

생성 모델과 시각 지능

Generative Model and Visual Intelligence

02 주차 |

딥러닝 기초

상명대학교 컴퓨터과학과
민 경 하

학습목차

1. 세상 단순한 인공 신경망
2. 세상 단순한 인공 신경망의 구현 (이론)
3. 세상 단순한 인공 신경망의 구현 (Numpy)
4. 세상 단순한 인공 신경망의 구현 (PyTorch)
5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)



1. 세상 단순한 인공 신경망

1. 세상 단순한 인공 신경망

세상에서 가장 단순한 분류 문제

개와 늑대를 구분하는 분류 문제



VS



세상에서 개와 늑대를
가장 잘 구분하는 존재는?

사람

사람이 어떻게
분류하는지 물어보면?

1. 세상 단순한 인공 신경망

세상에서 가장 단순한 분류 문제

개와 늑대를 구분하는 분류 문제



VS



● 사람이 개와 늑대를 구분하는 방법은?

귀

눈

혀

꼬리

.....

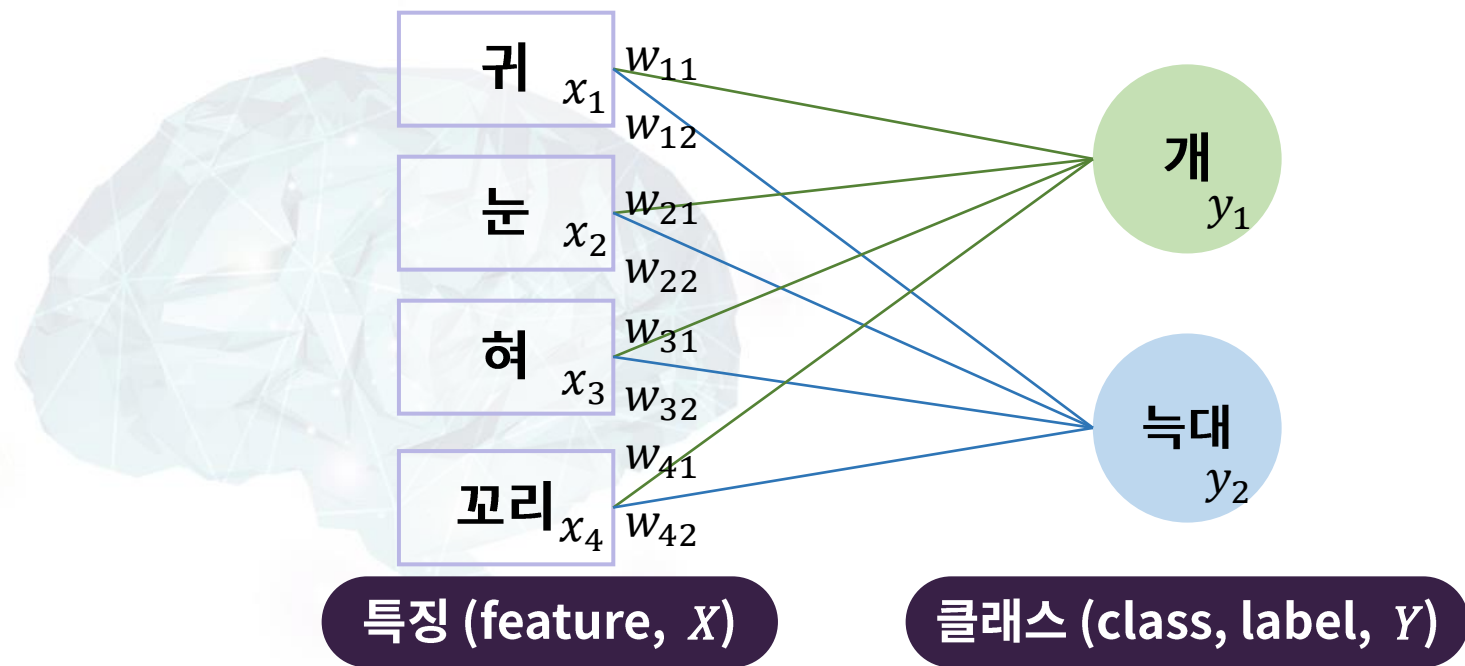
귀여운가?

1. 세상 단순한 인공 신경망

분류 문제 → 인공 신경망 (Neural network)

$$w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 = y_1$$

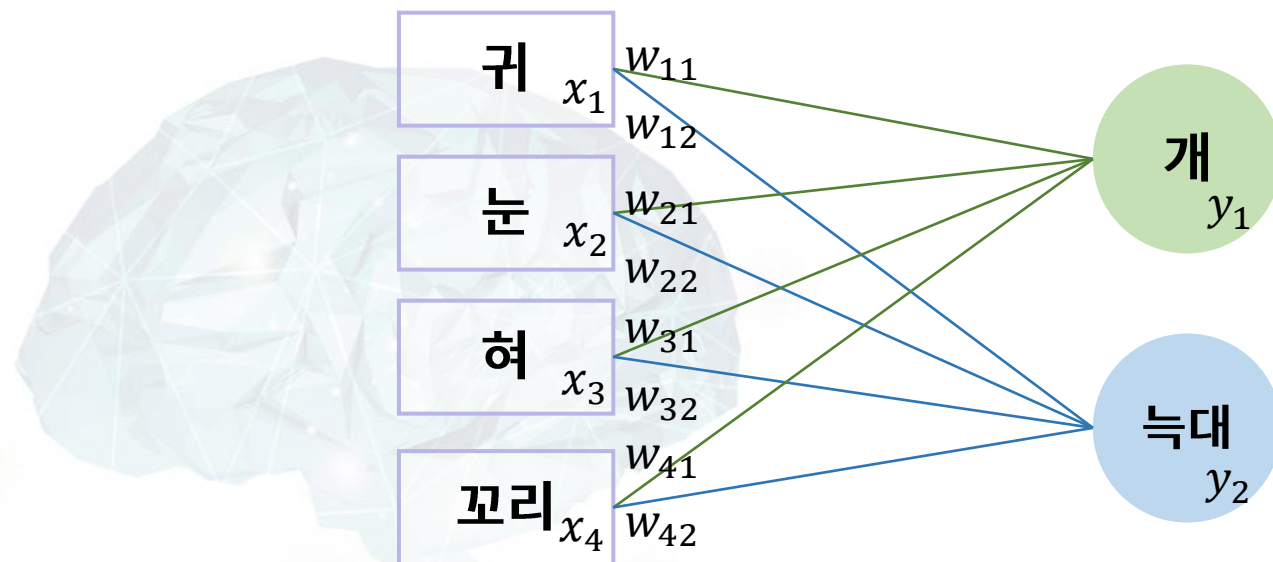
$$w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 = y_2$$



1. 세상 단순한 인공지능

분류 문제 → 인공지능 (Neural network)

$$\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad WX = Y$$

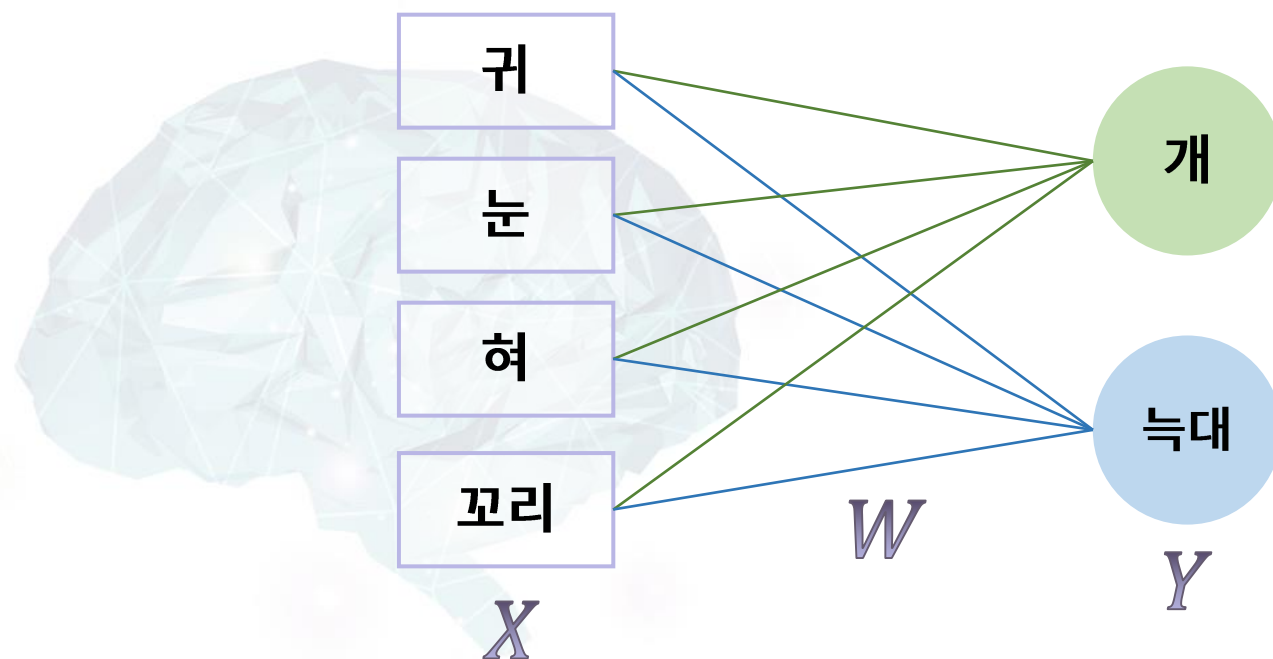


특징 (feature, X)

클래스 (class, label, Y)

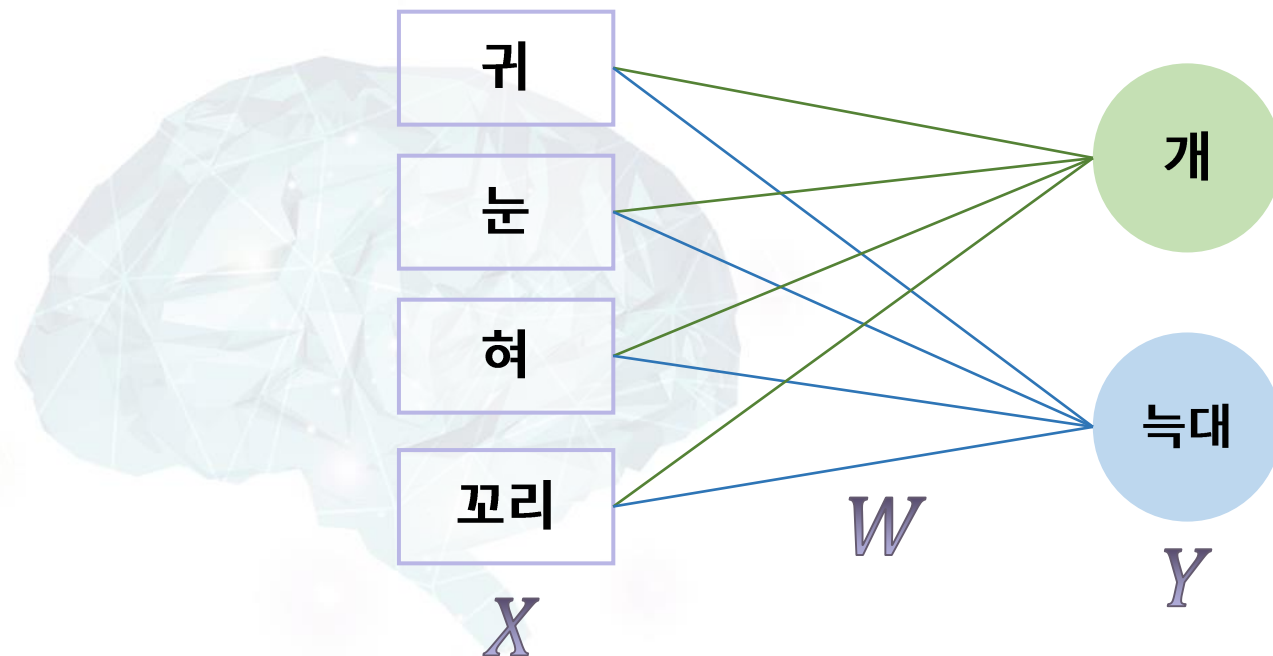
분류 문제 → 인공 신경망 (Neural network)

- 기존의 분류 방법 (Explicit method)
 - W를 결정하는 것이 가장 어려운 문제
 - 각 특징의 역할을 결정해야 함 → algorithm 설계



분류 문제 → 인공 신경망 (Neural network)

- 기존의 기계 학습 (Semi-implicit method)
 - ✓ 학습을 통해서 W 를 결정함 (Implicit)
 - ✓ 어떤 요소가 X 에 포함되는지 결정해야함 (Explicit)

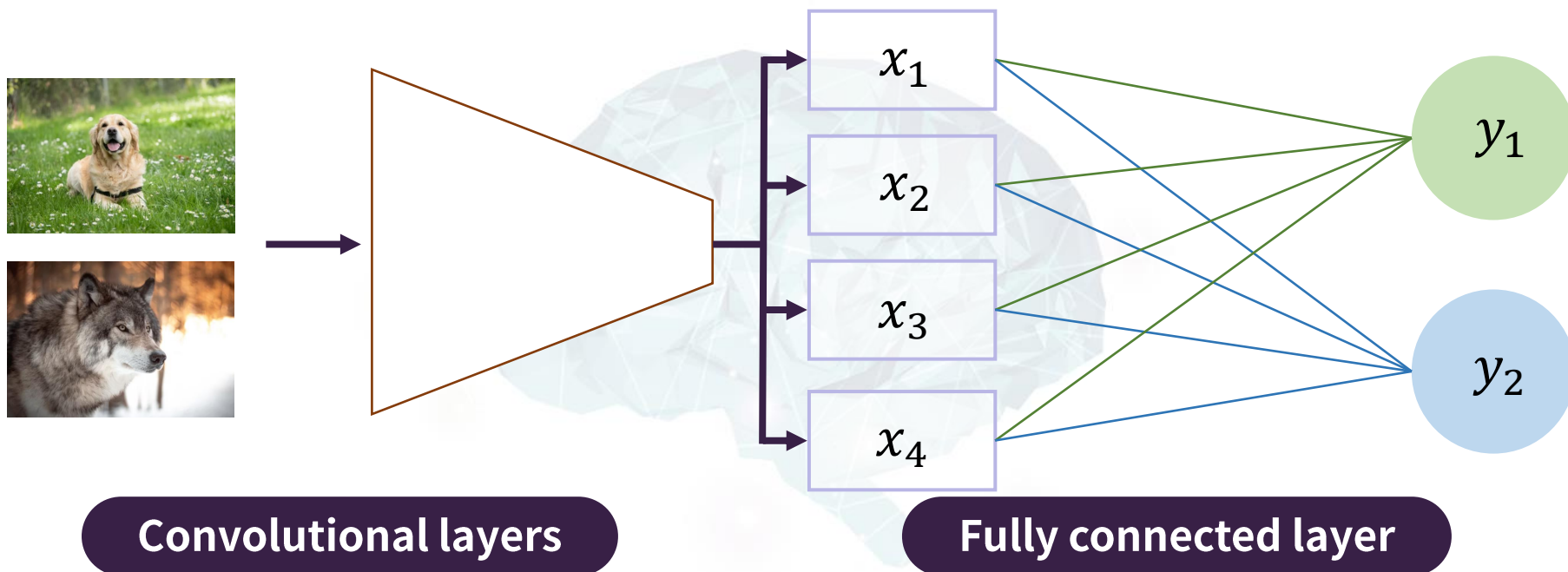


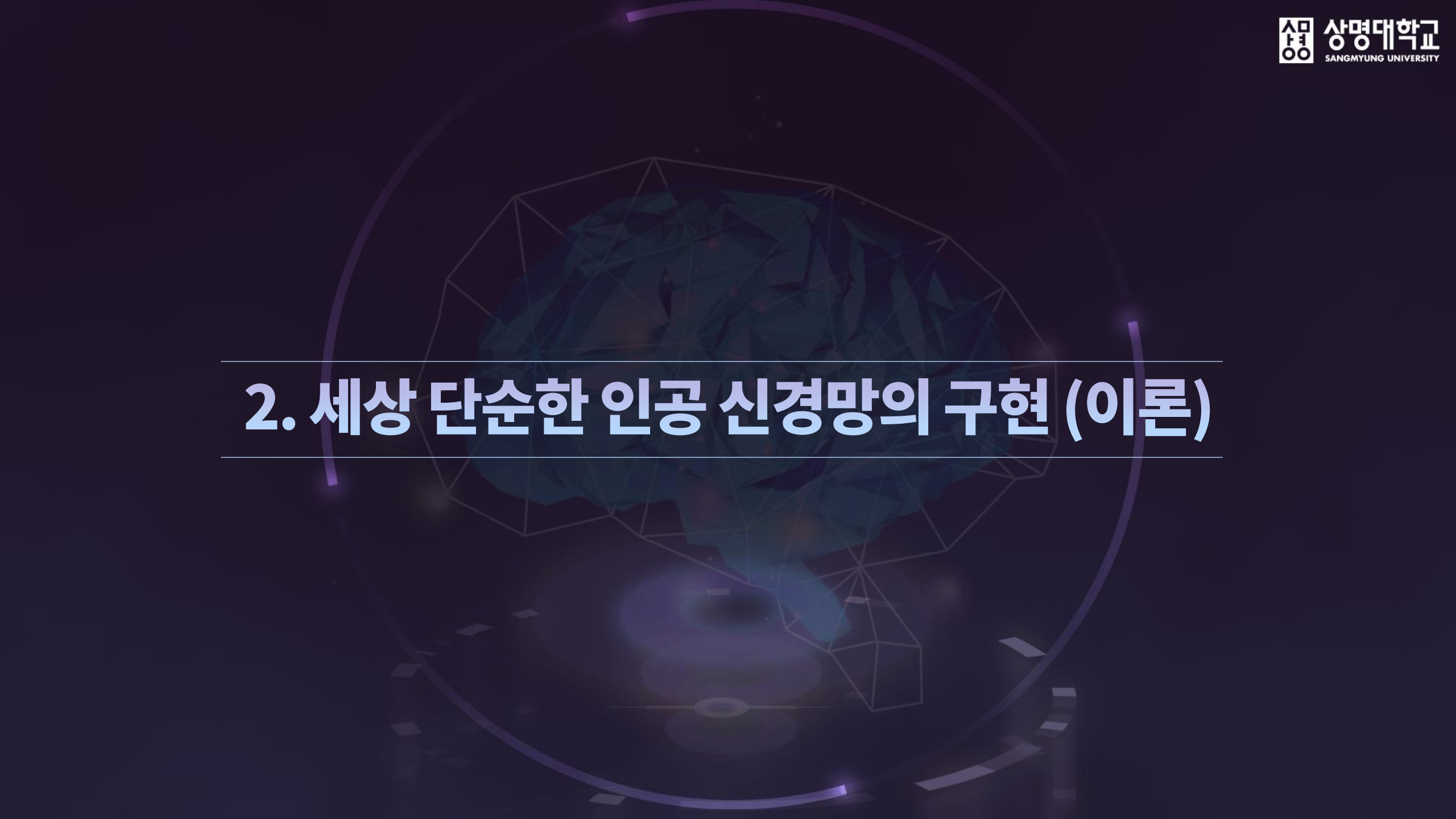
1. 세상 단순한 인공 신경망

분류 문제 → 인공 신경망 (Neural network)

● 심층 학습 (Implicit method)

- ✓ 학습을 통해서 W 를 결정함
- ✓ 어떤 요소가 X 에 포함되는지 학습을 통해서 결정함





2. 세상 단순한 인공 신경망의 구현 (이론)

2. 세상 단순한 인공 신경망의 구현 (이론)

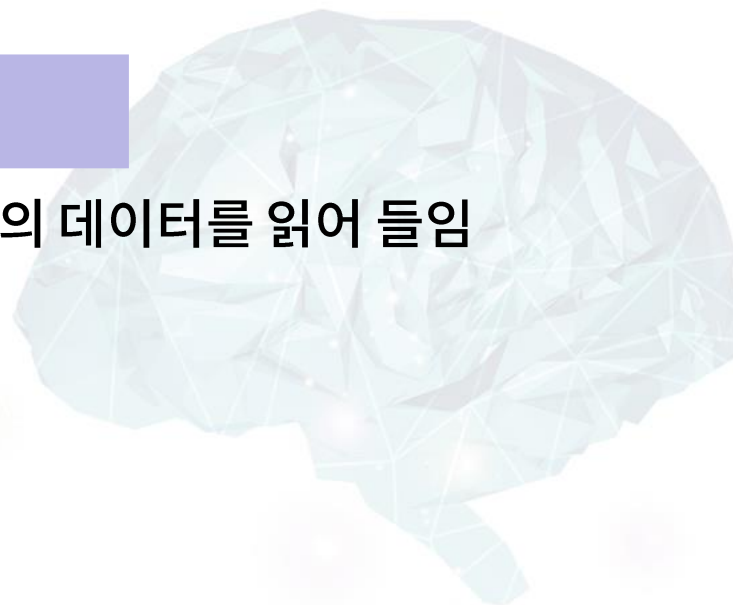
인공 신경망의 구현의 4단계

인공 신경망 모델 정의

- ✓ 인공 신경망의 구조를 모델 표현
- ✓ 다양한 라이브러리를 이용한 모델 정의

데이터 로딩

- ✓ 미리 저장된 대용량의 데이터를 읽어 들임



2. 세상 단순한 인공 신경망의 구현 (이론)

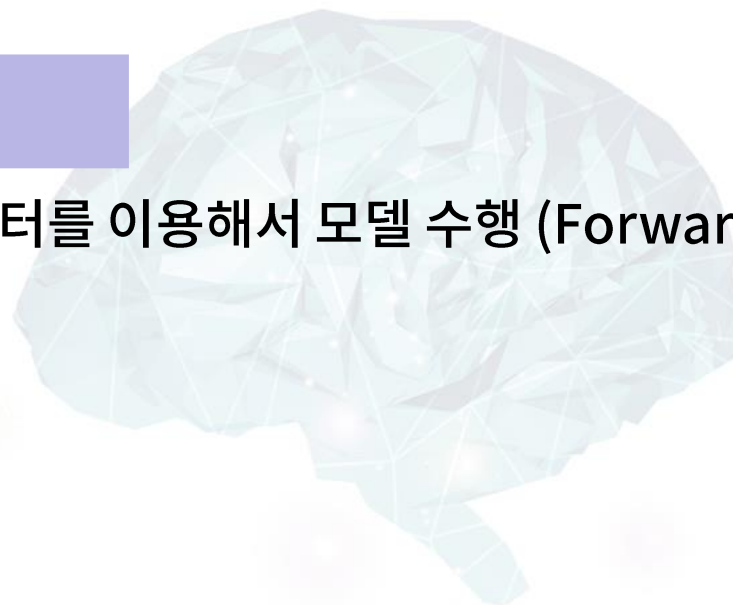
인공 신경망의 구현의 4단계

훈련

- ✓ 가장 많은 시간이 요구됨
- ✓ Forward propagation: 모델에 데이터 적용 → Loss 함수 계산
- ✓ Backward propagation: Gradient 계산 → Gradient descent 수행

테스트 (실행)

- ✓ 훈련이 끝난 파라미터를 이용해서 모델 수행 (Forward propagation)



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

Forward propagation

- 주어진 특징에 대해서 W 를 적용해서 예측 레이블 (Predicted label, \hat{y})을 계산
- 실제 레이블 (True label, y)과 비교 \rightarrow Loss 함수

Backward propagation

- 파라미터 (W)가 Loss 함수 (L)에 미치는 영향을 계산 \rightarrow Gradient 계산 ($\frac{\partial L}{\partial W}$)
- Gradient descent를 적용해서 W 값 갱신

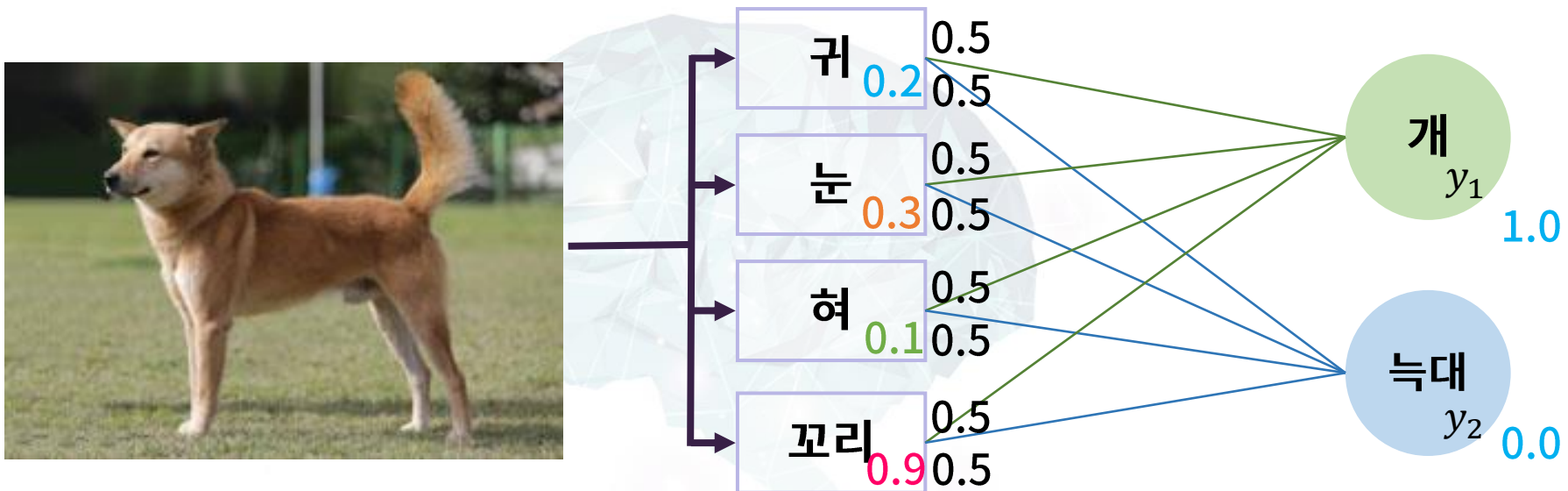
❗ 이 과정을 Loss가 충분히 감소할 때까지 반복

2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

특징 4가지

- ✓ 귀의 크기 (0.2)
- ✓ 눈의 크기 (0.3)
- ✓ 혀의 길이 (0.1)
- ✓ 꼬리의 길이 (0.9)



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- 주어진 특징에 대해서 W 를 적용해서 예측 레이블 (predicted label, \hat{y})을 계산

$$\hat{y} = WX$$

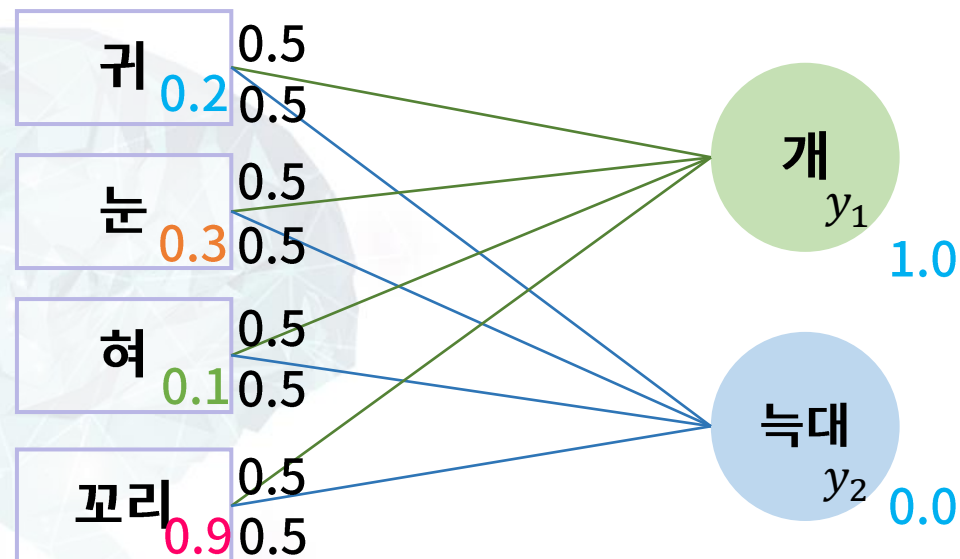
$$\hat{y}_i = w_{1i} * x_1 + w_{2i} * x_2 + w_{3i} * x_3 + w_{4i} * x_4$$

특징에 W 를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L 에 대한 W 의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- 주어진 특징에 대해서 W 를 적용해서 예측 레이블 (predicted label, \hat{y})을 계산

$$\hat{y}_1 = w_{11} * 0.2 + w_{21} * 0.3 + w_{31} * 0.1 + w_{41} * 0.9 = 0.5 * 0.2 + 0.5 * 0.3 + 0.5 * 0.1 + 0.5 * 0.9 = 0.75$$

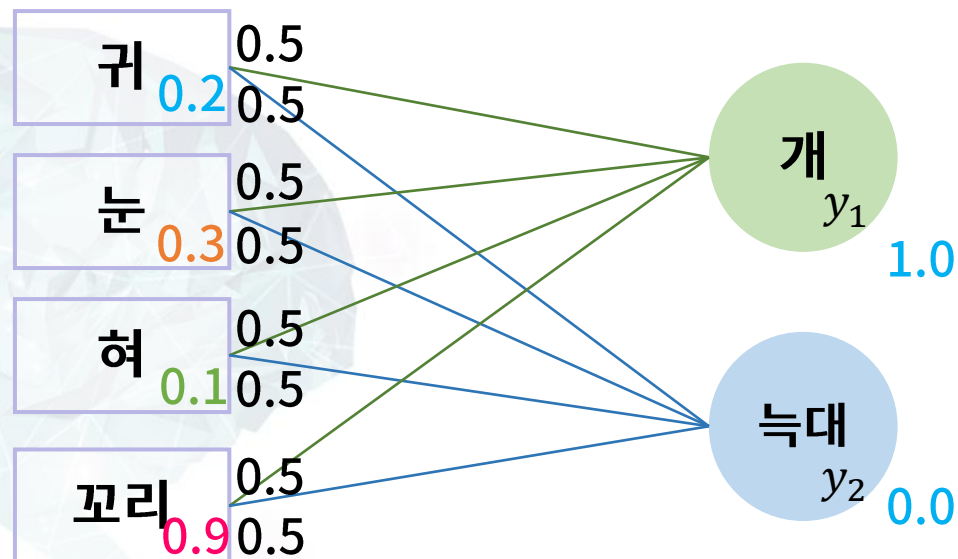
$$\hat{y}_2 = w_{12} * 0.2 + w_{22} * 0.3 + w_{32} * 0.1 + w_{42} * 0.9 = 0.5 * 0.2 + 0.5 * 0.3 + 0.5 * 0.1 + 0.5 * 0.9 = 0.75$$

특징에 W 를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L 에 대한 W 의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- 실제 레이블 (true label, y)과 비교 \rightarrow loss 함수

$$L = \frac{1}{2} (\|\hat{y}_1 - y_1\|^2 + \|\hat{y}_2 - y_2\|^2)$$

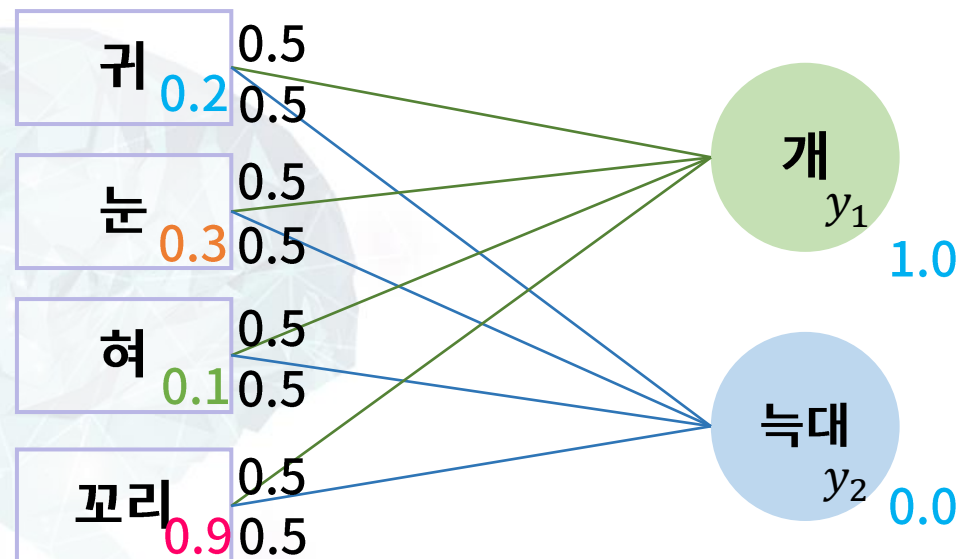
$$L = \frac{1}{2} (\|0.75 - 1.0\|^2 + \|0.75 - 0.0\|^2) = \frac{1}{2} (0.25^2 + 0.75^2) = 0.3125$$

특징에 W 를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L 에 대한 W 의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- 파라미터 (W)가 loss 함수 (L)에 미치는 영향을 계산

$$\frac{\partial L}{\partial w_{11}} \quad \frac{\partial L}{\partial w_{12}} \quad \frac{\partial L}{\partial w_{13}} \quad \frac{\partial L}{\partial w_{14}} \quad \frac{\partial L}{\partial w_{21}} \quad \frac{\partial L}{\partial w_{22}} \quad \frac{\partial L}{\partial w_{23}} \quad \frac{\partial L}{\partial w_{24}}$$

$$L = \frac{1}{2} (\|\hat{y}_1 - y_1\|^2 + \|\hat{y}_2 - y_2\|^2)$$

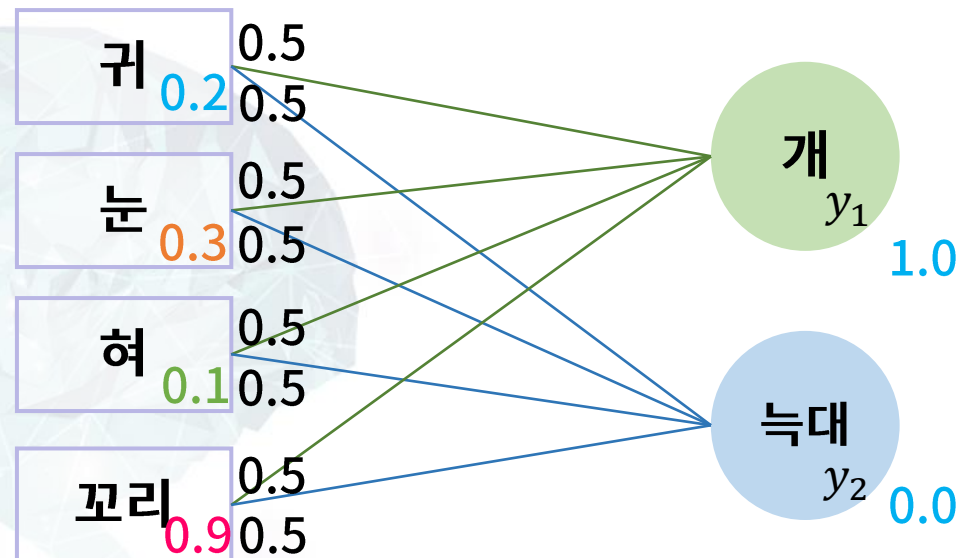
$$= \frac{1}{2} (\|w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 - 1.0\|^2 + \|w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 - 0.0\|^2)$$

특징에 W를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L에 대한 W의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- 파라미터 (W)가 loss 함수 (L)에 미치는 영향을 계산

$$\frac{\partial L}{\partial w_{41}} = x_4(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 - 1.0) = 0.9 * (0.1 + 0.15 + 0.05 + 0.45 - 1.0) = -0.225$$

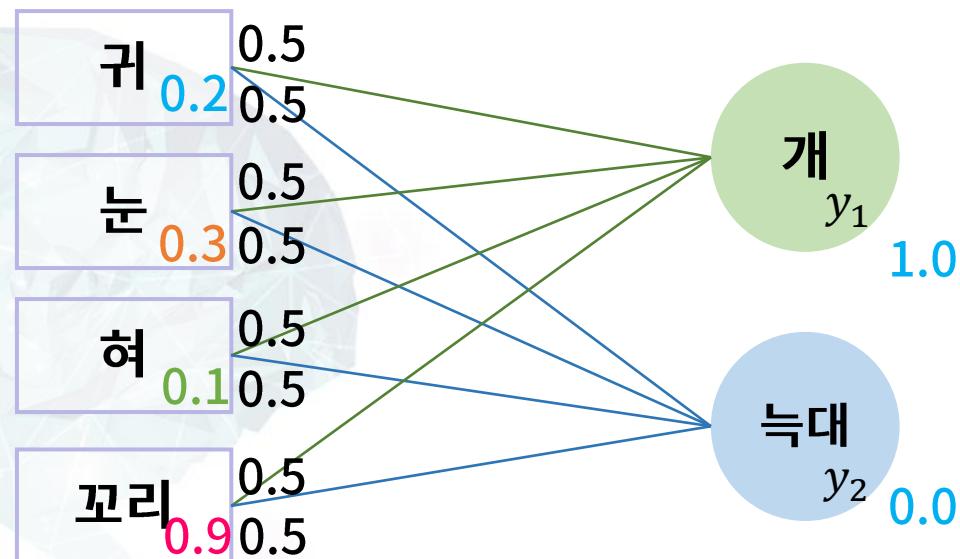
$$\frac{\partial L}{\partial w_{42}} = x_4(w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 - 0.0) = 0.9 * (0.1 + 0.15 + 0.05 + 0.45 - 0.0) = 0.625$$

특징에 W를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L에 대한 W의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- Gradient descent를 적용해서 W값 갱신

$$w = w - lr \frac{\partial L}{\partial w}$$

$$w_{41} = w_{41} - lr \frac{\partial L}{\partial w_{41}} = 0.5 - 0.01 * (-0.225) = 0.5025$$

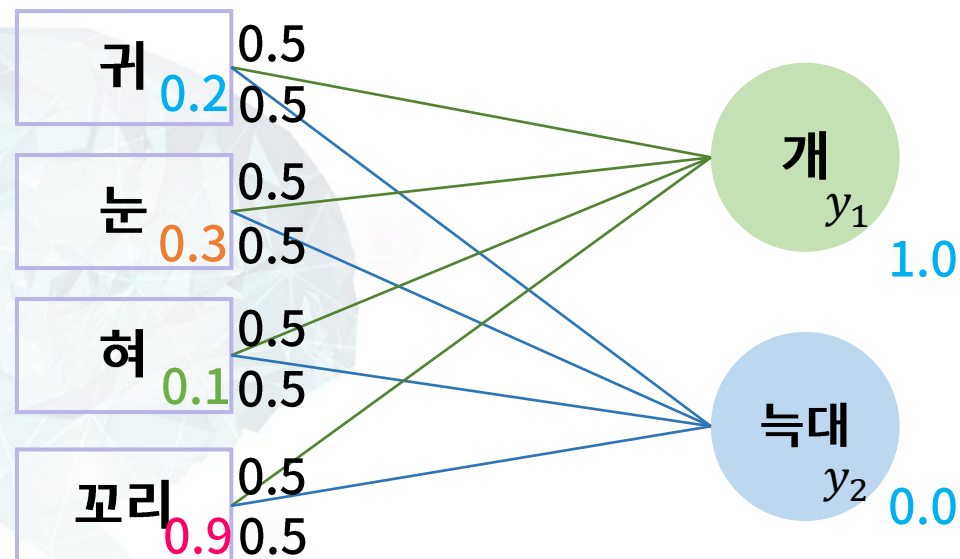
$$w_{42} = w_{42} - lr \frac{\partial L}{\partial w_{42}} = 0.5 - 0.01 * (0.625) = 0.4938$$

특징에 W를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L에 대한 W의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- Gradient descent를 적용해서 W값 갱신

$$w = w - lr \frac{\partial L}{\partial w}$$

$$w_{41} = w_{41} - lr \frac{\partial L}{\partial w_{41}} = 0.5 - 0.01 * (-0.225) = 0.5025$$

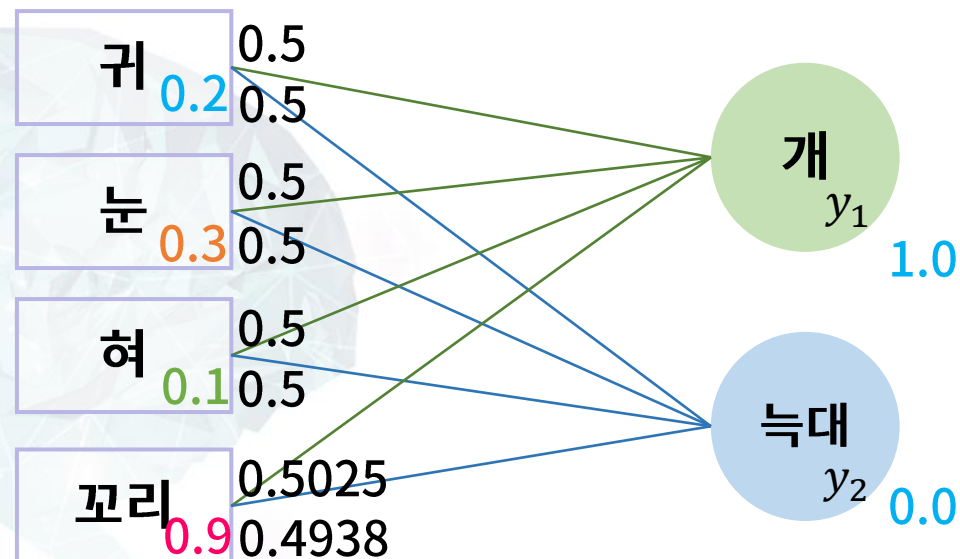
$$w_{42} = w_{42} - lr \frac{\partial L}{\partial w_{42}} = 0.5 - 0.01 * (0.625) = 0.4938$$

특징에 W를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L에 대한 W의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



2. 세상 단순한 인공 신경망의 구현 (이론)

인공 신경망의 훈련

- (다시) 주어진 특징에 대해서 W 를 적용해서 예측 레이블 (predicted label, \hat{y})을 계산

$$\hat{y}_1 = 0.5 * 0.2 + 0.5 * 0.3 + 0.5 * 0.1 + 0.5025 * 0.9 = 0.75225$$

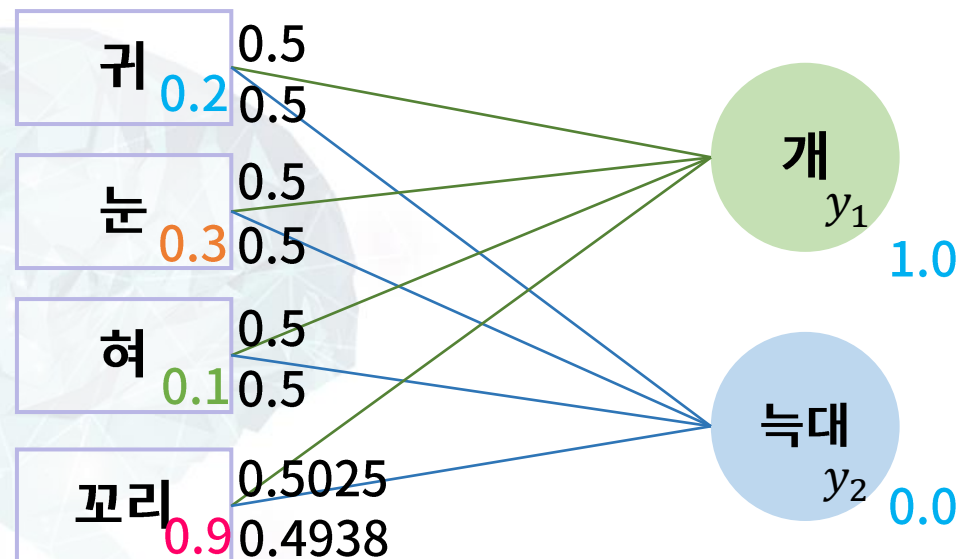
$$\hat{y}_2 = 0.5 * 0.2 + 0.5 * 0.3 + 0.5 * 0.1 + 0.4938 * 0.9 = 0.74442$$

특징에 W 를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L 에 대한 W 의 gradient 계산 ($\frac{\partial L}{\partial w}$)

Gradient descent를 이용해서 W 갱신



인공 신경망의 훈련

- ➡ (다시) 실제 레이블 (true label, y)과 비교 \rightarrow loss 함수

$$L = \frac{1}{2} (\|0.75225 - 1.0\|^2 + \|0.74442 - 0.0\|^2) = \frac{1}{2} (0.24775^2 + 0.74442^2) = 0.3078$$

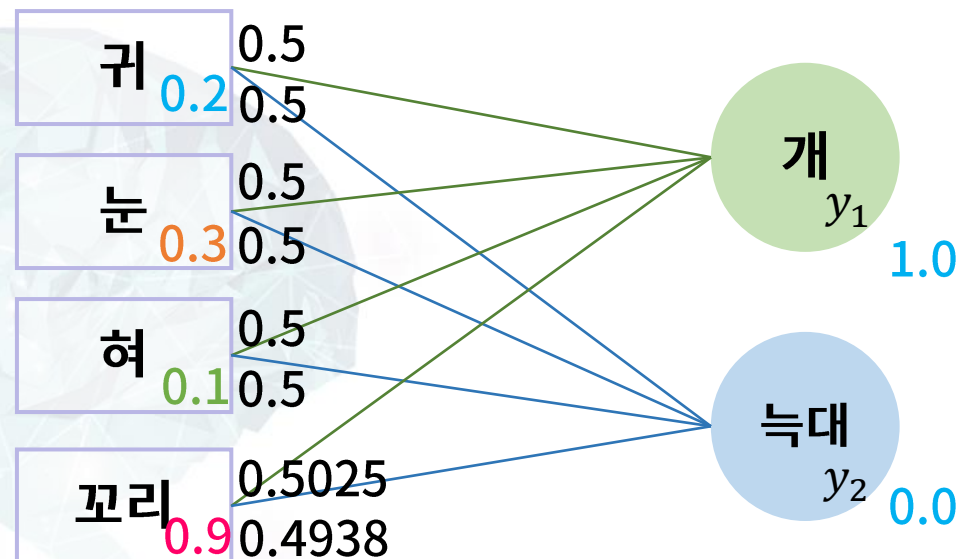
0.3125에 비해서 감소

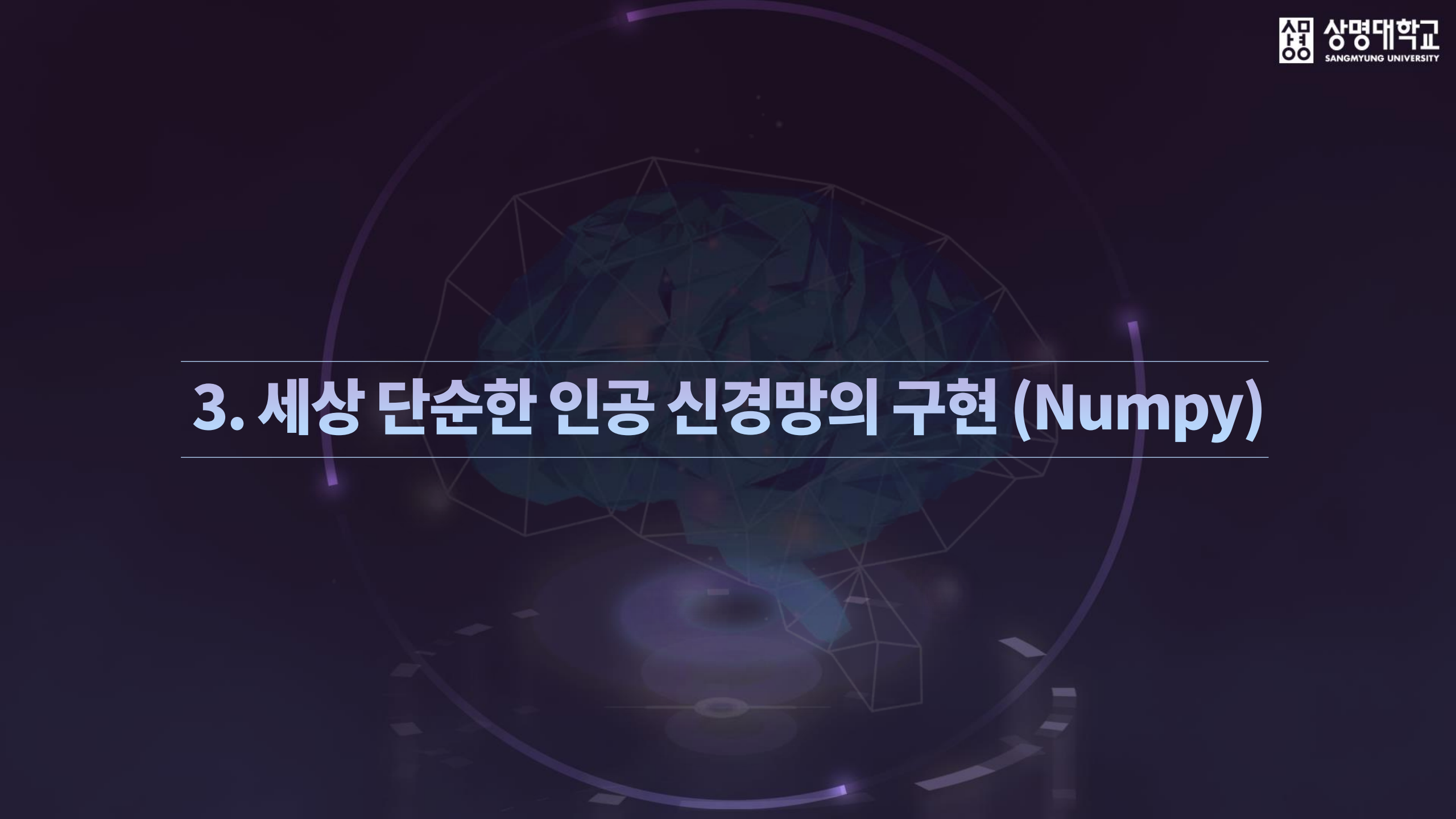
특징에 W 를 적용해서 예측값 \hat{y} 계산

\hat{y} 과 y 를 비교해서 loss 함수 L 계산

L에 대한 W의 gradient 계산 ($\frac{\partial L}{\partial W}$)

Gradient descent를 이용해서 W 갱신





3. 세상 단순한 인공 신경망의 구현 (Numpy)

3. 세상 단순한 인공 신경망의 구현 (Numpy)

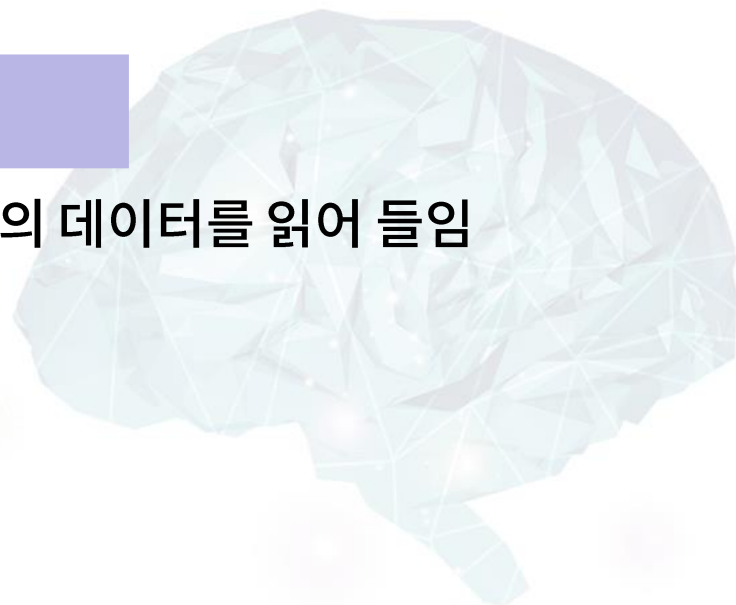
인공 신경망의 구현의 4단계 (복습)

인공 신경망 모델 정의

- ✓ 인공 신경망의 구조를 모델 표현
- ✓ 다양한 라이브러리를 이용한 모델 정의

데이터 로딩

- ✓ 미리 저장된 대용량의 데이터를 읽어 들임



3. 세상 단순한 인공 신경망의 구현 (Numpy)

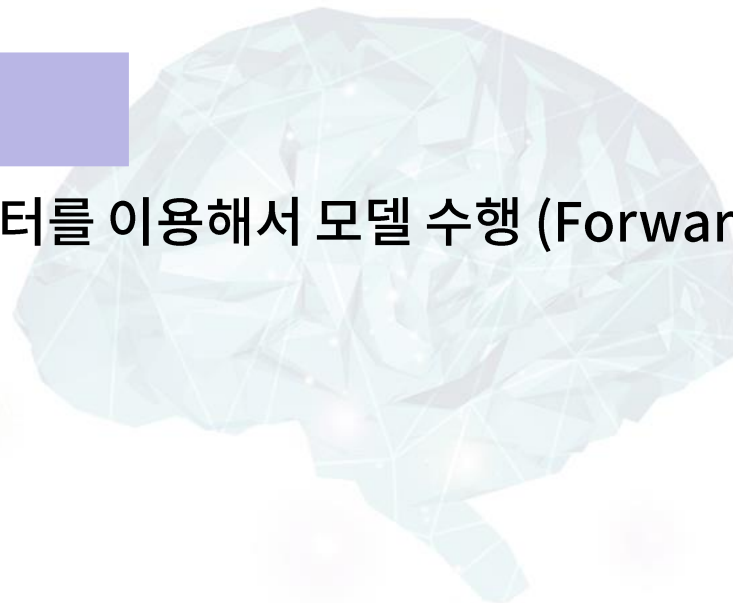
인공 신경망의 구현의 4단계 (복습)

훈련

- ✓ 가장 많은 시간이 요구됨
- ✓ Forward propagation: 모델에 데이터 적용 → Loss 함수 계산
- ✓ Backward propagation: Gradient 계산 → Gradient descent 수행

테스트 (실행)

- ✓ 훈련이 끝난 파라미터를 이용해서 모델 수행 (Forward propagation)



3. 세상 단순한 인공 신경망의 구현 (Numpy)

세상 단순한 인공 신경망

- 인공 신경망 모델 정의

- ✓ 특별히 모델 정의 안함
- ✓ N, D_{in}, D_{out}
- ✓ 사용하는 변수: x, w, y

- 데이터 로딩

- ✓ x, w, y : 난수 발생해서 사용

- 훈련

- ✓ \hat{y} (y_{pred}) 계산
- ✓ loss 계산
- ✓ gradient 계산
- ✓ gradient descent 수행

```
import numpy as np

N, Din, Dout = 64, 4, 2

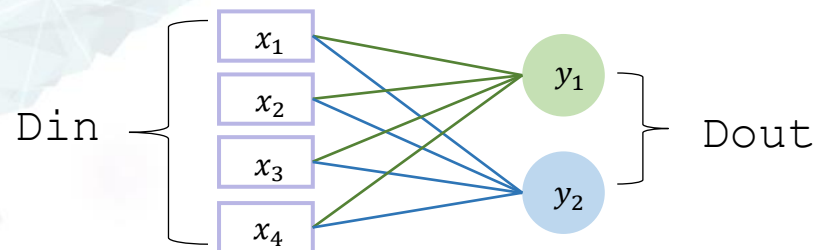
x = np.random.randn(N, Din)
y = np.random.randn(N, Dout)
w = np.random.randn(Din, Dout)

lr = 1e-6
for t in range(5000):
    y_pred = np.matmul(x, w)

    loss = (0.5 * (y_pred - y)**2).sum()

    grad_y_pred = (y_pred - y)
    grad_w = np.matmul(np.transpose(x), grad_y_pred)

    w -= lr * grad_w
```



3. 세상 단순한 인공 신경망의 구현 (Numpy)

세상 단순한 인공 신경망

- 인공 신경망 모델 정의

- ✓ 특별히 모델 정의 안함
- ✓ N, Din, Dout
- ✓ 사용하는 변수: x, w, y

- 데이터 로딩

- ✓ x, w, y: 난수 발생해서 사용

- 훈련

- ✓ \hat{y} (y_pred) 계산
- ✓ loss 계산
- ✓ gradient 계산
- ✓ gradient descent 수행

```
import numpy as np

N, Din, Dout = 64, 4, 2

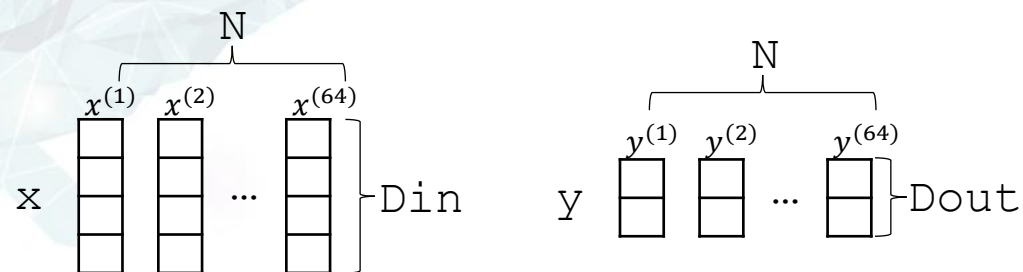
x = np.random.randn(N, Din)
y = np.random.randn(N, Dout)
w = np.random.randn(Din, Dout)

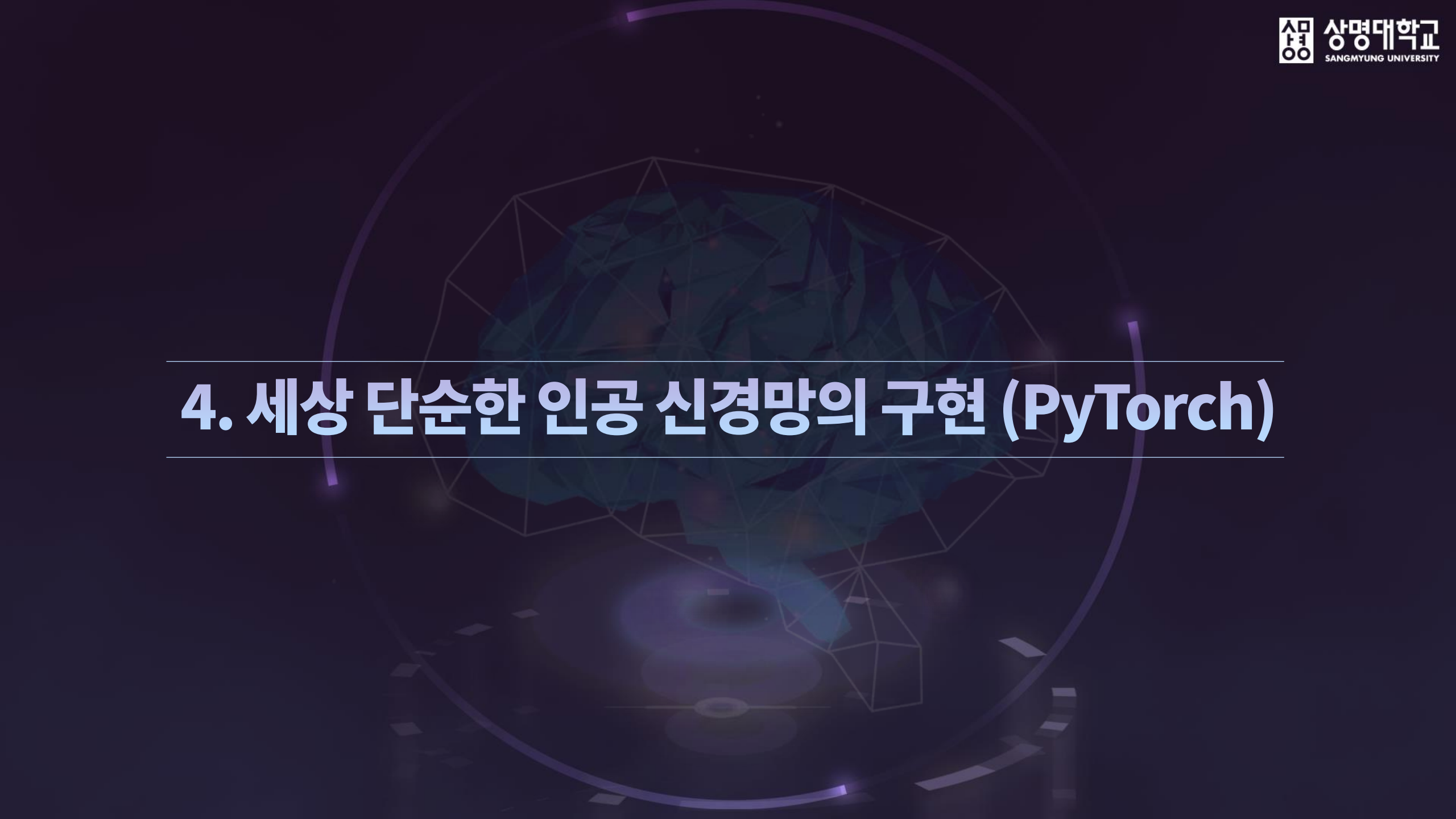
lr = 1e-6
for t in range (5000):
    y_pred = np.matmul(x, w)

    loss = (0.5*(y_pred - y)**2).sum()

    grad_y_pred = (y_pred - y)
    grad_w = np.matmul (np.transpose(x),
                        grad_y_pred)

    w -= lr * grad_w
```





4. 세상 단순한 인공 신경망의 구현 (PyTorch)

4. 세상 단순한 인공 신경망의 구현 (PyTorch)

세상 단순한 인공 신경망 (1)

Pytorch를 이용한 구현

- ✓ import torch
- ✓ device 지정
 - » cpu VS gpu
- ✓ torch.rand () 함수 사용
- ✓ np.array → torch.tensor 기반 연산

```
import numpy as np

N, Din, Dout = 64, 4, 2

x = np.random.randn(N, Din)
y = np.random.randn(N, Dout)
w = np.random.randn(Din, Dout)

lr = 1e-6
for t in range (5000):
    y_pred = np.matmul(x, w)

    loss = (0.5*(y_pred - y)**2).sum()

    grad_y_pred = (y_pred - y)
    grad_w = np.matmul (np.transpose(x),
                        grad_y_pred)

    w -= lr * grad_w
```



```
import torch

device = torch.device ('cpu')

N, Din, Dout = 64, 4, 2

x = torch.randn(N, Din, device = device)
y = torch.randn(N, Dout, device = device)
w = torch.randn(Din, Dout, device = device)

lr = 1e-6
for t in range (500):
    y_pred = x.mm(w)

    loss = 0.5*(y_pred - y).pow(2).sum()

    grad_y_pred = (y_pred - y)
    grad_w = x.t().mm(grad_y_pred)

    w -= lr * grad_w
```

4. 세상 단순한 인공 신경망의 구현 (PyTorch)

세상 단순한 인공 신경망 (2)

- autograd

- ☑ backward () 함수를 이용한 자동 gradient 계산

```
import torch

N, Din, Dout = 64, 4, 2

x = torch.randn(N, Din, device = device)
y = torch.randn(N, Dout, device = device)
w = torch.randn(Din, Dout, device = device,
                 requires_grad=True)

lr = 1e-6
for t in range (500):
    y_pred = x.mm(w)

    loss = 0.5*(y_pred - y).pow(2).sum()

    loss.backward ( )

    with torch.no_grad():
        w -= lr * w.grad
        w.grad.zero_()
```


4. 세상 단순한 인공 신경망의 구현 (PyTorch)

세상 단순한 인공 신경망 (3)

nn 라이브러리 이용

- ✓ nn.Sequential ()을 이용한 model 정의
- ✓ nn.Linear ()를 이용한 linear layer 표현
- ✓ nn.functional.mse_loss ()를 이용한 loss 함수 계산

LINEAR

CLASS torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None) [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

This module supports [TensorFloat32](#).

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shapes:

- Input: $(N, *, H_{in})$ where $*$ means any number of additional dimensions and $H_{in} = \text{in_features}$
- Output: $(N, *, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} = \text{out_features}$

Variables

- **-Linear.weight** – the learnable weights of the module of shape $(\text{out_features}, \text{in_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in_features}}$
- **-Linear.bias** – the learnable bias of the module of shape (out_features) . If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in_features}}$

```
import torch
```

```
N, Din, Dout = 64, 4, 2
```

```
x = torch.randn(N, Din, device = device)
```

```
y = torch.randn(N, Dout, device = device)
```

```
w = torch.randn(Din, Dout, device = device,
                 requires_grad=True)
```

```
lr = 1e-6
```

```
for t in range (500):
```

```
    y_pred = x.mm(w)
```

```
    loss = 0.5*(y_pred - y).pow(2).sum()
```

```
    loss.backward ( )
```

```
    with torch.no_grad():
```

```
        w -= lr * w.grad
```

```
        w.grad.zero_()
```

4. 세상 단순한 인공 신경망의 구현 (PyTorch)

세상 단순한 인공 신경망 (4)

nn 라이브러리 이용

- ✓ Optimizer를 이용한 gradient descent 수행

- ✓ 많이 사용하는 Optimizer

 - » Adam

 - » SGD

```
import torch

N, Din, Dout = 64, 4, 2

x = torch.randn(N, Din, device = device)
y = torch.randn(N, Dout, device = device)

model = torch.nn.Sequential (
    torch.nn.Linear(Din, Dout))

lr = 1e-6

optimizer = torch.optim.Adam(model.parameters(),
                               lr = lr)

for t in range (500):
    y_pred = x.mm(w)

    loss = torch.nn.functional.mse_loss (y_pred, y)

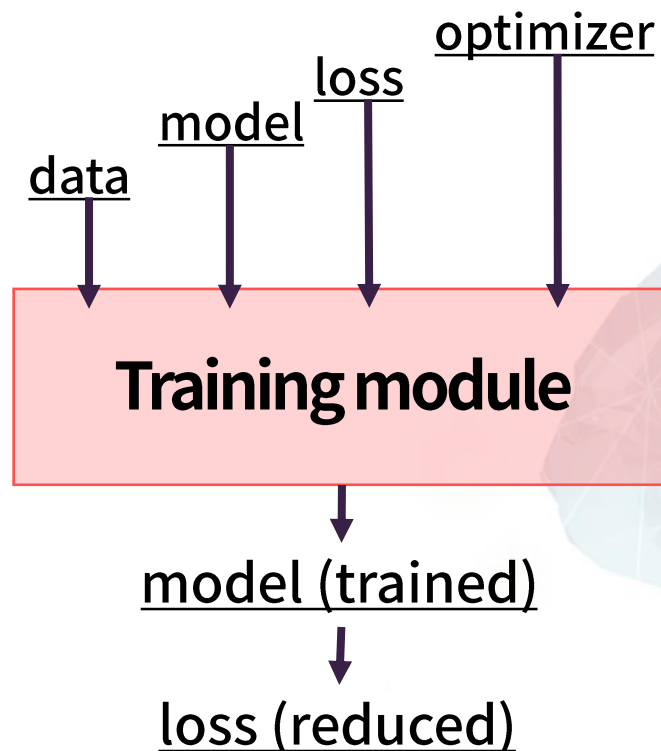
    loss.backward ( )

    optimizer.step()
    optimizer.zero_grad()
```

4. 세상 단순한 인공 신경망의 구현 (PyTorch)

세상 단순한 인공 신경망 (5)

- nn.Module을 상속받는 class 선언을 통한 모델 구현



```
import torch

class MyOneLayerNet ( torch.nn.Module ):
    def __init__(self, Din, Dout):
        super(MyOneLayerNet, self).__init__()
        self.linear = torch.nn.Linear(Din, Dout)

    def forward (self, x):
        y_pred = self.linear(x)
        return y_pred

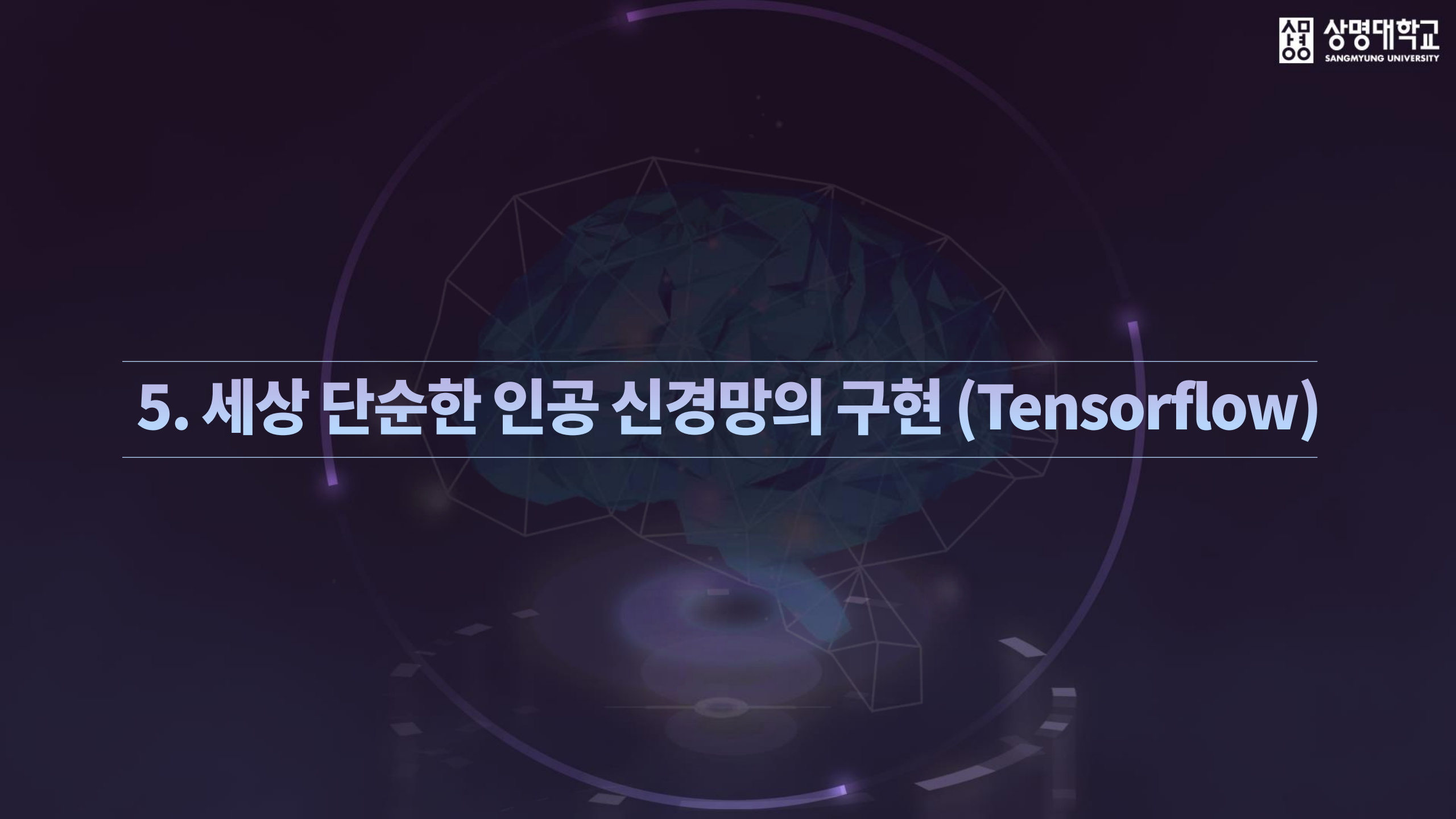
N, Din, Dout = 64, 4, 2
x = torch.randn(N, Din, device = device)
y = torch.randn(N, Dout, device = device)

model = MyOneLayerNet ( Din, Dout )
optimizer = torch.optim.Adam(model.parameters(),
                               lr = 1e-6)

for t in range (500):
    y_pred = model (x)
    loss = torch.nn.functional.mse_loss (y_pred, y)

    loss.backward ( )

    optimizer.step()
    optimizer.zero_grad()
```



5. 세상 단순한 인공 신경망의 구현 (Tensorflow)

5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)

PyTorch vs Tensorflow

- 가장 많이 사용되는 Deep learning 구현 도구

Caffe
(UC Berkeley)
C++ 기반, 2014



Torch
(NYU/Facebook)
Lua 기반, 2002



Theano
(Univ. of Montreal)
Python 기반, 2007

theano

PyTorch
(Facebook)
Python, C++ 기반, 2016



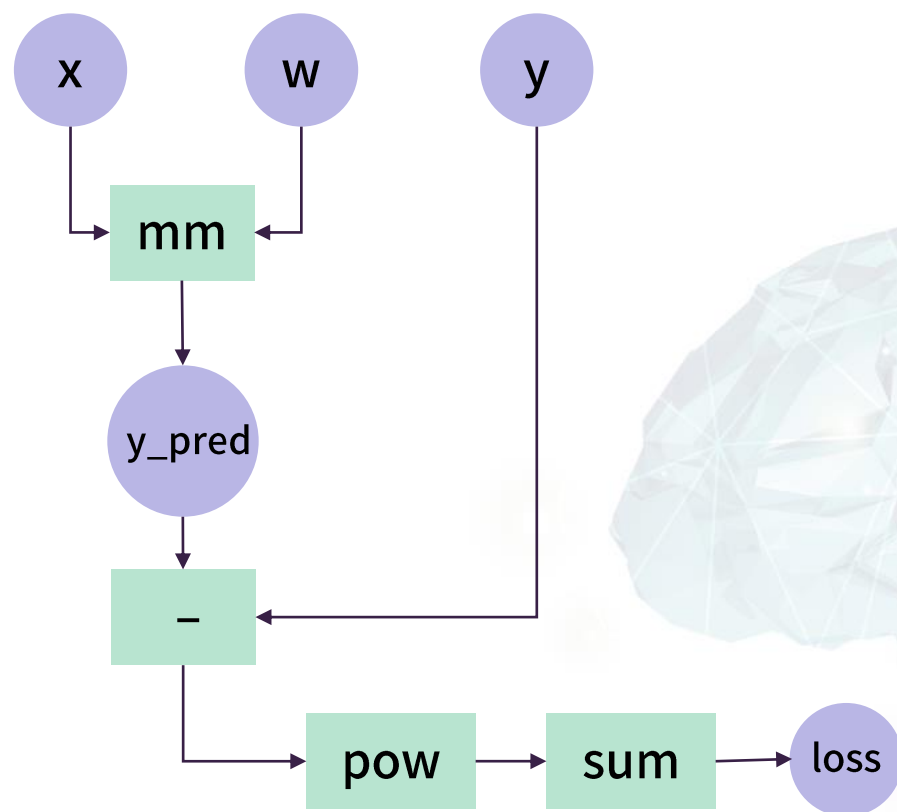
Tensorflow
(Google)
Python, C++ 기반, 2015



5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)

PyTorch vs Tensorflow

- Forward propagation 과정은 computational graph로 표현할 수 있음



```
for t in range (500):  
    y_pred = x.mm(w)  
  
    loss = 0.5*(y_pred - y).pow(2).sum()  
  
    grad_y_pred = (y_pred - y)  
    grad_w = x.t().mm(grad_y_pred)  
  
    w -= lr * grad_w
```

5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)

PyTorch vs Tensorflow

PyTorch

Dynamic computational graph
변수 선언과 할당을 동시에 수행

```
x = torch.randn(N, Din, device = device)
y = torch.randn(N, Dout, device = device)
w = torch.randn(Din, Dout, device = device)

for t in range (500):
    y_pred = x.mm(w)
    loss = 0.5*(y_pred - y).pow(2).sum()
    grad_y_pred = (y_pred - y)
    grad_w = x.t().mm(grad_y_pred)

    w -= lr * grad_w
```

TensorFlow

Static computational graph
변수 선언과 할당을 분리해서 수행

```
x = tf.placeholder(tf.float32,shape=(N, Din))
y = tf.placeholder(tf.float32,shape=(N, Dout))
w = tf.placeholder(tf.float32,shape=(Din,Dout))

with tf.Session() as sess:
    values = {x: np.random.randn(N, Din),
              y: np.random.randn(N, Dout),
              w: np.random.randn(Din, Dout),}

    out = sess.run([loss], feed_dict=values)
```


5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)

PyTorch vs Tensorflow

PyTorch

Dynamic computational graph
그래프 생성과 연산을 동시에 수행

```
for t in range (500):  
    y_pred = x.mm(w)  
  
    loss = 0.5*(y_pred - y).pow(2).sum()  
  
    grad_y_pred = (y_pred - y)  
    grad_w = x.t().mm(grad_y_pred)  
  
    w -= lr * grad_w
```

TensorFlow

Static computational graph
그래프 생성과 연산을 분리해서 수행

```
y_pred = tf.matmul(w, x)  
diff = y_pred - y  
loss = tf.reduce_mean  
        (tf.reduce_sum (diff**2, axis=1))  
  
with tf.Session() as sess:  
    out = sess.run ([loss])
```

5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)

TensorFlow 기반 구현

- 변수 선언
- Computational graph 구성
- 실행 (tf.Session())

```
import numpy as np
import tensorflow as tf
N, Din, Dout = 64, 4, 2

X = tf.placeholder(tf.float32, shape=(N, Din))
Y = tf.placeholder(tf.float32, shape=(N, Dout))
W = tf.placeholder(tf.float32, shape=(Din, Dout))

Y_pred = tf.matmul(w, x)
Diff = y_pred - y
Loss = tf.reduce_mean
      (tf.reduce_sum (diff**2, axis=1))

grad_w = tf.gradients(loss, [w])

with tf.Session() as sess:
    values = {x: np.random.randn(N, Din),
              y: np.random.randn(N, Dout),
              w: np.random.randn(Din, Dout),}

    out = sess.run([loss, grad_w],
                    feed_dict=values)
    loss_val, grad_w_val = out
```



5. 세상 단순한 인공 신경망의 구현 (Tnesorflow)

PyTorch

```
import torch

device = torch.device ('cpu')

N, Din, Dout = 64, 4, 2

x = torch.randn(N, Din, device = device)
y = torch.randn(N, Dout, device = device)
w = torch.randn(Din, Dout, device = device)

lr = 1e-6
for t in range (500):
    y_pred = x.mm(w)

    loss = 0.5*(y_pred - y).pow(2).sum()

    grad_y_pred = (y_pred - y)
    grad_w = x.t().mm(grad_y_pred)

    w -= lr * grad_w
```

TensorFlow

```
import numpy as np
import tensorflow as tf
N, Din, Dout = 64, 4, 2

X = tf.placeholder(tf.float32, shape=(N, Din))
Y = tf.placeholder(tf.float32, shape=(N, Dout))
W = tf.placeholder(tf.float32, shape=(Din, Dout))

Y_pred = tf.matmul(w, x)
Diff = y_pred - y
Loss = tf.reduce_mean
        (tf.reduce_sum (diff**2, axis=1))

grad_w = tf.gradients(loss, [w])

with tf.Session() as sess:
    values = {x: np.random.randn(N, Din),
              y: np.random.randn(N, Dout),
              w: np.random.randn(Din, Dout),}

    out = sess.run([loss, grad_w],
                    feed_dict=values)
    loss_val, grad_w_val = out
```