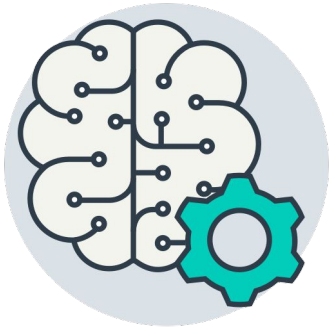
# 다루고자 하는 주제

- Data definition
- Hypothesis
- Compute loss
- Gradient descent

# Data definition

What would be the grade if I study 4 hours?

4 hours



? points

**Prediction**

| Hours (x) | Points (y) |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | ? |

**Training dataset**

**Test dataset**

# Data definition

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])
```

$$X_{\text{train}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad Y_{\text{train}} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

| Hours (x) | Points (y) |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |

- 데이터는 `torch.tensor`!
- 입력 따로, 출력 따로!
  - 입력: `x_train`
  - 출력: `y_train`
  - 입출력은 x, y 로 구분

# Hypothesis

$$y = Wx + b$$

Weight

Bias

# Hypothesis

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])
```

```
W = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)
hypothesis = x_train * W + b
```

- Weight 와 Bias 0으로 초기화
  - 항상 출력 0을 예측
- `requires_grad=True`
  - 학습할 것이라고 명시

$$y = Wx + b$$

# Compute loss

Mean Squared Error (MSE)

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} \left( H(x^{(i)}) - y^{(i)} \right)^2$$

Mean

Prediction

Target

# Compute loss

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])
```

```
W = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)
hypothesis = x_train * W + b
```

```
cost = torch.mean((hypothesis - y_train) ** 2)
```

- `torch.mean` 으로 평균 계산!
- 한 줄인데 읽기 편한 코드 :)

$$\frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)})-y^{(i)}\right)^2$$

# Gradient descent

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])
```

```
W = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)
hypothesis = x_train * W + b
```

```
cost = torch.mean((hypothesis - y_train) ** 2)
```

```
optimizer = optim.SGD([W, b], lr=0.01)


optimizer.zero_grad()
cost.backward()
optimizer.step()
```

- `torch.optim` 라이브러리 사용
  - `[W, b]` 는 학습할 tensor들
  - `lr=0.01` 은 learning rate

- 항상 붙어다니는 3줄
  - `zero_grad()` 로 gradient 초기화
  - `backward()` 로 gradient 계산
  - `step()` 으로 개선

# Full training code

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])

W = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)

optimizer = optim.SGD([W, b], lr=0.01)

nb_epochs = 1000
for epoch in range(1, nb_epochs + 1):
    hypothesis = x_train * W + b
    cost = torch.mean((hypothesis - y_train) ** 2)

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()
```

한번만

1. 데이터 정의
2. Hypothesis 초기화
3. Optimizer 정의

반복!

1. Hypothesis 예측
2. Cost 계산
3. Optimizer 로 학습

# What’s Next?

- How does gradient descent minimize cost?