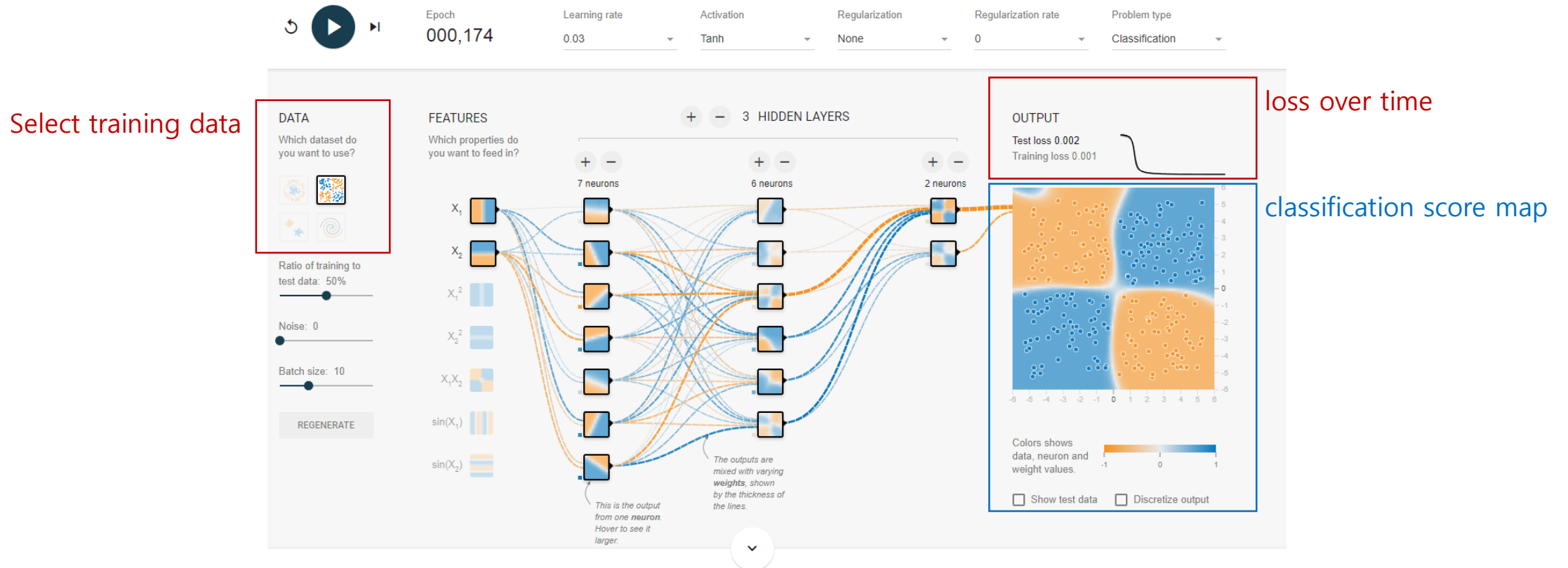
# Demo: two layers radial dataset, $h=(3,3)$

- Hidden units 이 부족하면 복잡한 공간을 제대로 이해하지 못합니다.



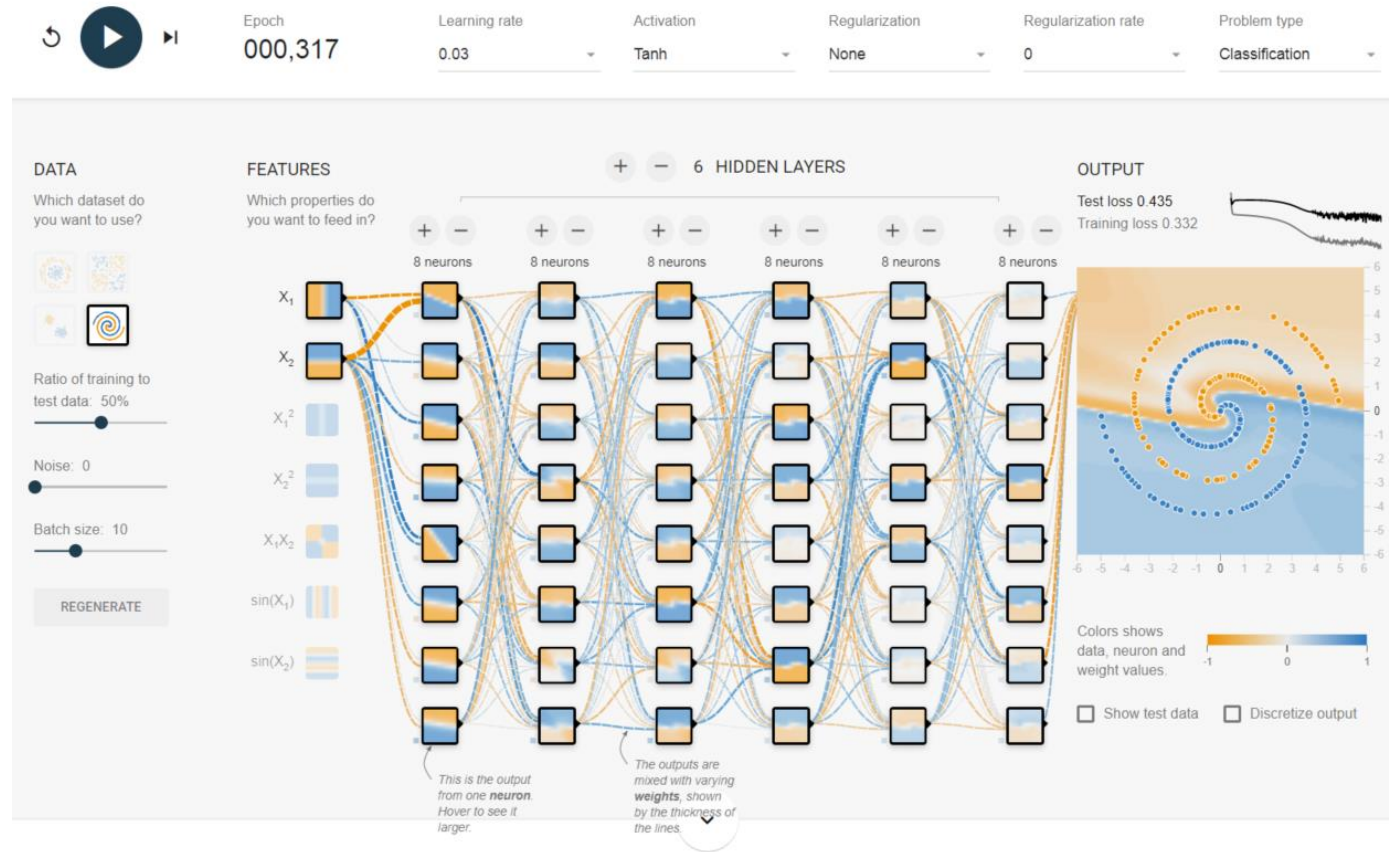
# Demo

- <http://playground.tensorflow.org/> 에서 neural network 의 hidden layer 에 의해서 데이터 공간이 어떻게 변하는지 살펴볼 수 있습니다.



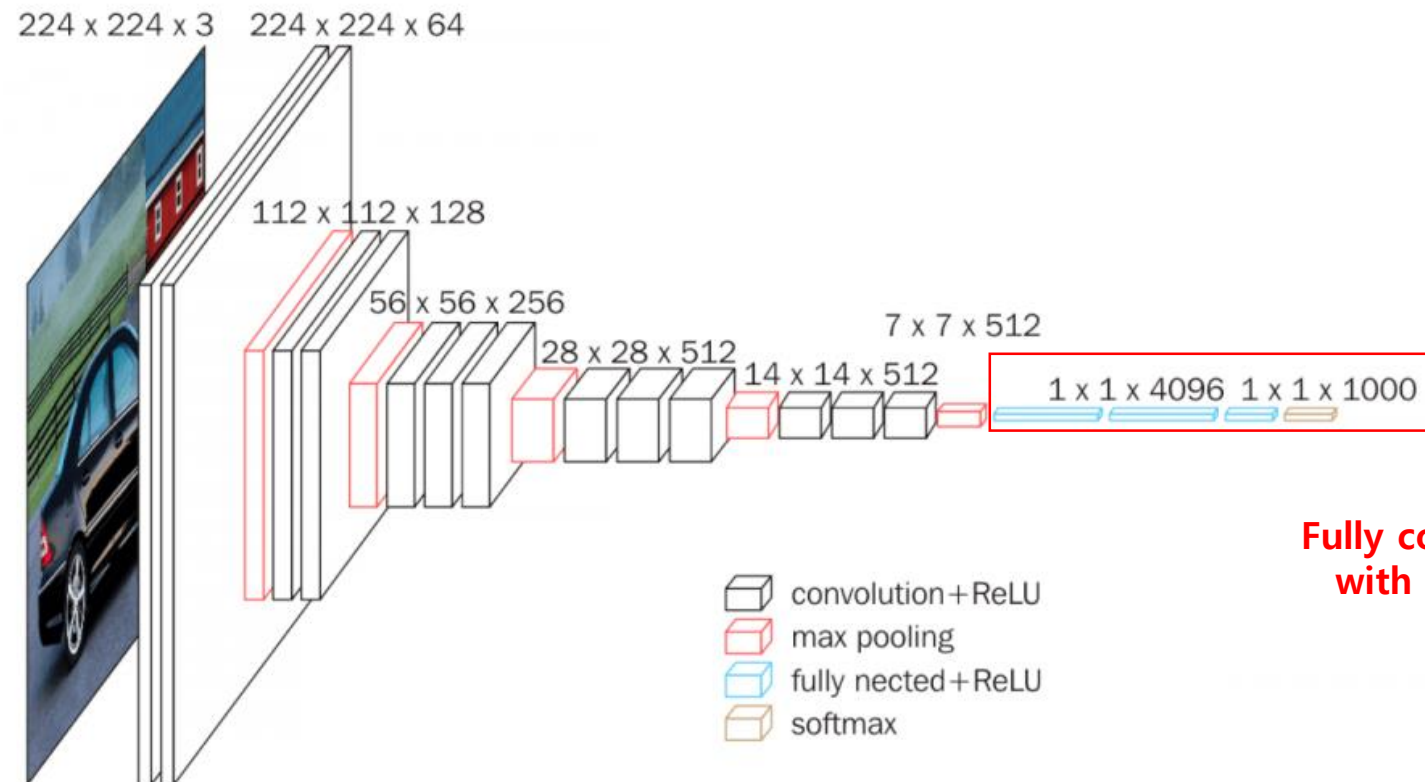
# Demo

- 뉴럴 네트워크에 적합한 데이터 + 세팅이 있습니다.
- 방향을 기준으로 representation 을 학습하므로 spiral 데이터는 학습하기 어렵습니다.



# Feed-forward Neural Network

- VGG16 같은 딥러닝 모델도 마지막 layer 는 Softmax regression 입니다. 이미지처럼 복잡한 데이터를 linear separable 한 representation 으로 변형합니다.



**Fully connected network  
with Softmax output**

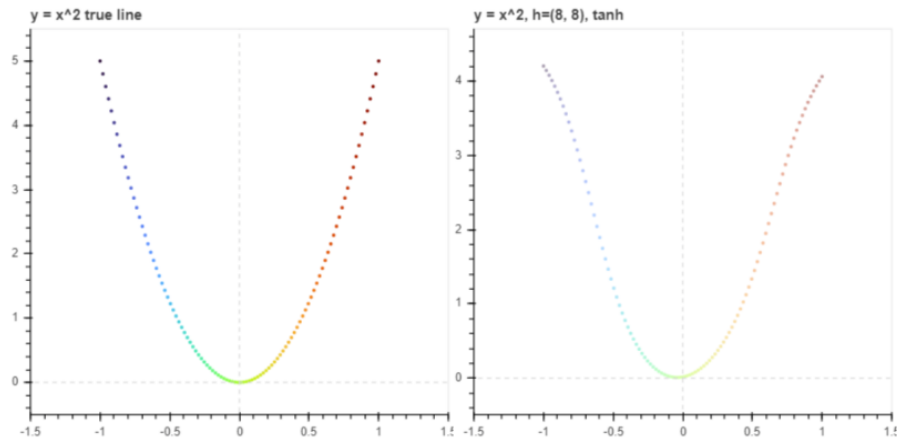
# Feed-forward for Regression

---

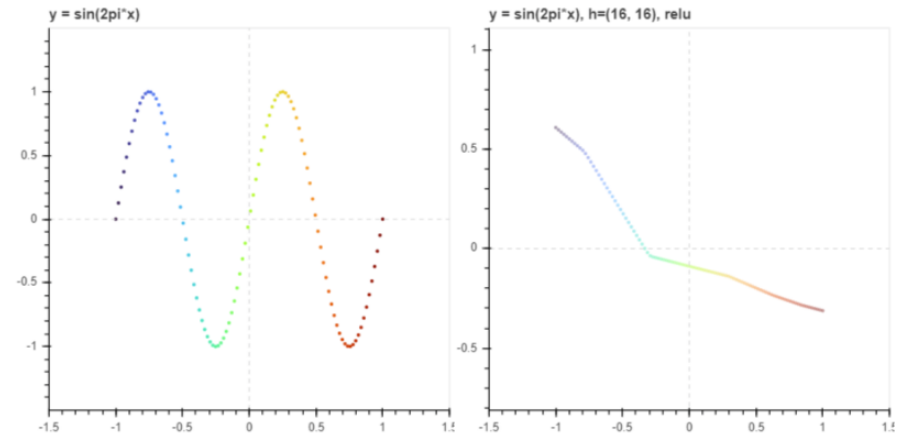
- 분류 문제를 위해서 마지막 hidden output 를 이용하여 softmax 를 수행합니다.
  - $\hat{y} = f_l(f_{l-1}(\dots f_1(x))) = f_l(h_{l-1})$
- 회귀 문제는  $h_{l-1}$  를 input 으로 선형회귀모델을 이용할 수 있습니다.
  - $\hat{y} = f_{l-1}(\dots f_1(x))^T \beta = (h_{l-1})^T \beta$
  - 분류 문제는 각 클래스의 데이터들이 비슷한 공간에 모여있도록  $h_{l-1}$  를 학습하면 되지만, 회귀 문제는  $h_{l-1}$  와  $\beta$  간의 선형 관계까지 학습해야 합니다.

# Feed-forward for Regression

- 분류 문제는 각 클래스의 데이터들이 비슷한 공간에 모여있도록  $h_{l-1}$  를 학습하면 되지만, 회귀 문제는  $h_{l-1}$  와  $\beta$  간의 선형 관계까지 학습해야 합니다.



$n = 100$ ,  $y = x^2$  데이터에 대한  
 $h=(8,8)$  feed forward network 학습 결과

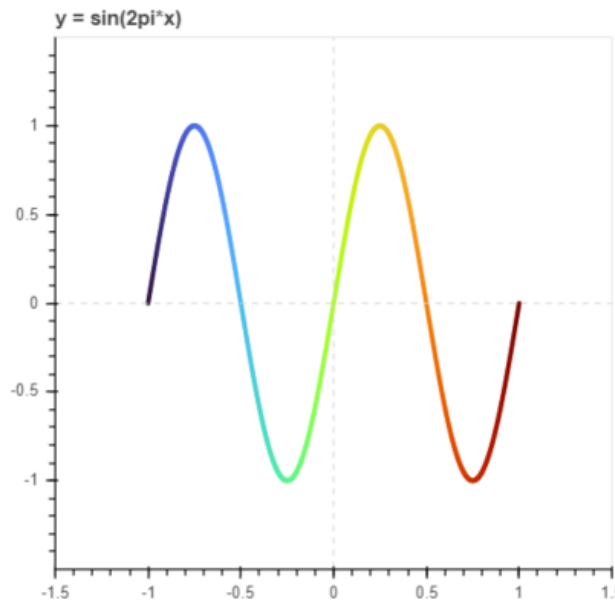


$n = 100$ ,  $y = \sin(2\pi x)$  데이터에 대한  
 $h=(16,16)$  feed forward network 학습 결과

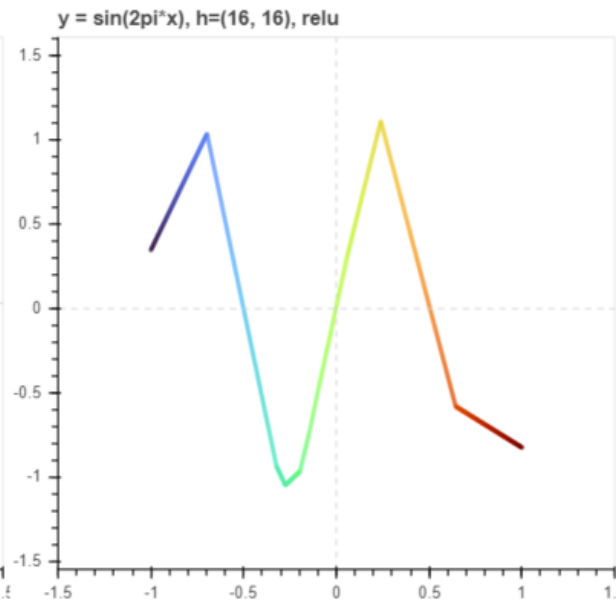


# Feed-forward for Regression

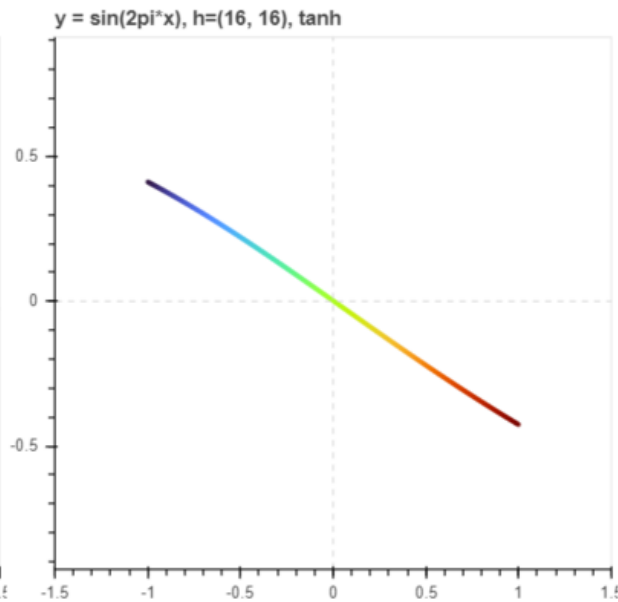
- 대체로 데이터가 많을수록 (패턴이 반복될수록) 학습이 잘 이뤄집니다.
- 활성화함수, 초기화 결과에 따라 학습이 잘 이뤄지지 않기도 합니다.



$n = 1000, y = \sin(2\pi x)$  데이터



$h = (16, 16) + \text{ReLU}$  학습 결과



$h = (16, 16) + \tanh$  학습 결과