Multi-modal deep learning and fusion

https://youtu.be/JFL7Ovrjbr4

Contents

CryptoCraft LAB

Multi-modal deep learning

- **multi-modal**
  - 다양한 형태의 데이터
  - 하나의 source를 나타내는 여러 종류의 데이터들

    ex) 동영상 → 소리, 프레임, 프레임의 흐름, 자막(텍스트) 등

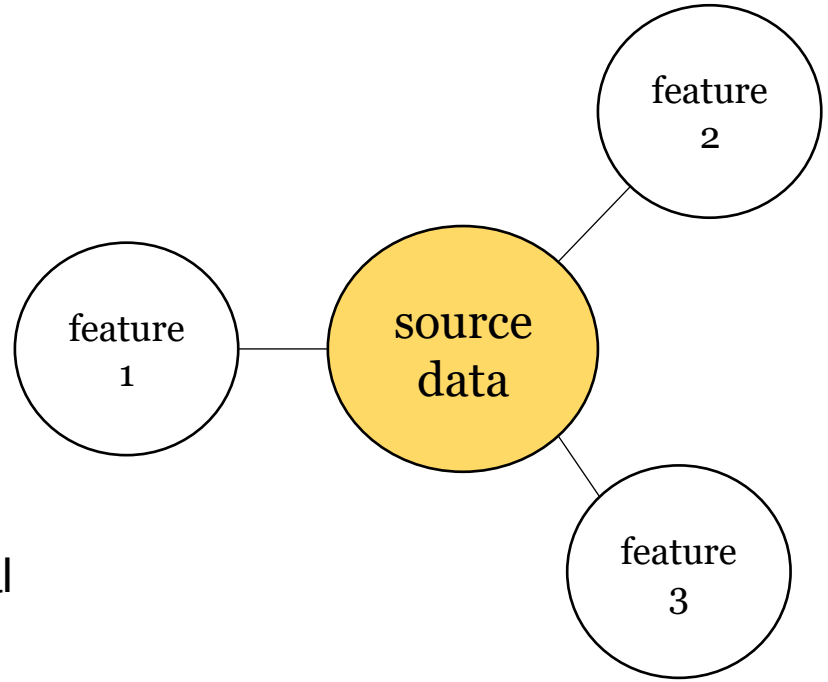
    고양이 → 소리, 촉감, 냄새 등
  - 그냥 feature가 많은 것이 아니라 다른 차원의 data

    ex) 고양이의 소리를 나타내는 값을 쭉 나열한 것은 single-modal

    → 소리 or 촉감 or …

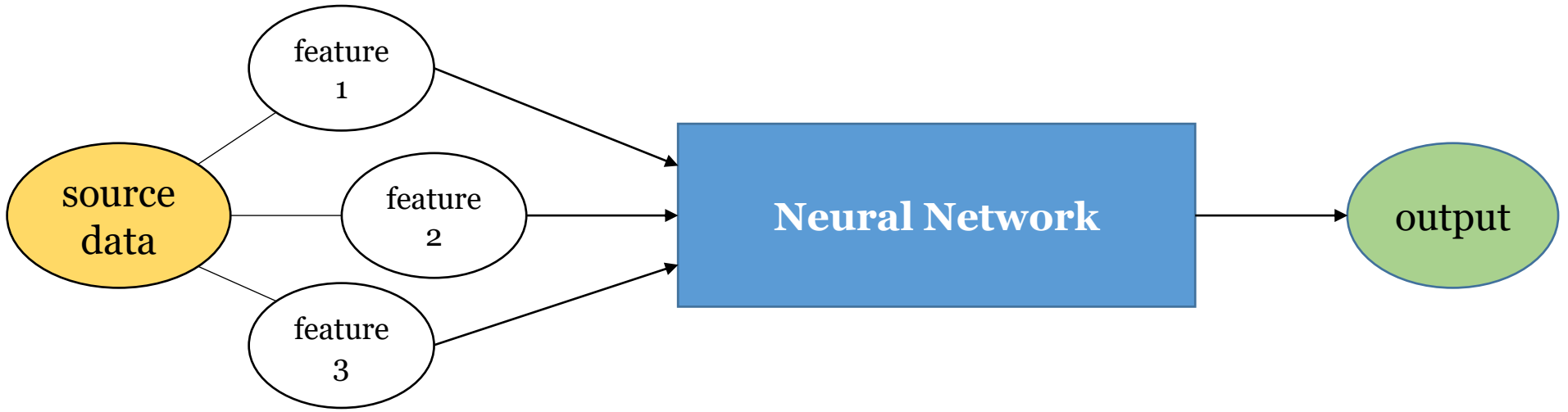
    고양이의 소리, 촉감, 냄새 등에 대한 정보가 있으면 multi-modal

    → 소리 and 촉감 and …



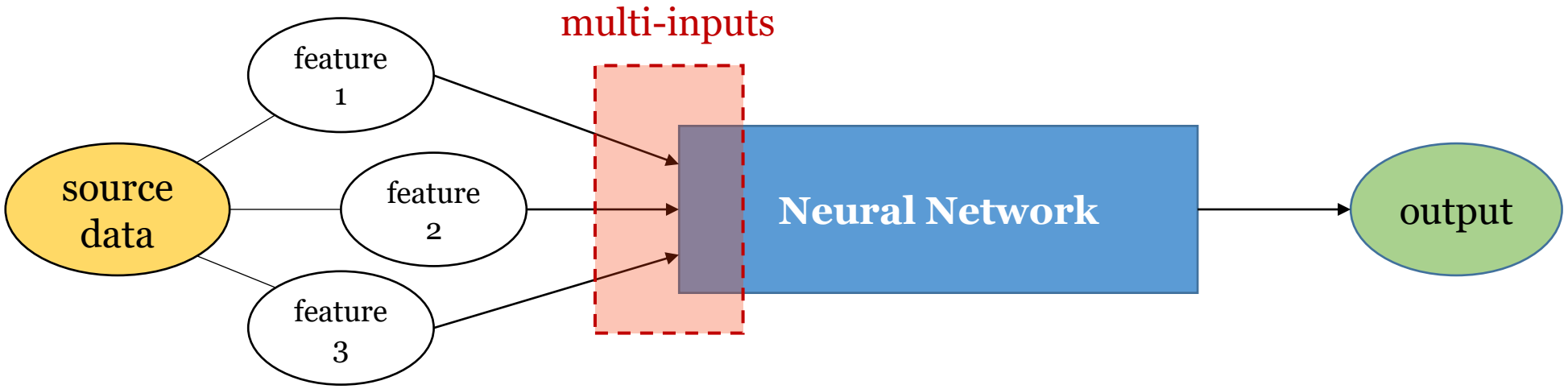
Multi-modal deep learning

- **multi-modal deep learning**
  - multimodal data를 입력으로 하여 학습
  - 하나의 정보를 나타내는 각 데이터들의 특징을 잘 조합
  - 즉, 여러 데이터들을 가지고 해당 source data가 가지는 label을 더 잘 예측할 수 있도록 조합되어야.
  - 여러 정보를 보완적으로 활용하므로 일반적으로 single-modal보다 성능이 좋음



Multi-modal deep learning

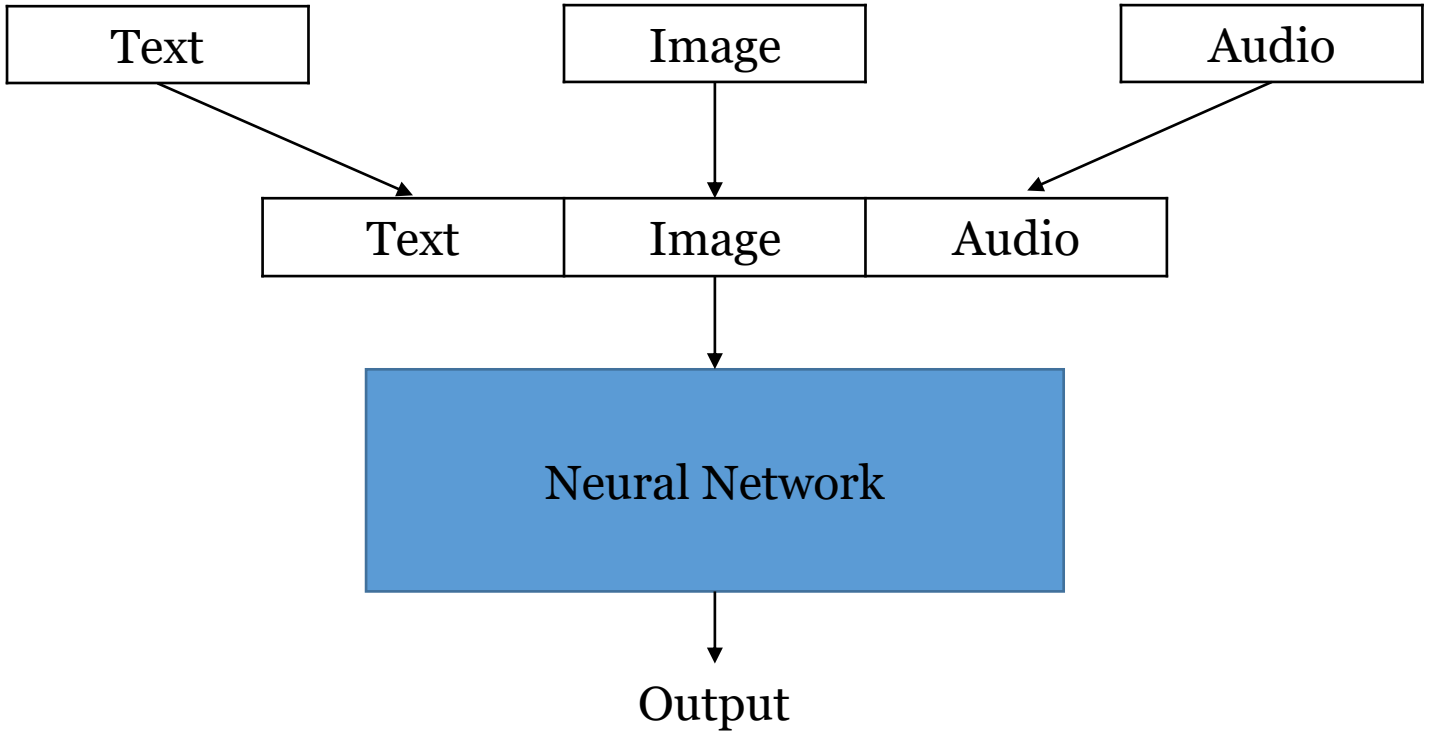
- **multi-inputs**
  - 다음과 같은 3가지 유형으로 설계 가능
    1. data concat
    2. trained feature (feature map) concat
    3. classifier (network) concat



Data concat

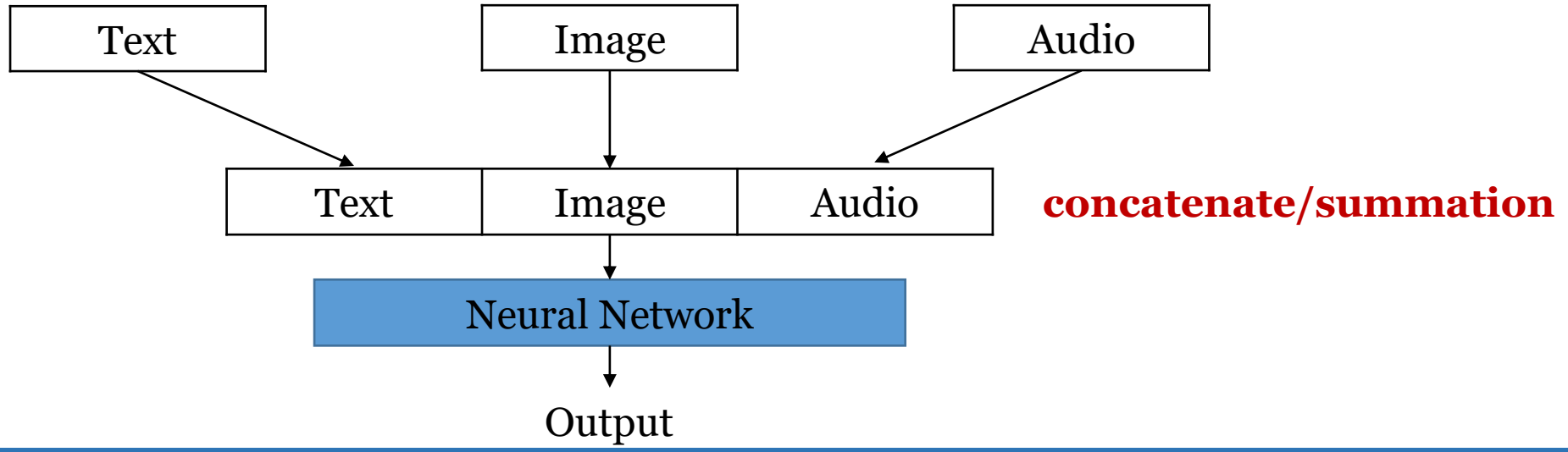
- **Input data 통합**
  - 각 데이터들을 통합해서 하나의 데이터로 추출

    ex) 이미지, 텍스트 등 서로 다른 차원의 데이터들을 통합



Early fusion

- data concat임, 즉, 신경망에 입력하기 전에 단일 특성 벡터로 통합 $f_L = \mathbf{H}_L(\mathbf{H}_{L-1}(\cdots \mathbf{H}_1(f_{BV} \oplus f_{FV} \oplus f_{RGB})))$
- 데이터의 차원 축소, 데이터 소스간의 샘플링 속도 통일 등의 이슈가 있음
  → 데이터 통합 과정에서 많은 양의 데이터들이 공제됨
- 학습은 한 번만 하면 되지만, 각 feature들을 하나의 공통된 벡터로 표현하기 어려움
- 그러나 여러 특성을 갖는 데이터를 학습하는 것 (뒤에 나올 late fusion에 비해 좀 더 진정한 multi-feature 표현)



차원축소

- PCA (주성분 분석) → 차원 축소
  - data 하나에 대한 것을 분석하는 것이 아니라,
    여러 데이터들이 모여 하나의 분포를 이룰 때, 해당 분포에서의 주성분을 분석
  - 원래 데이터의 분포를 최대한 보존하면서, 새로운 기저를 찾음
    → 고차원 공간의 데이터를 저차원으로 변환
  - 아래와 같은 데이터 셋의 경우 하나의 데이터가 11차원을 가짐 → 데이터 분석이 어려움

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4

*주성분
: 데이터들의 분산이 가장 큰 방향벡터, 각 주성분들은 수직
→ 따라서, 주성분 벡터는 n차원(데이터의 차원) 공간을 나타내는 기저
: 한 장의 100*100크기의 이미지는 10000차원 벡터이고,
해당 이미지는 10000차원에서 한 점에 대응

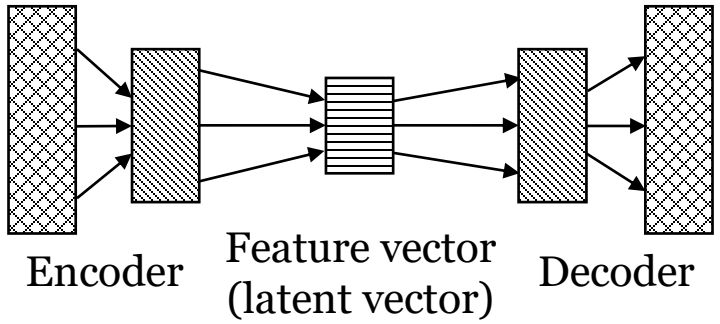
차원축소

- AutoEncoder (오토인코더)

- encoder → latent vector → decoder

  : 입력에서 특징 추출 후, 해당 특징 값 (인코더를 통해 압축된 특징 벡터)을 기반으로 원래의 인풋으로 복구

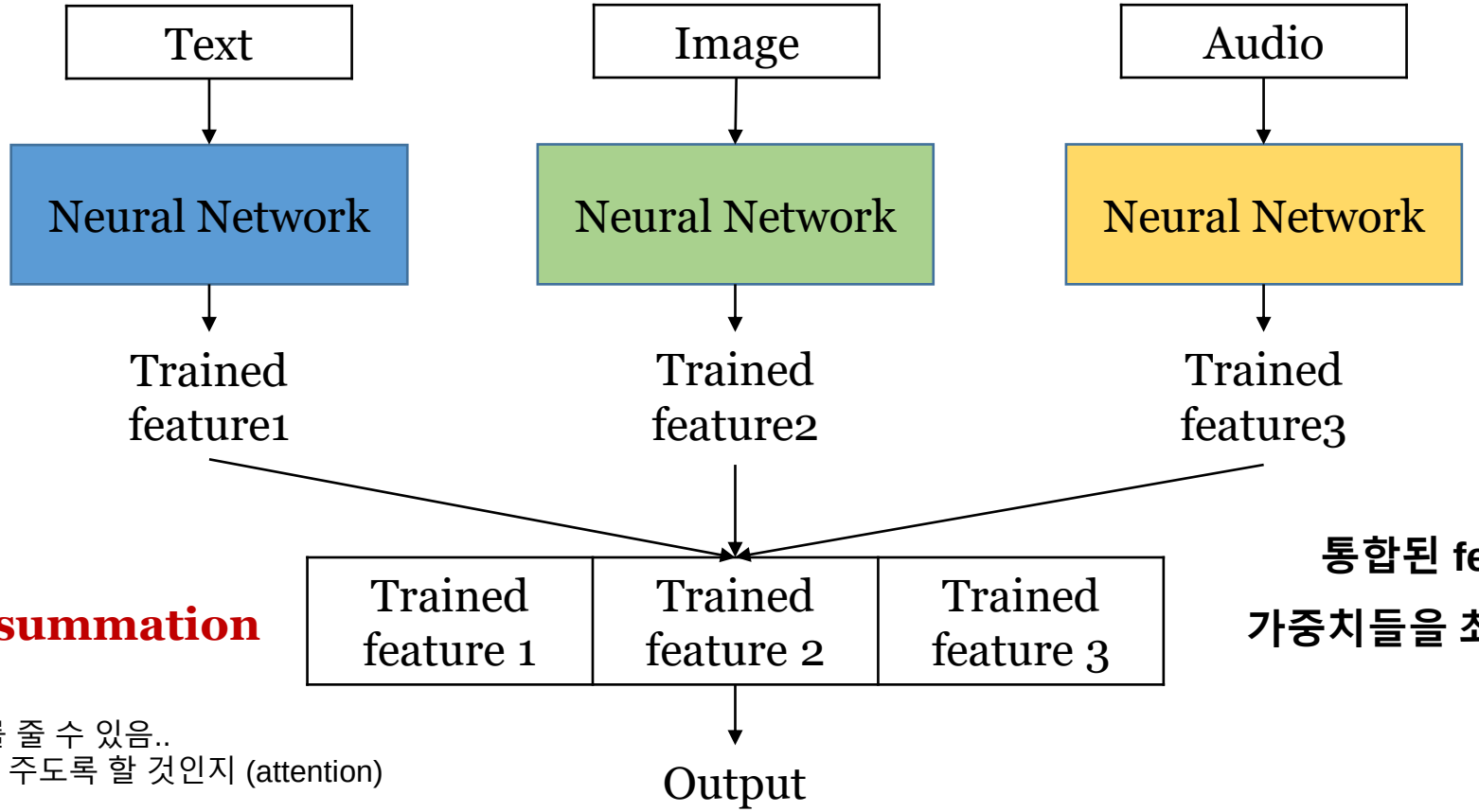
  → 원래의 특성을 보존하면서 노이즈는 없애고 특징 값을 잘 살림



Trained feature concat

$$f_L = (\mathbf{H}_L^{BV}(\cdots \mathbf{H}_1^{BV}(f_{BV}))) \oplus (\mathbf{H}_L^{FV}(\cdots \mathbf{H}_1^{FV}(f_{FV}))) \oplus (\mathbf{H}_L^{RGB}(\cdots \mathbf{H}_1^{RGB}(f_{RGB})))$$

- **Feature map이 통합되는 것**
  - 각 데이터가 서로 다른 신경망에 입력되어 추출된 특징값을 통합



"Late fusion"

concatenate/summation

통합된 feature vector가 갖는 가중치들을 최종 학습 (layer 필요)하여 output 결정

*해당 가중치들에 가중치?를 줄 수 있음..
어떤 feature를 더 큰 영향을 주도록 할 것인지 (attention)

Trained feature concat

- **Late fusion**
  - 각 입력 데이터에 대해 독립적으로 학습한 후 concat
  - 따라서 각 데이터, 각 모델에 대한 학습(에러 전파)가 독립적으로 수행 → 주로 다중 데이터 학습 시 해당 방법 사용
  - 각 데이터 차원을 통합할 필요가 없음

    → 각기 다른 형태의 데이터들이 다수 존재할 때 유용

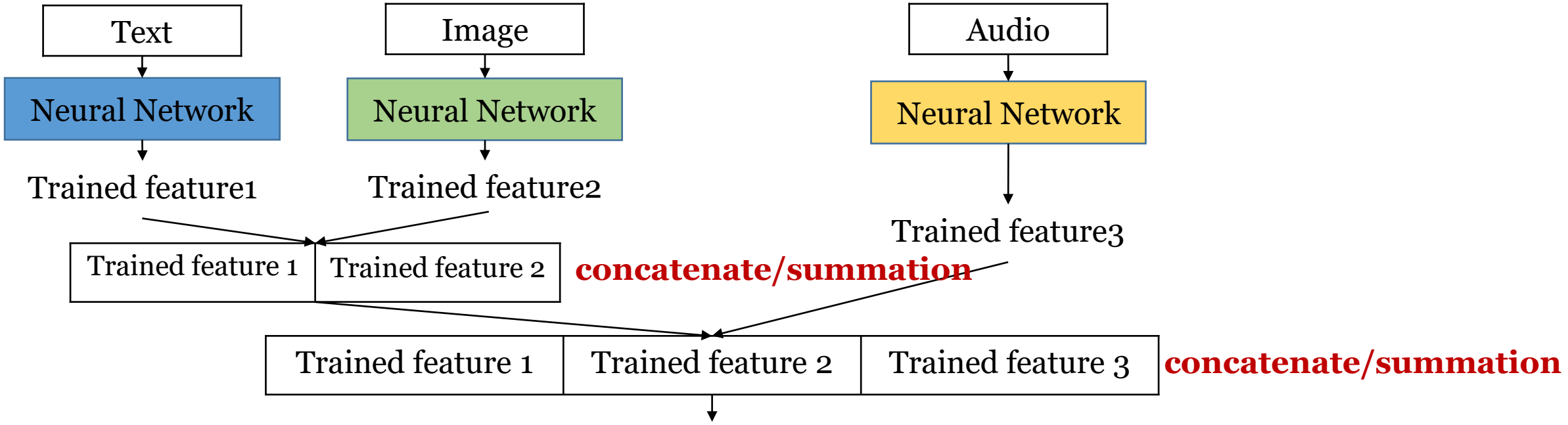
    → 즉, early fusion보다 더 유연하고 빠름
  - 여러 입력 중 하나가 잘못되는 경우 등에 대처 가능

    ex) 다중 센서 경우 하나의 센서가 동작하지 않는 경우에도 다른 센서들의 값으로 학습하여 커버 가능
  - 입력데이터의 신뢰성이나 중요도에 따라 가중치를 다르게 부여할 경우 더 신뢰성 있는 학습이 가능할 듯..
  - 그러나, 각 데이터마다 독립적으로 학습 → 자원 소모

    다양한 데이터 간의 잠재적인 상관관계에 대한 학습은 어렵

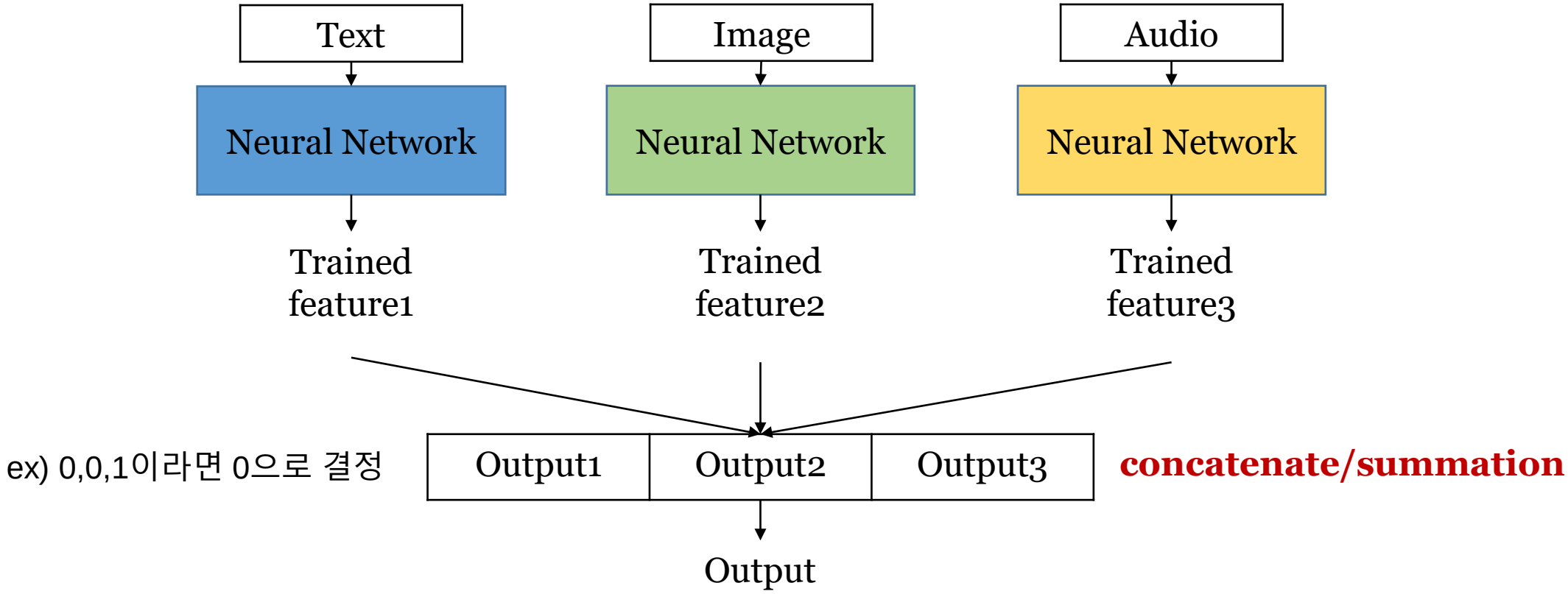
Trained feature concat

- **Intermediate fusion**
  - fusion 방법론 중 가장 유연한 방법
  - 앞서 본 방법들처럼 하나의 layer에 전부 합치거나, 2개의 modality는 통합하고, 다른 부분은 나중에 점진적으로 통합
    → 상관관계가 더 클 것으로 예상되는 입력 데이터들을 먼저 통합
    상관관계가 비교적 낮을 것으로 예상되는 입력 데이터들은 나중에 통합



Classifier concat

- 학습 결과를 통합
- late fusion의 경우 여러 network를 통해 feature를 output으로 하여 융합
- classifier concat의 경우 각각의 네트워크가 각각의 label을 가지고 학습된 후, 해당 결과를 통합



구현 예시 – WISA'20 확장..

PRINCE binary file

```
<SR>:
602:  8c 81   ldd   r24, Y+4
604:  88 2f   mov   r24, r24
      ...
```

…

SPARX binary file

```
<specky>:
1bde:  e9 81   ldd   r30, Y+1
1be0:  fa 81   ldd   r31, Y+2
       ...
```

SPARX binary file

```
<roundF>:
602:  84 27   eor   r24, r20
604:  95 27   eor   r25, r21
      ...
```

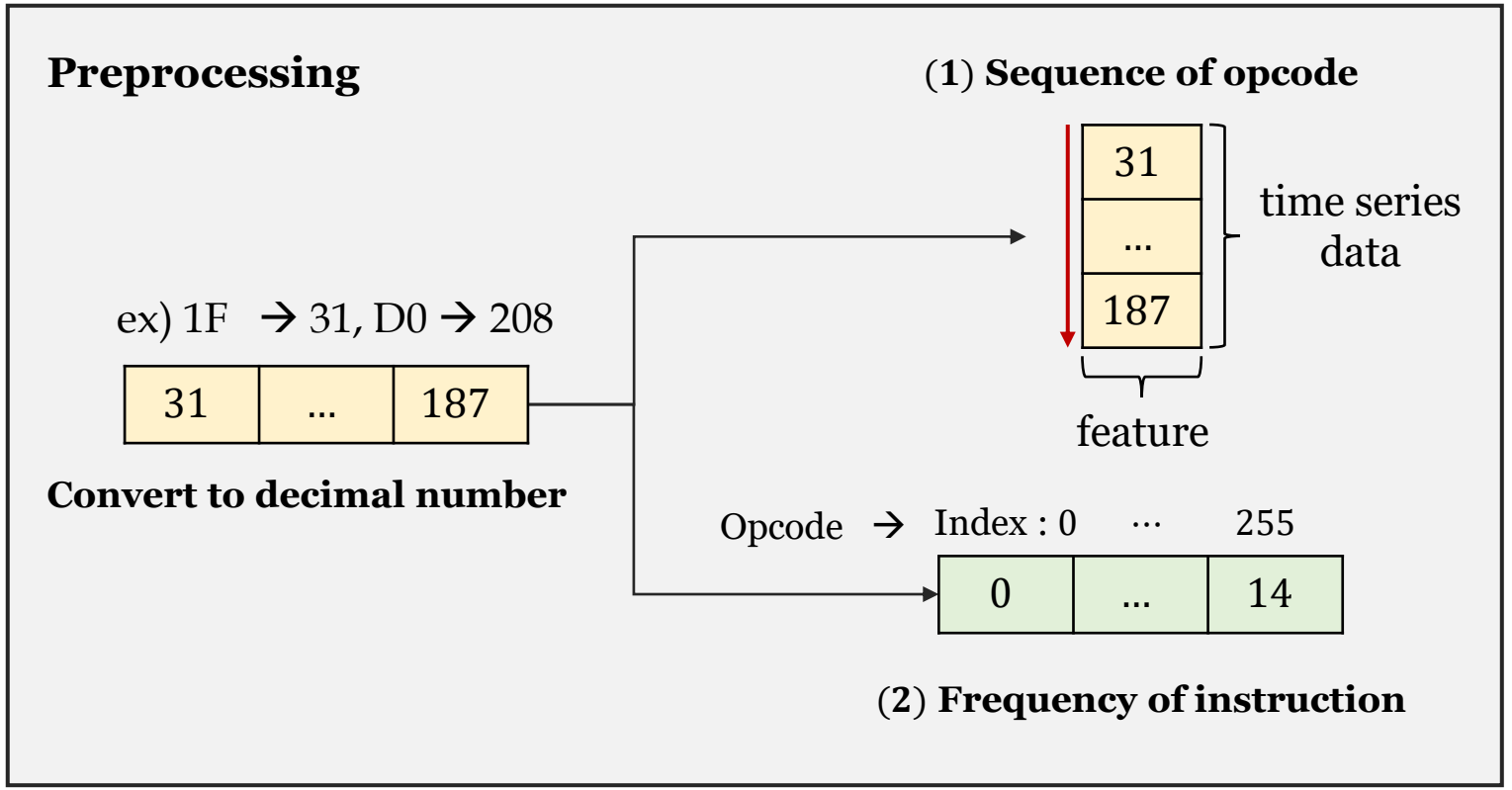
Extract opcode

Opcode sequence for each function

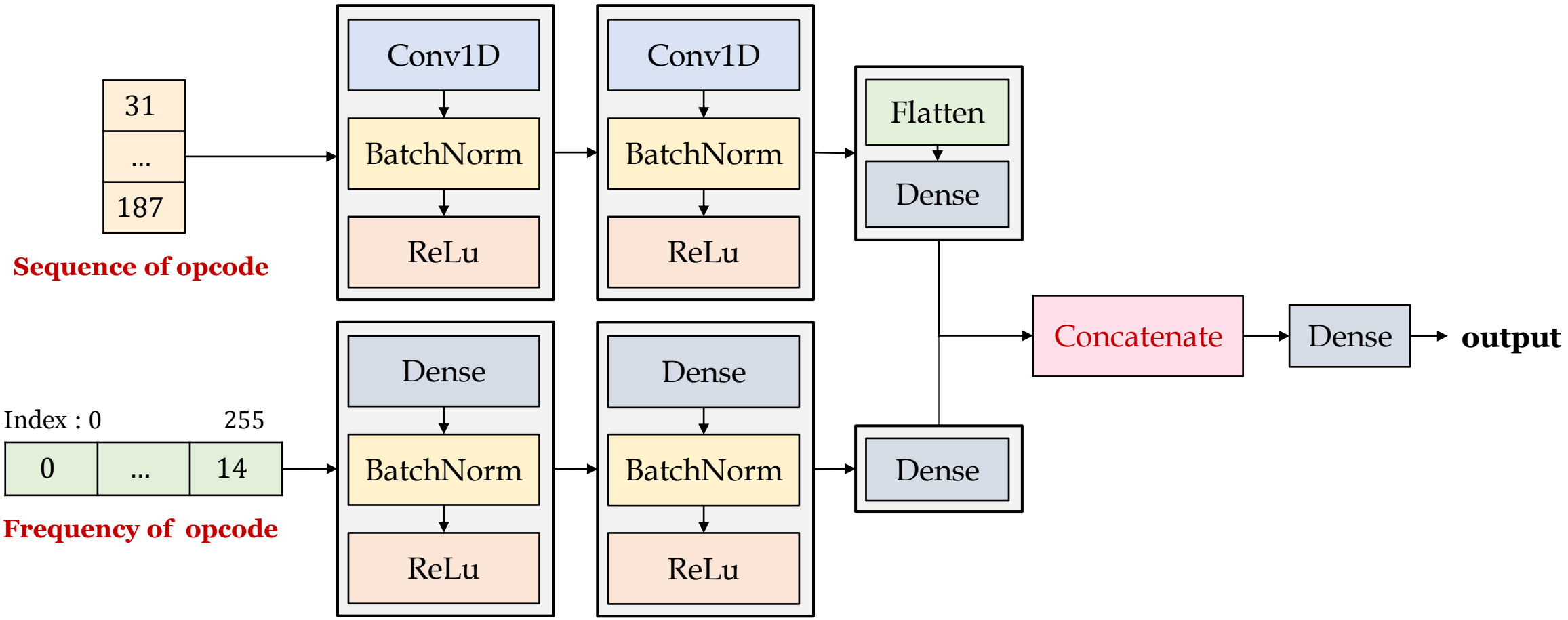
function 1	81	2f	…	95	PRINCE
function 2	81	81	…	95	SPARX
	…	…	…	…	…
function *n*	27	27	…	95	SPARX

Data **Label**

구현 예시 – WISA'20 확장..



구현 예시 – WISA'20 확장..



구현 예시 – WISA'20 확장..

```
input1 = Input(shape=(x1_train.shape[1:]))
conv1 = Conv1D(256, kernel_size=16, strides= 1)(input1)
conv1 = BatchNormalization()(conv1)
conv1 = Activation('relu')(conv1)

conv1 = Conv1D(32, kernel_size=4, strides = 1)(input1)
conv1 = BatchNormalization()(conv1)
conv1 = Activation('relu')(conv1)

conv1 = Flatten()(conv1)
dense1 = Dense(12, activation='relu')(conv1)


input2 = Input(shape=x2_train.shape[1])
dnn2 = Dense(256)(input2)
dnn2 = BatchNormalization()(dnn2)
dnn2 = Activation('relu')(dnn2)

dnn2 = Dense(32)(dnn2)
dnn2 = BatchNormalization()(dnn2)
dnn2 = Activation('relu')(dnn2)
dense2 = Dense(12, activation='relu')(dnn2)


merged = concatenate([dense1, dense2])
out = Dense(12, activation='softmax')(merged)

model = Model(inputs = [input1, input2], outputs = [out])
optimizer = Adam(lr=0.0002)
model.compile(optimizer = optimizer, loss = 'categorical_cro
```

- 2 input + late fusion이라 모델을 2개 구성 (CNN, FCN)
- concatenate 통해 융합 후 1개의 레이어 쌓아서 학습
  (아웃풋이 아니라 특징벡터 → 학습 필요)
- 최종 모델
  (input1, input2) → input (concat후 dense layer 거친 값) → output

```
history = model.fit([x1_train,x2_train], y_train, validation_data=([x1_val,x2_val],y_val) , batch_size=batch_num, verbose = 2, epochs=epoch_num)
```

- 피팅하는 부분
  → label은 1개, 입력은 2개
  → 두 타입의 인풋이 하나의 데이터를 나타내는 특징이며,
  하나로 분류됨 (역전파가 두 모델의 입력까지 가는 것임)

Q & A