DeepIntoDeep
1. Machine Learning and MLP

Seongchan Kim

2020320120@korea.ac.kr
Artificial Intelligence in KU (AIKU)
Department of Computer Science and Engineering, Korea University

AIKU

Part 1. Deep

Seongchan Kim

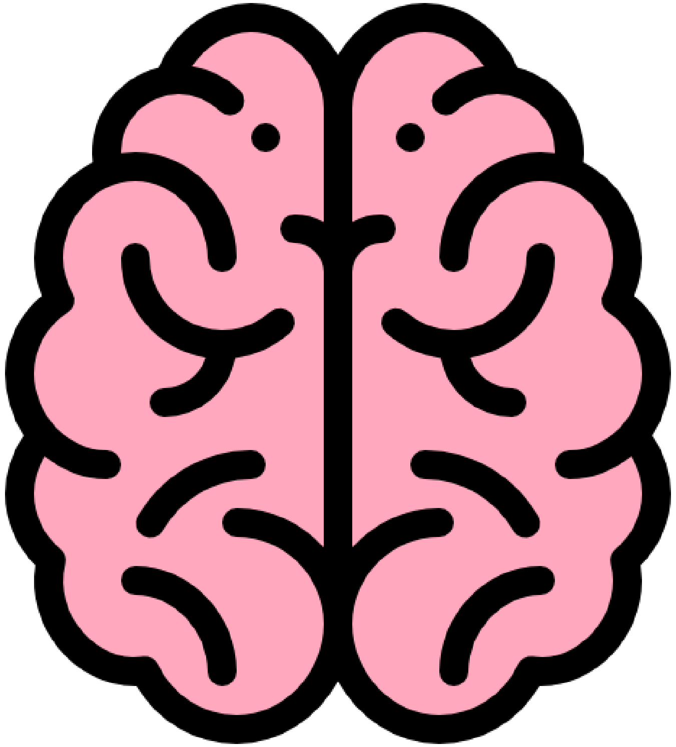
2020320120@korea.ac.kr

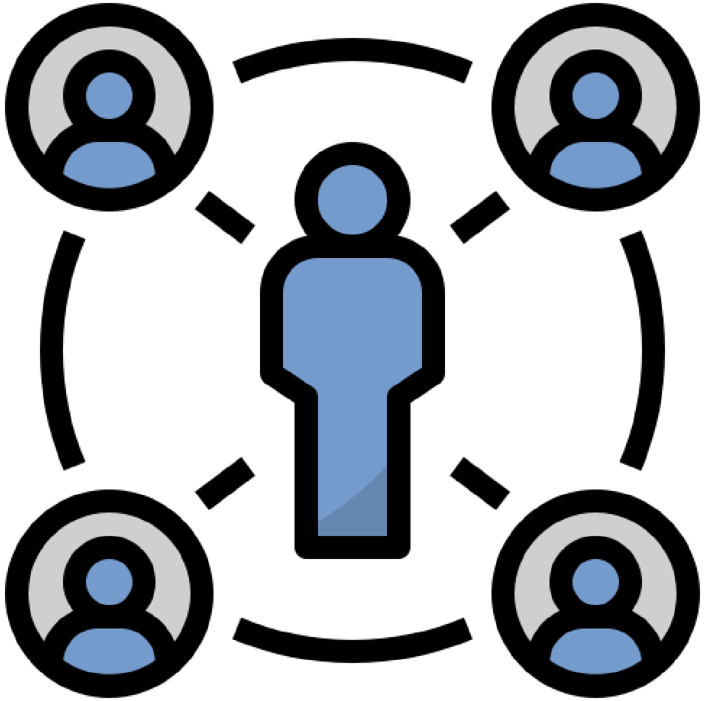
Artificial Intelligence in KU (AIKU)

Department of Computer Science and Engineering, Korea University

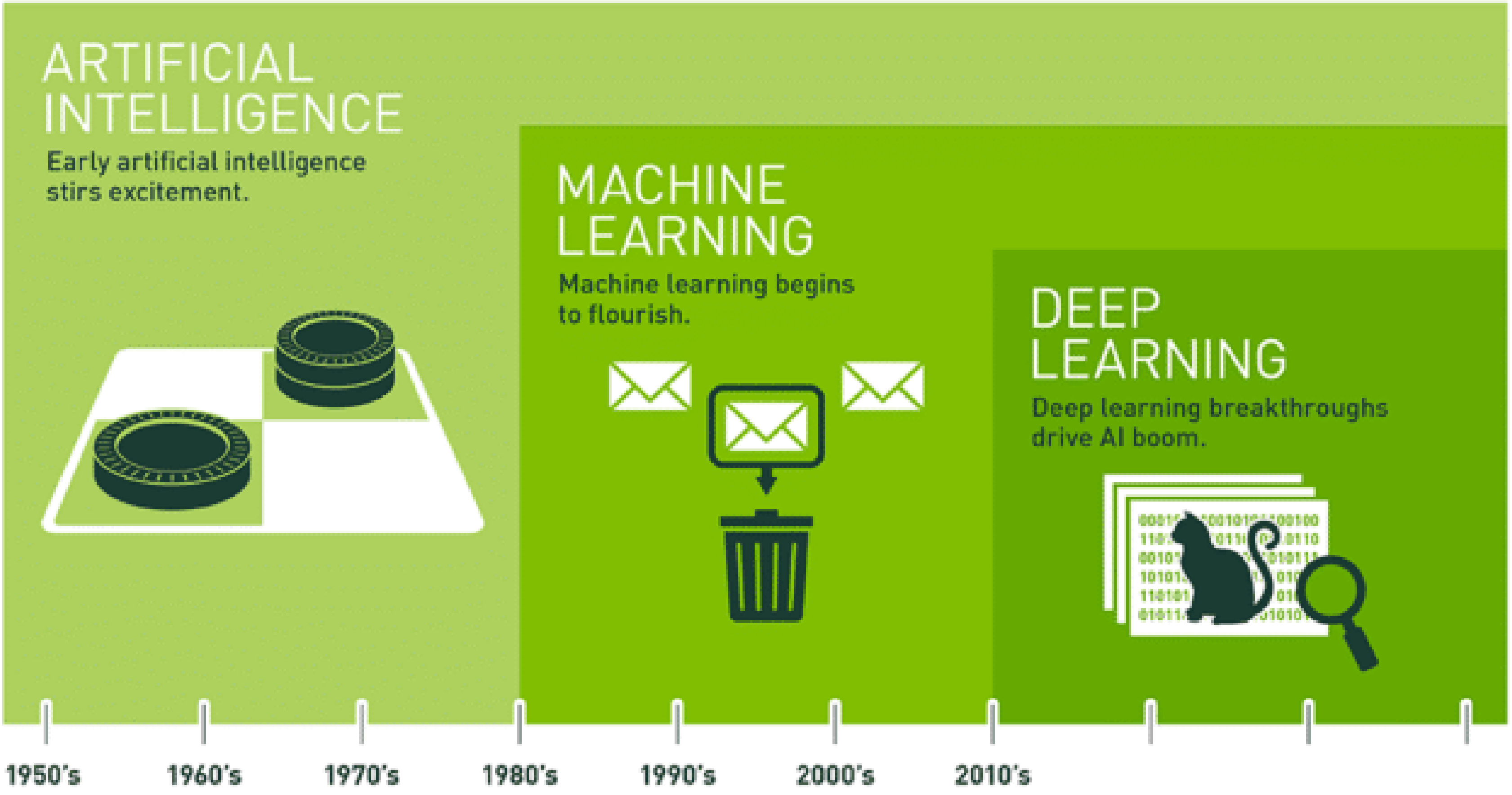
AIKU

What is Intelligence?



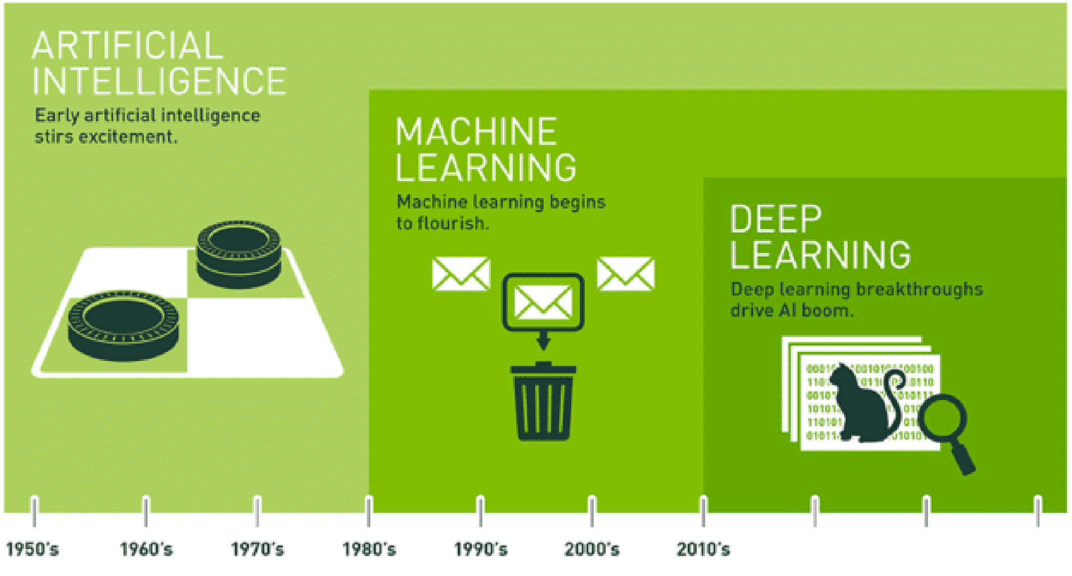


AI vs. Machine Learning vs. Deep Learning

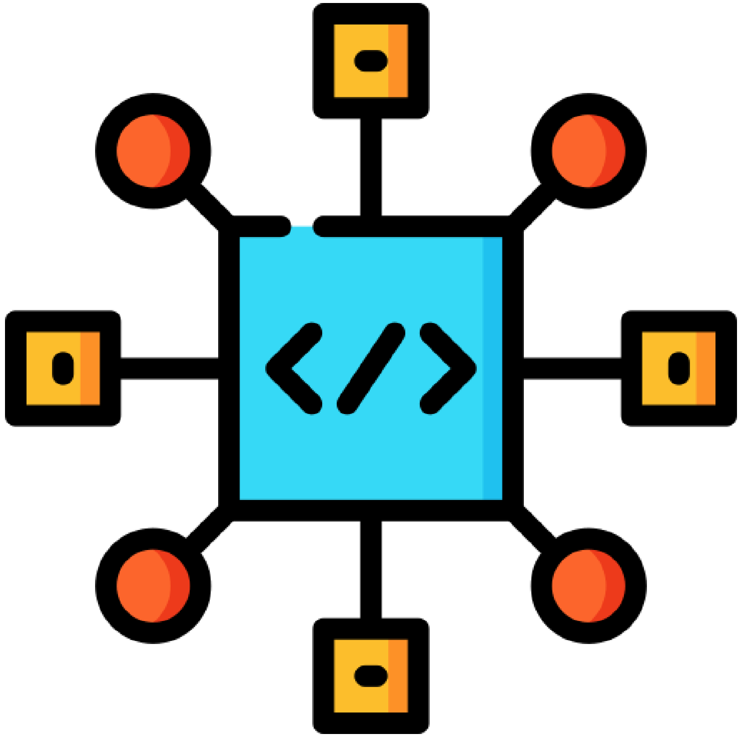


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

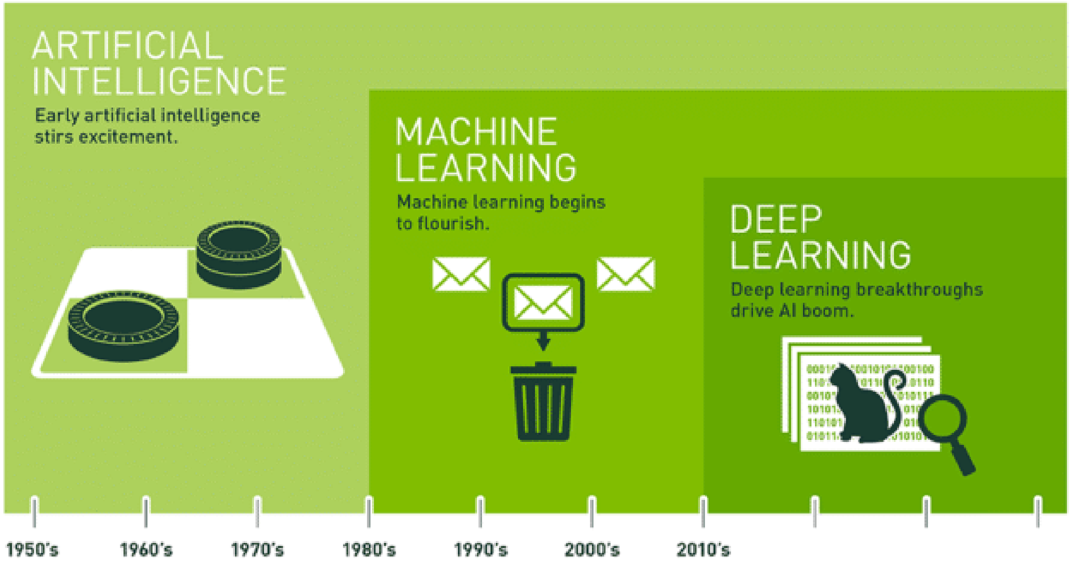
AI vs. Machine Learning vs. Deep Learning



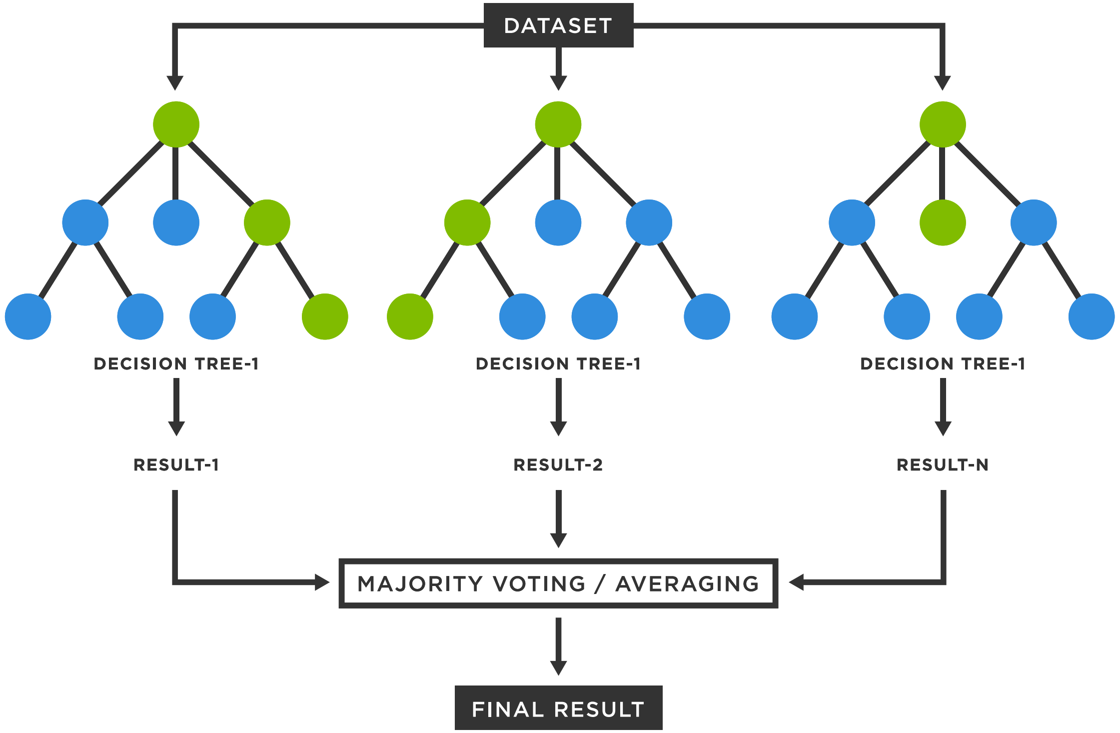
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.



AI vs. Machine Learning vs. Deep Learning

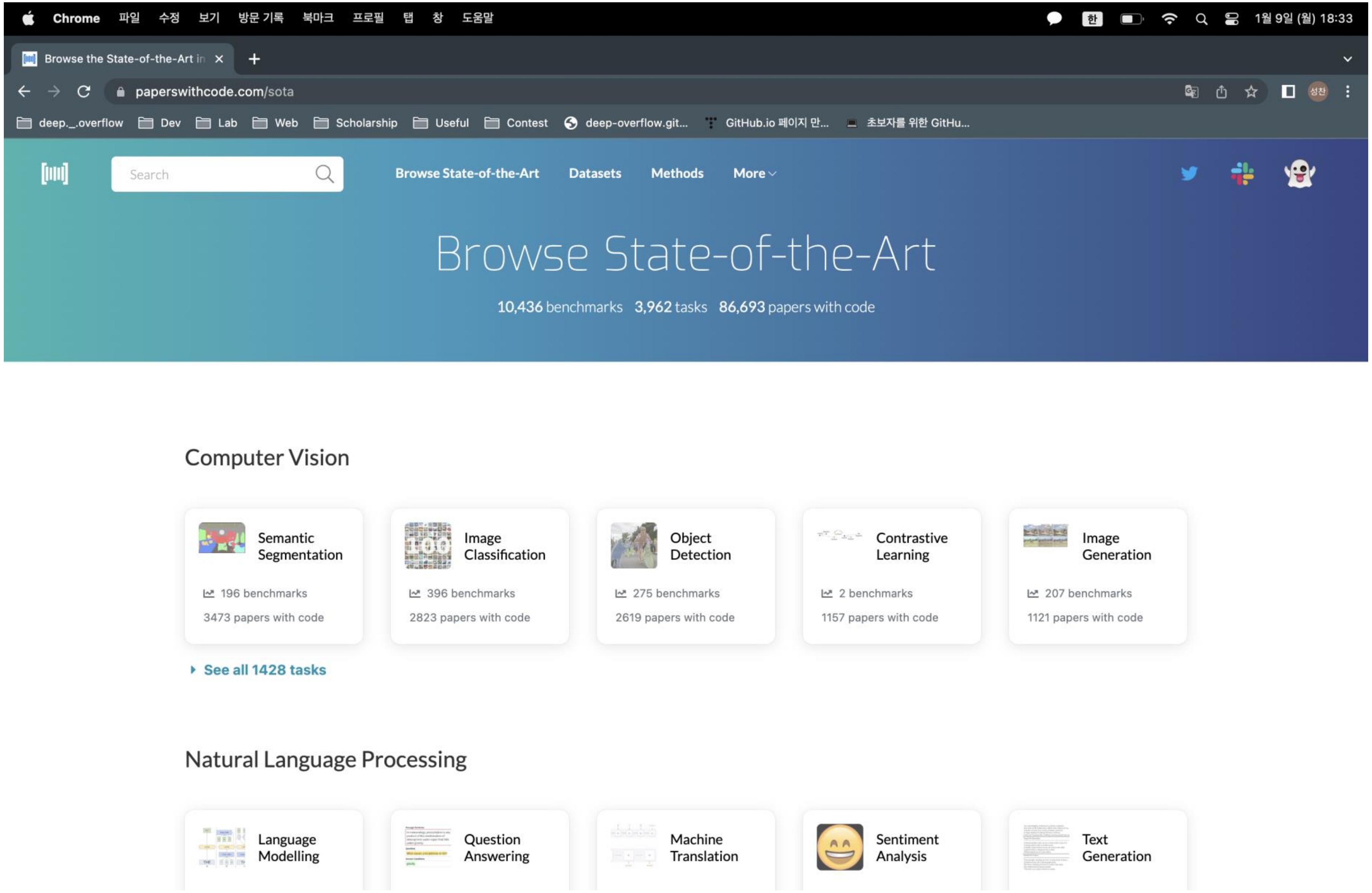


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.



What is Deep Learning?

Papers With Code: https://paperswithcode.com/

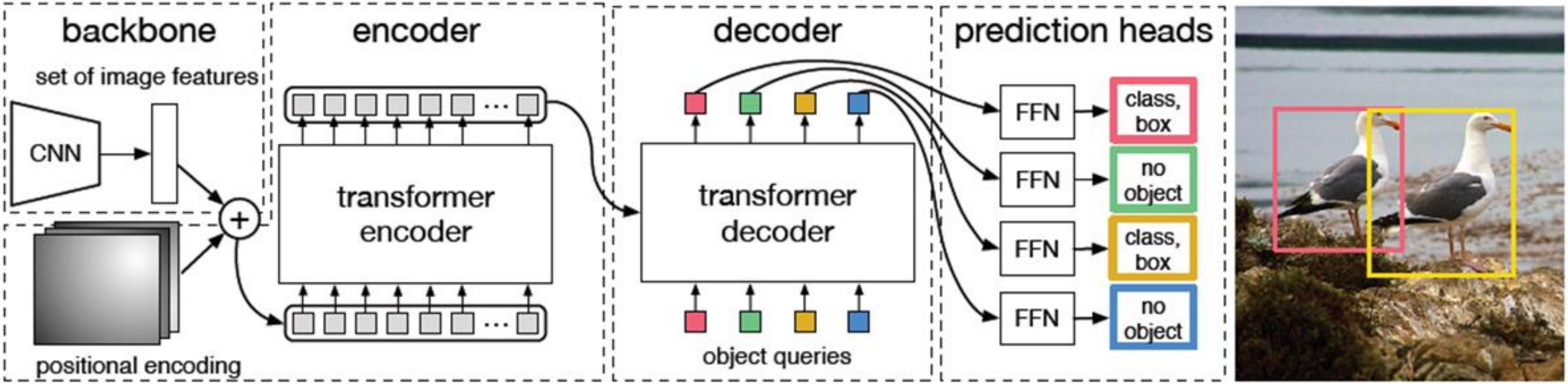


Deep Learning Component: Data, Model, Loss

Data

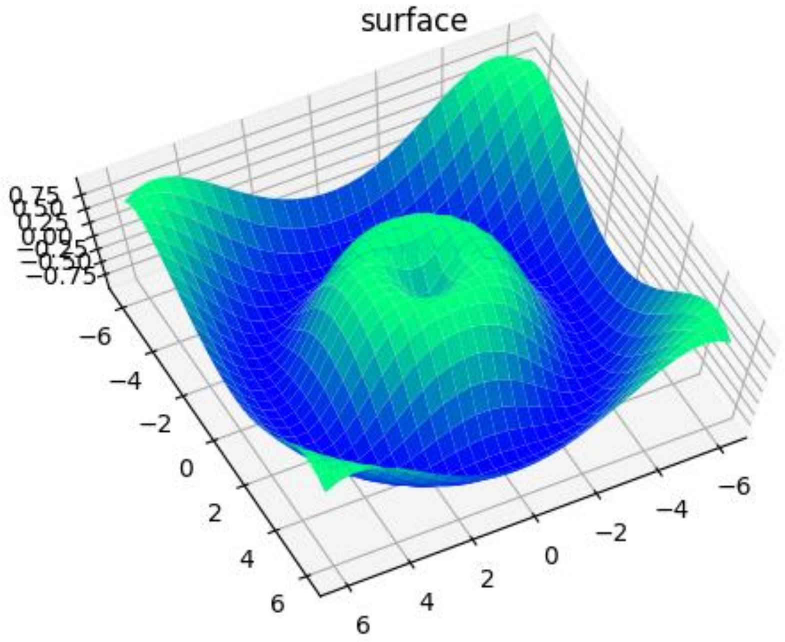
Deep Learning Component: Data, Model, Loss

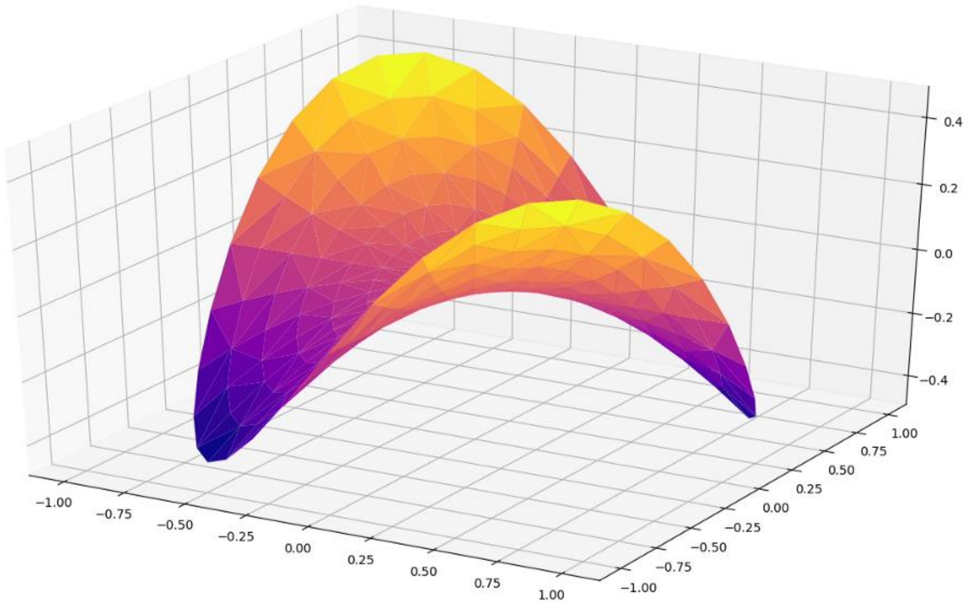
Model



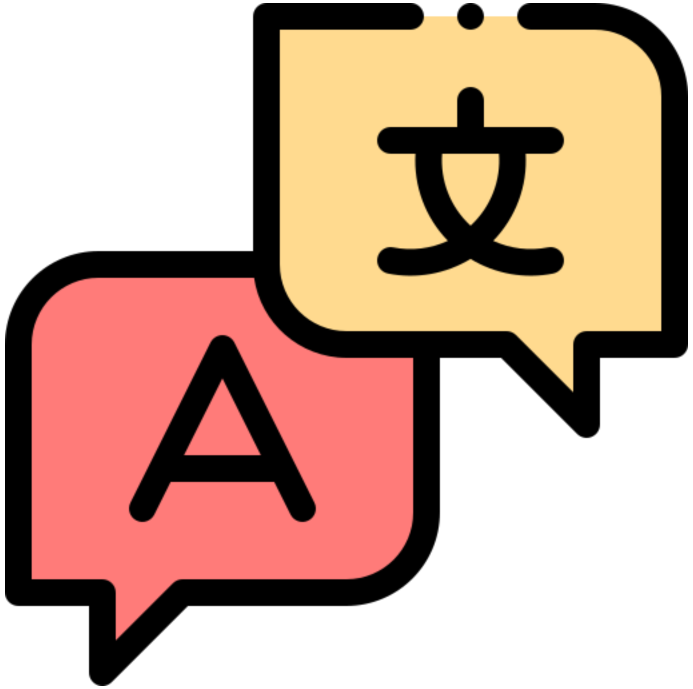
Deep Learning Component: Data, Model, Loss

Loss

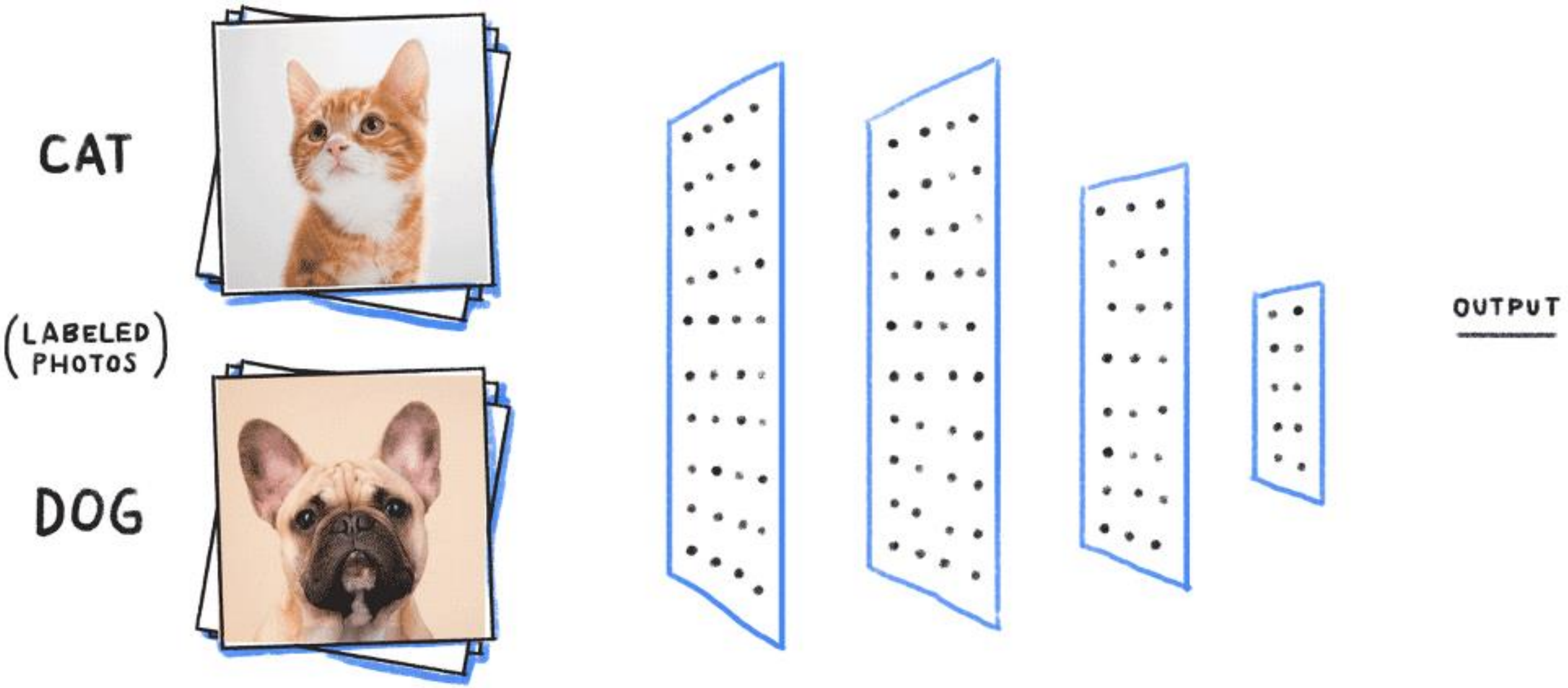




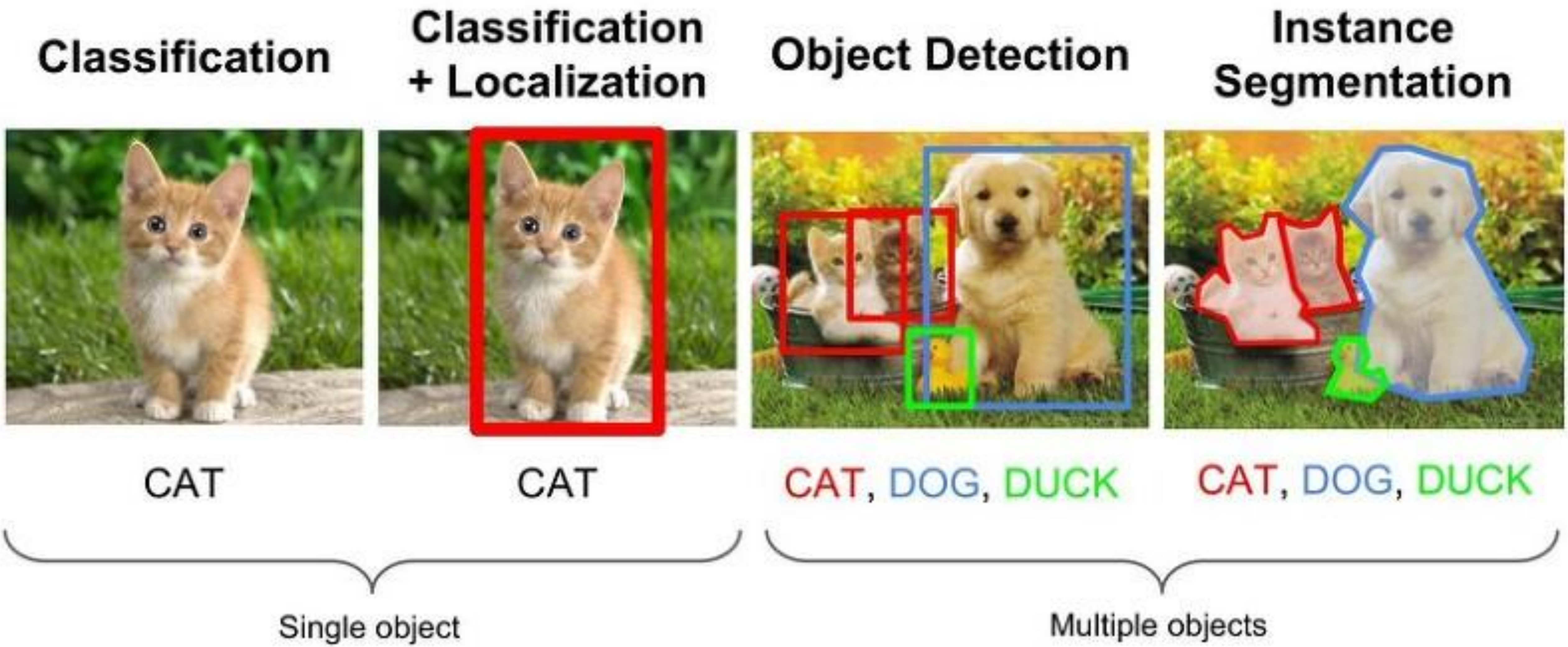
How to represent Data?



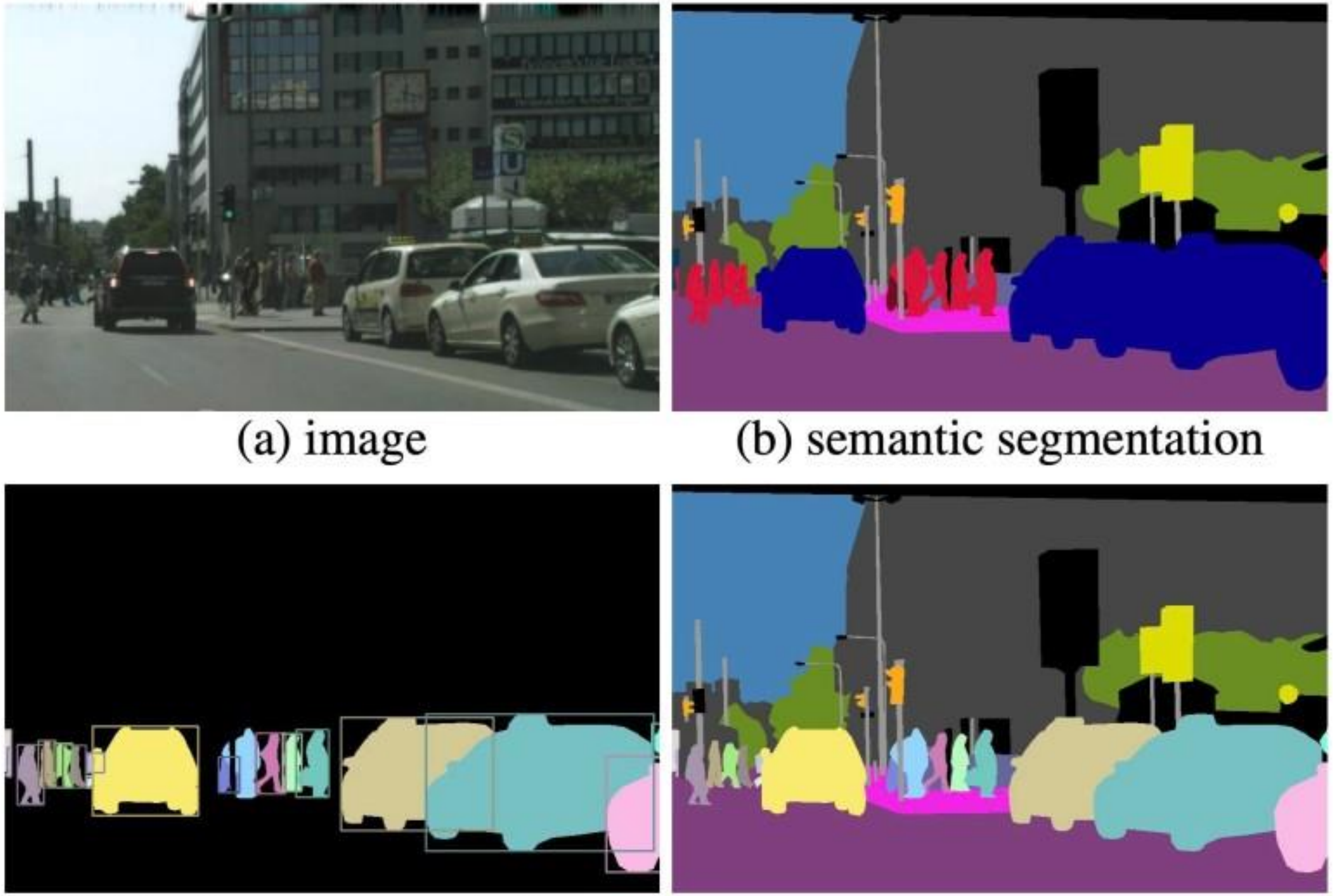
How to represent Data?



How to represent Data?



How to represent Data?



(a) image

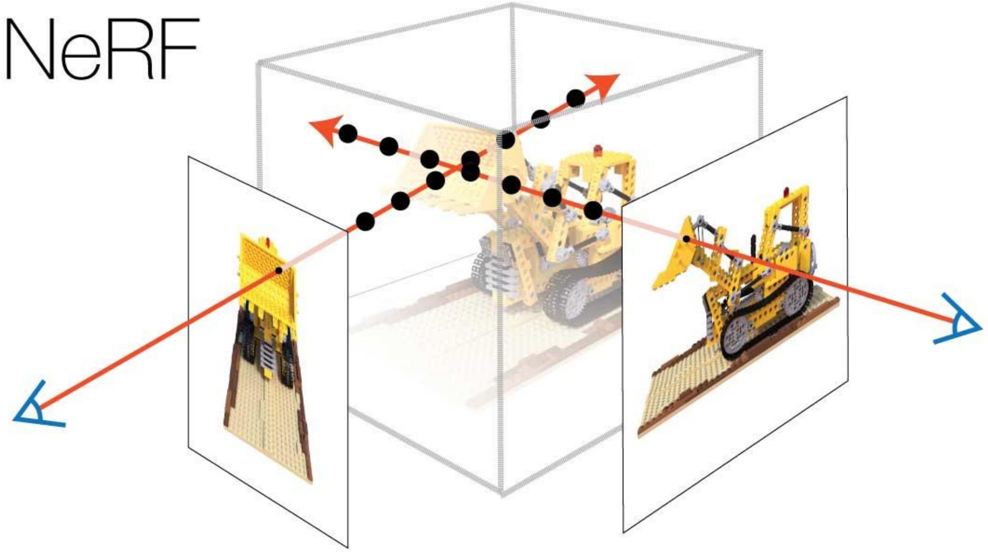
(b) semantic segmentation

(c) instance segmentation

(d) panoptic segmentation

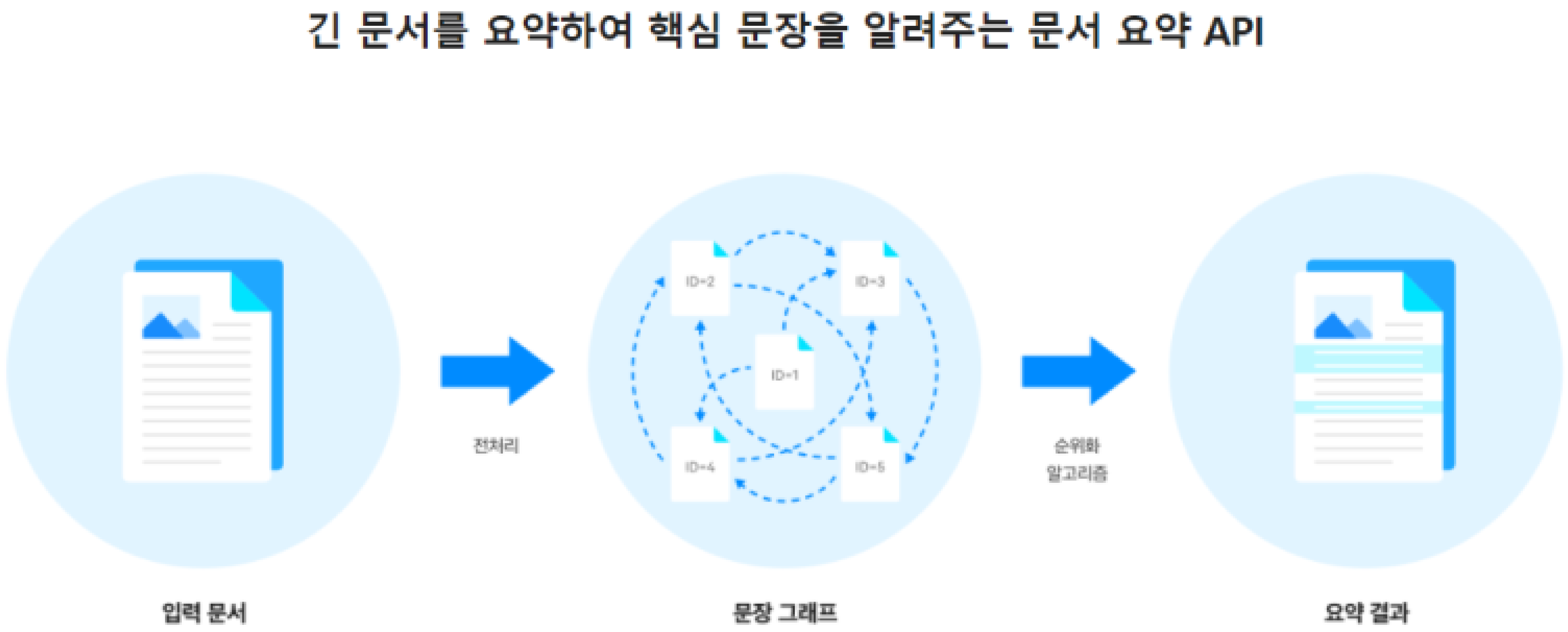
How to represent Data?

Neural Radiance Fields (NeRF)

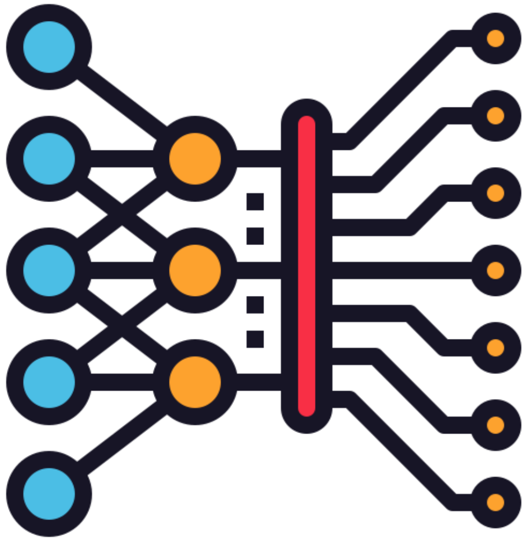


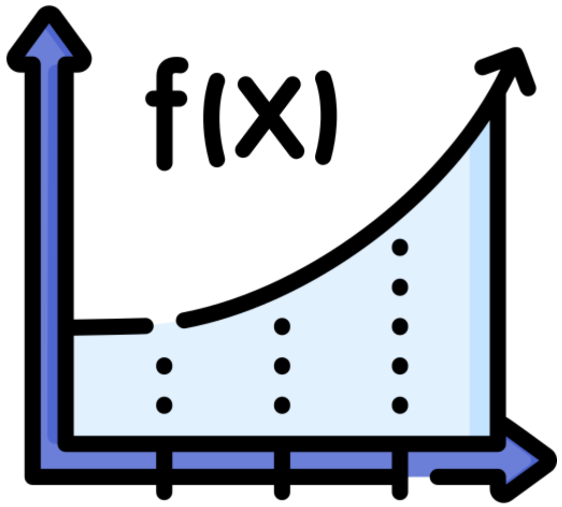
How to represent Data?

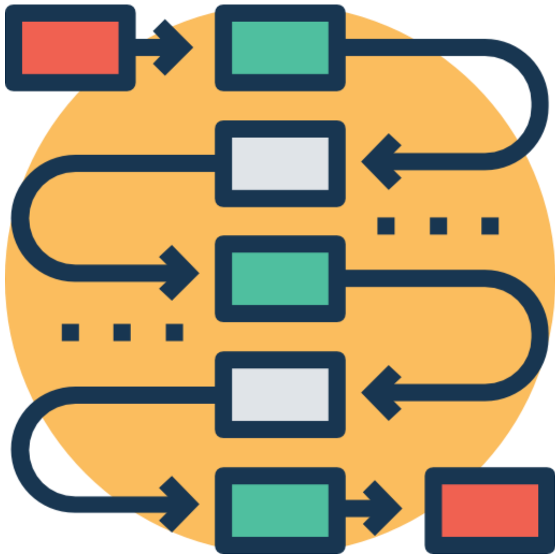
How to represent Data?

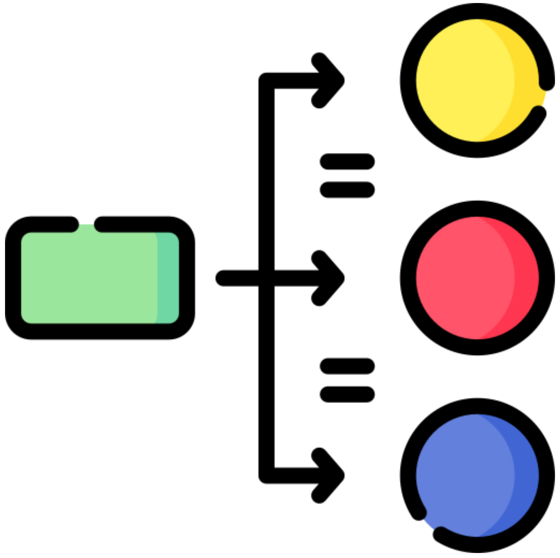


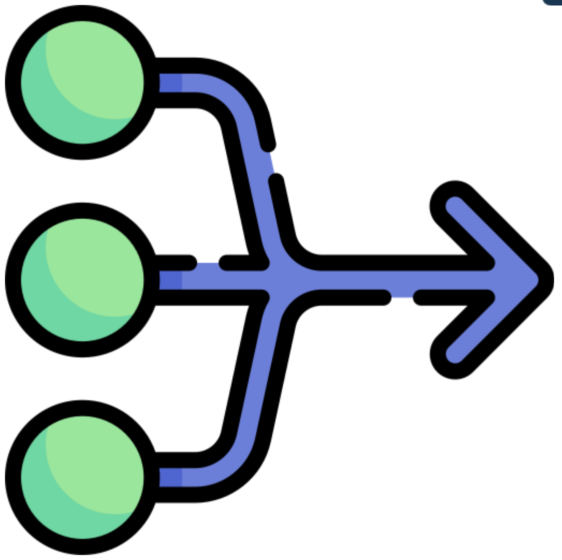
What is Model?









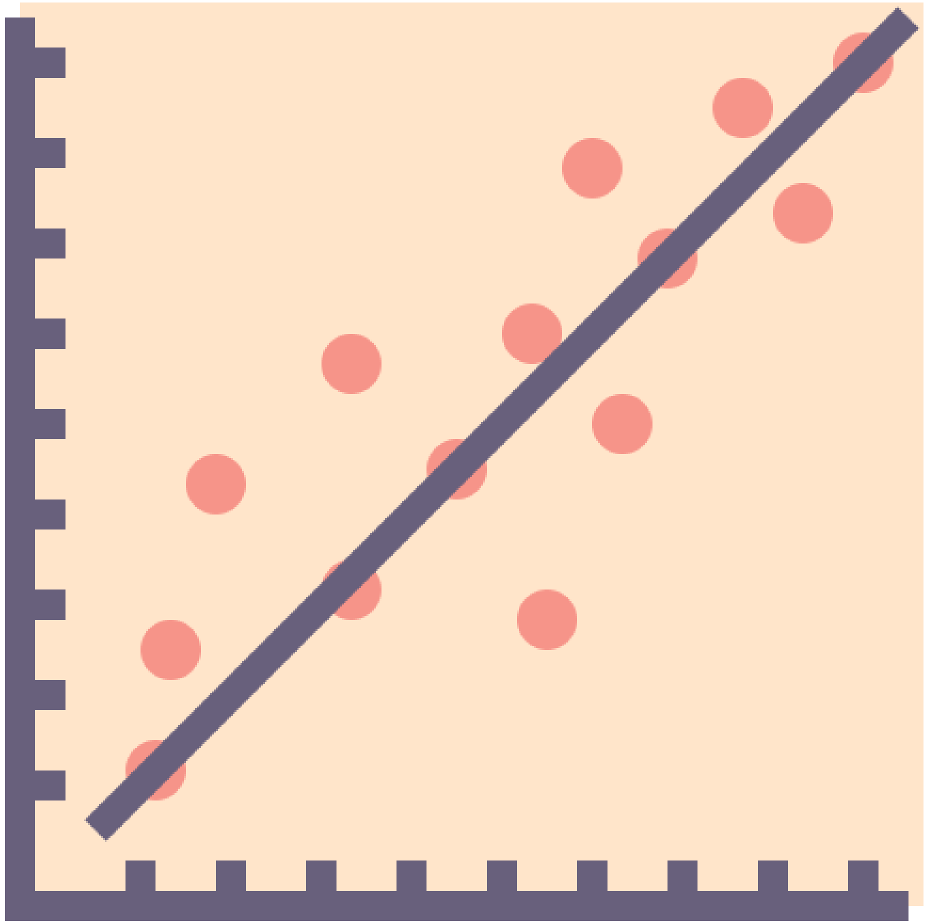


Linear Regression

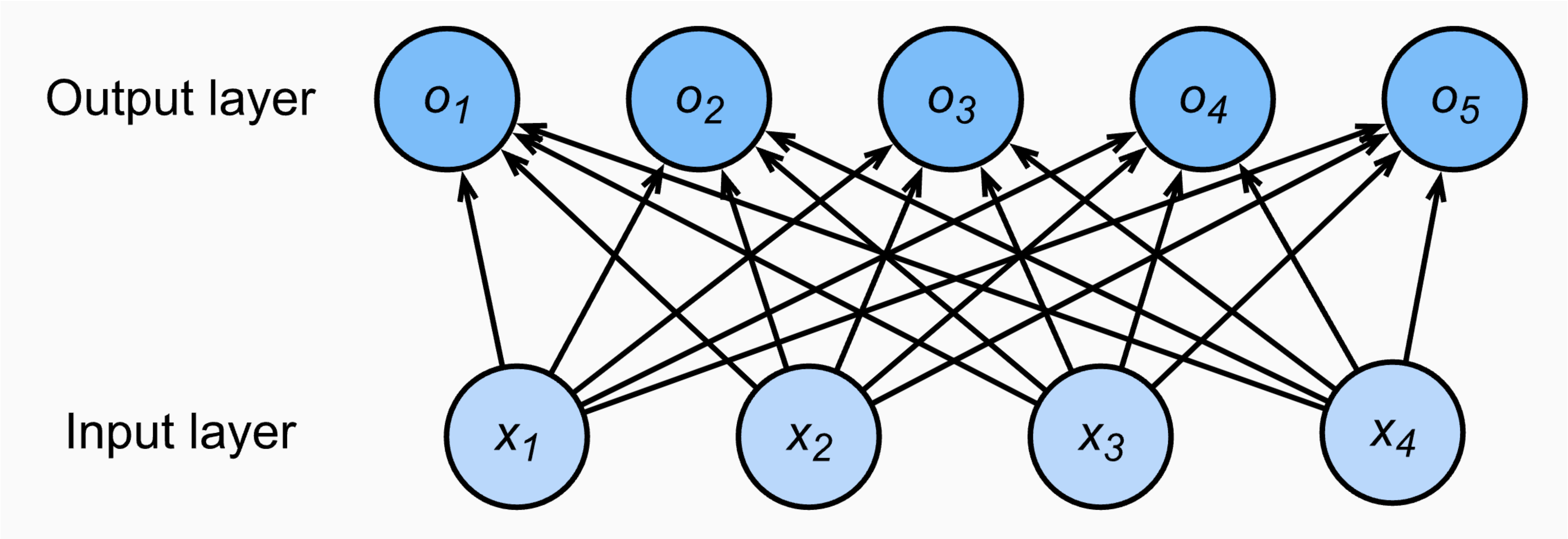
Regression은 Input Variables와 Output Variables의 관계를 통해

새로운 Input에 대한
Output을 예측하거나

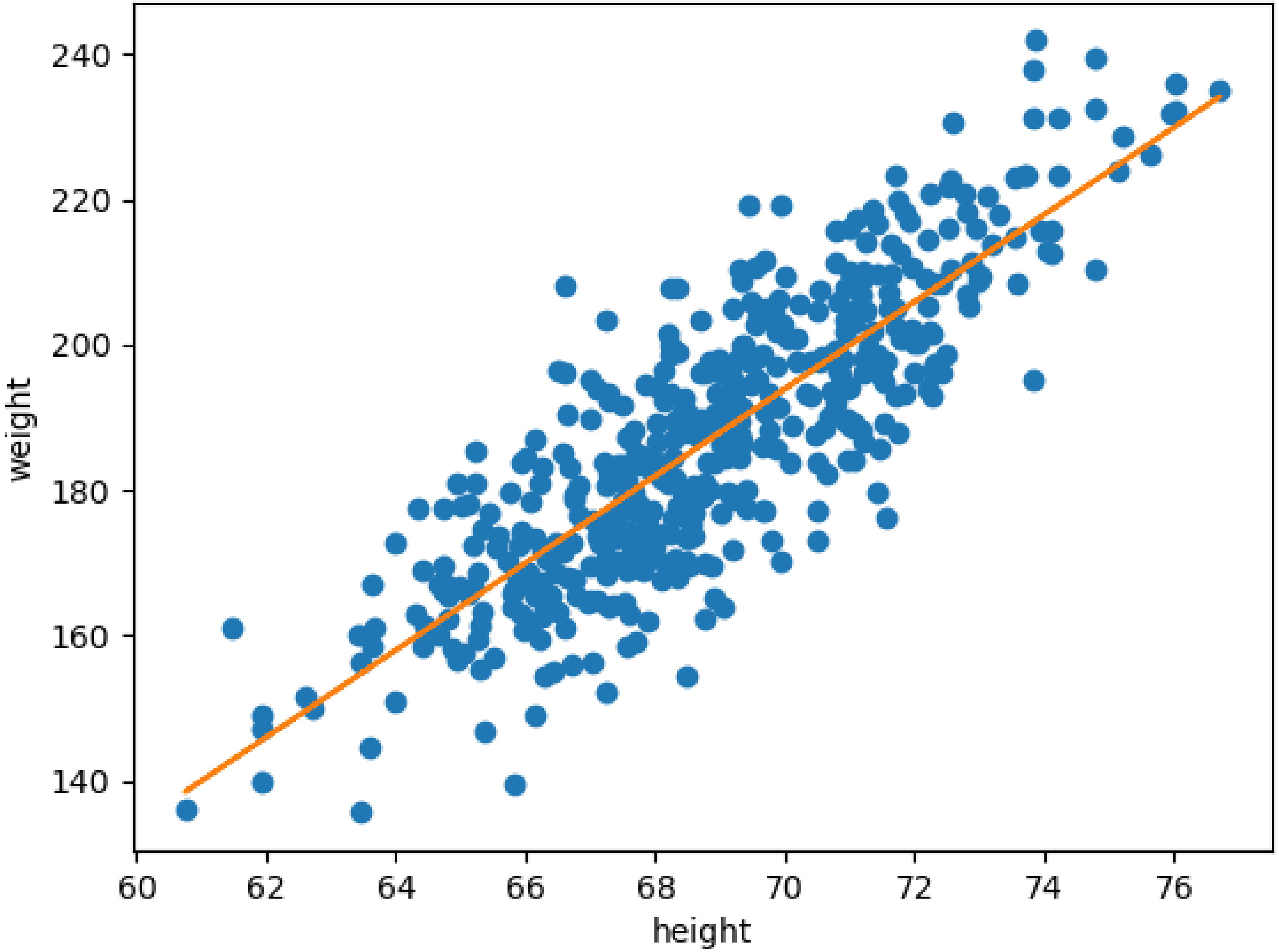
Output에 대한
Input의 영향을 이해할 수 있다.



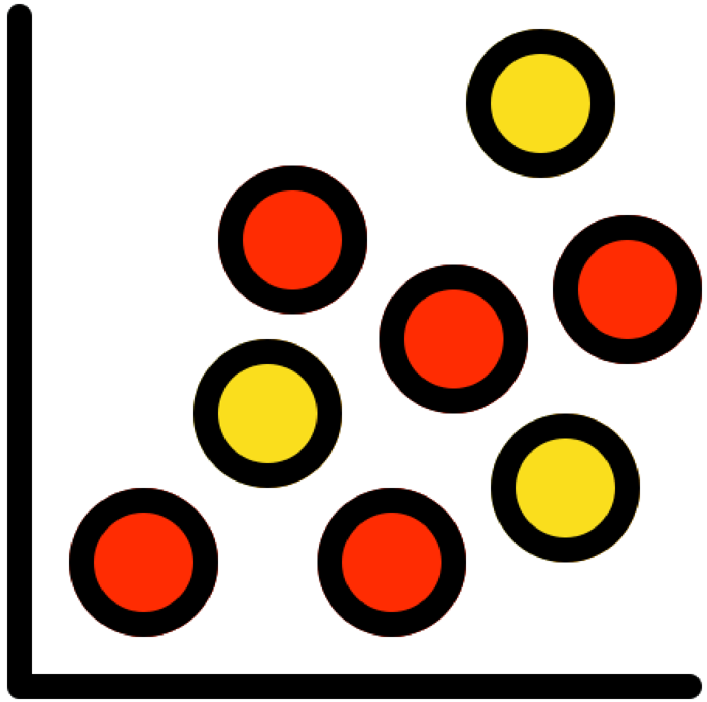
Linear Regression



Linear Regression



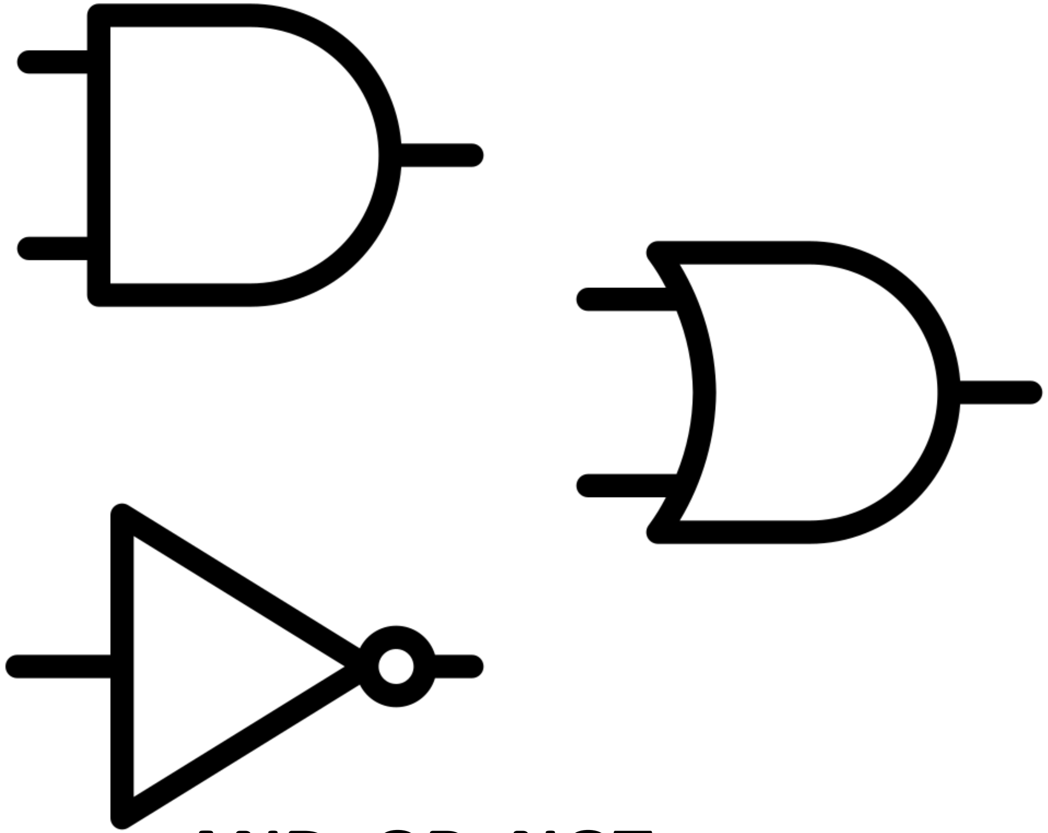
Linear Regression



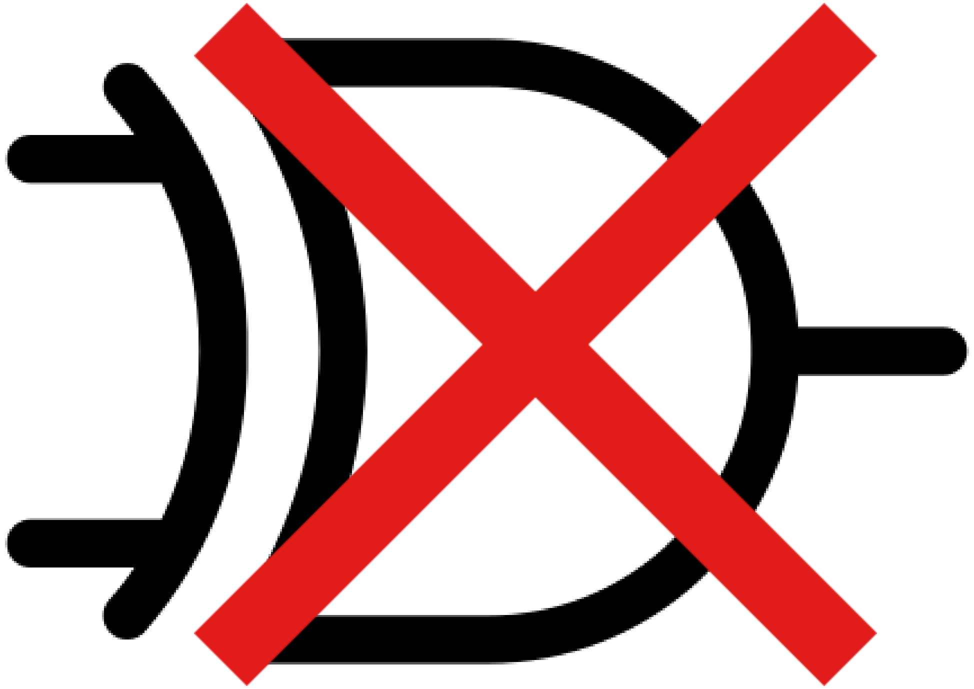
원인이 아니라 연관 관계

Linear Regression

Linear Regression의 한계

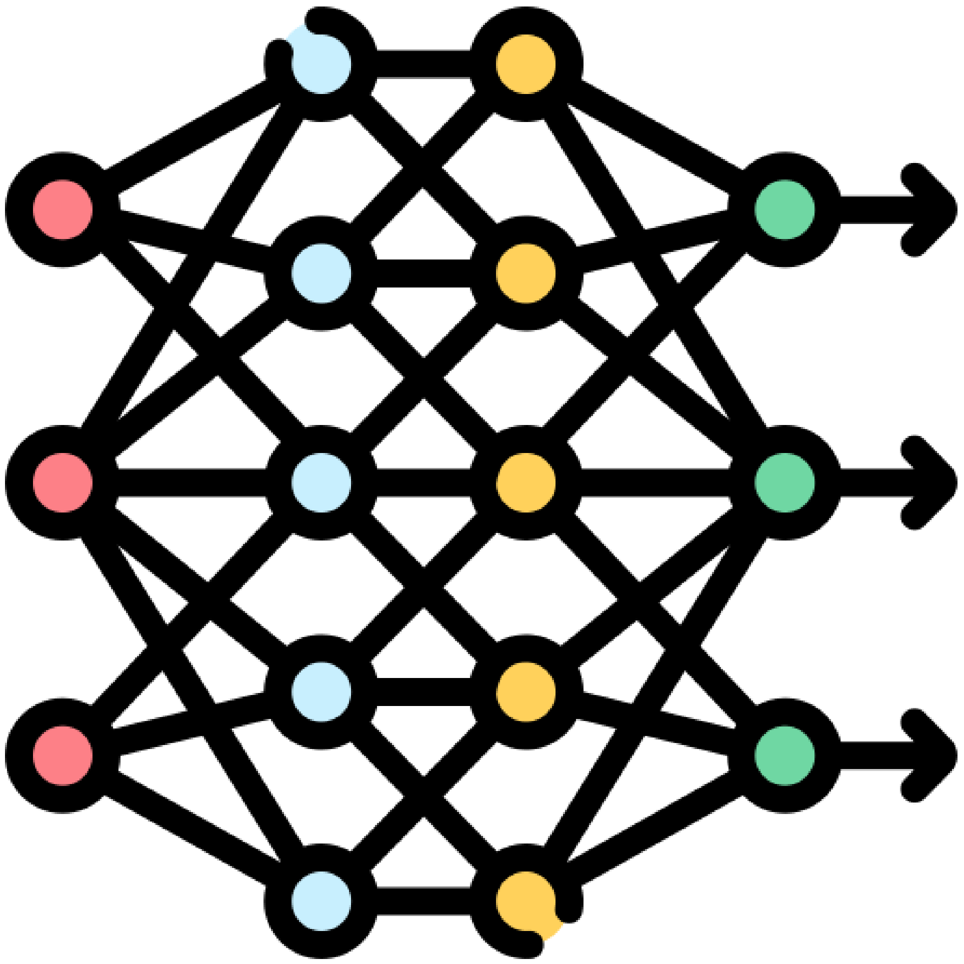


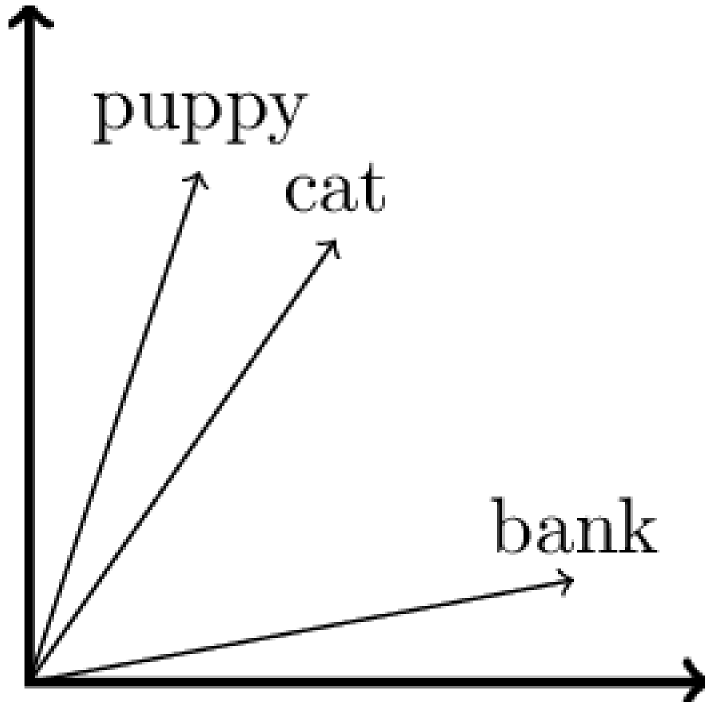
AND, OR, NOT



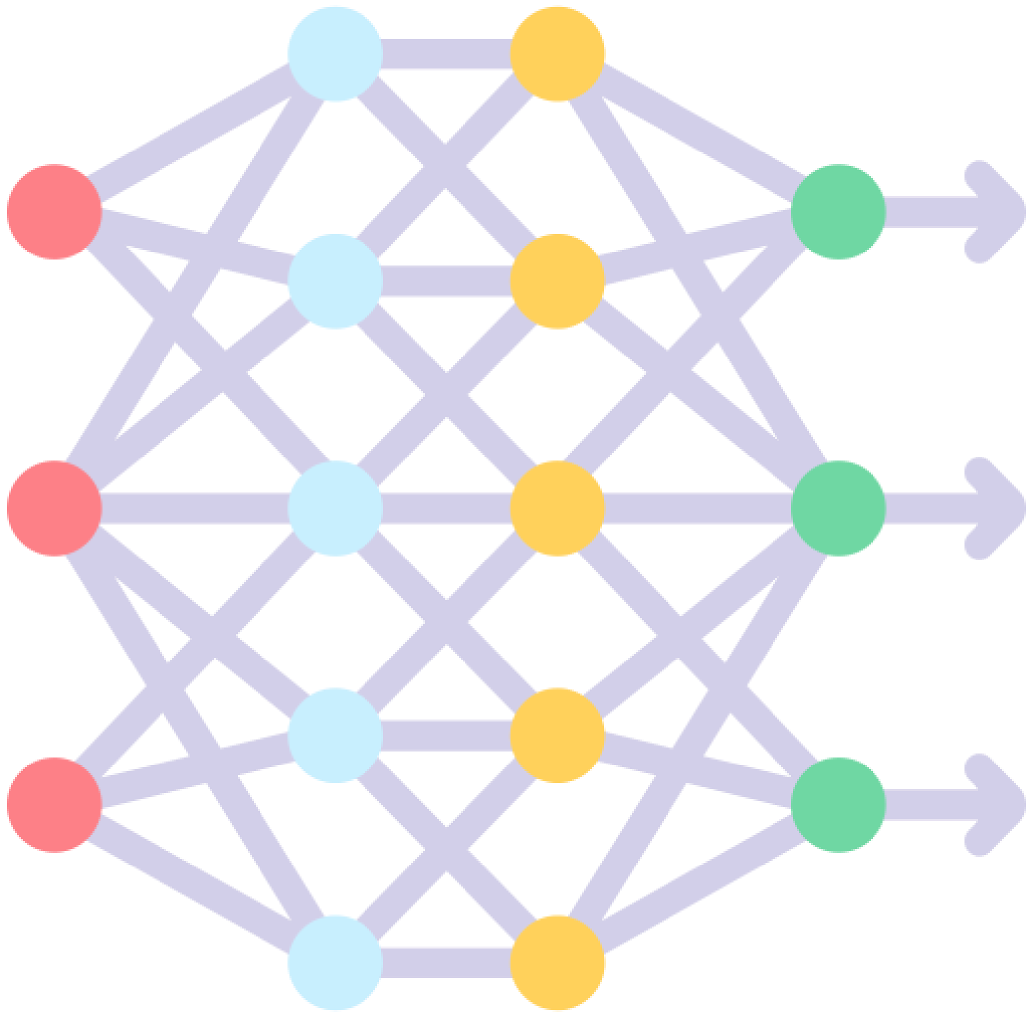
XOR

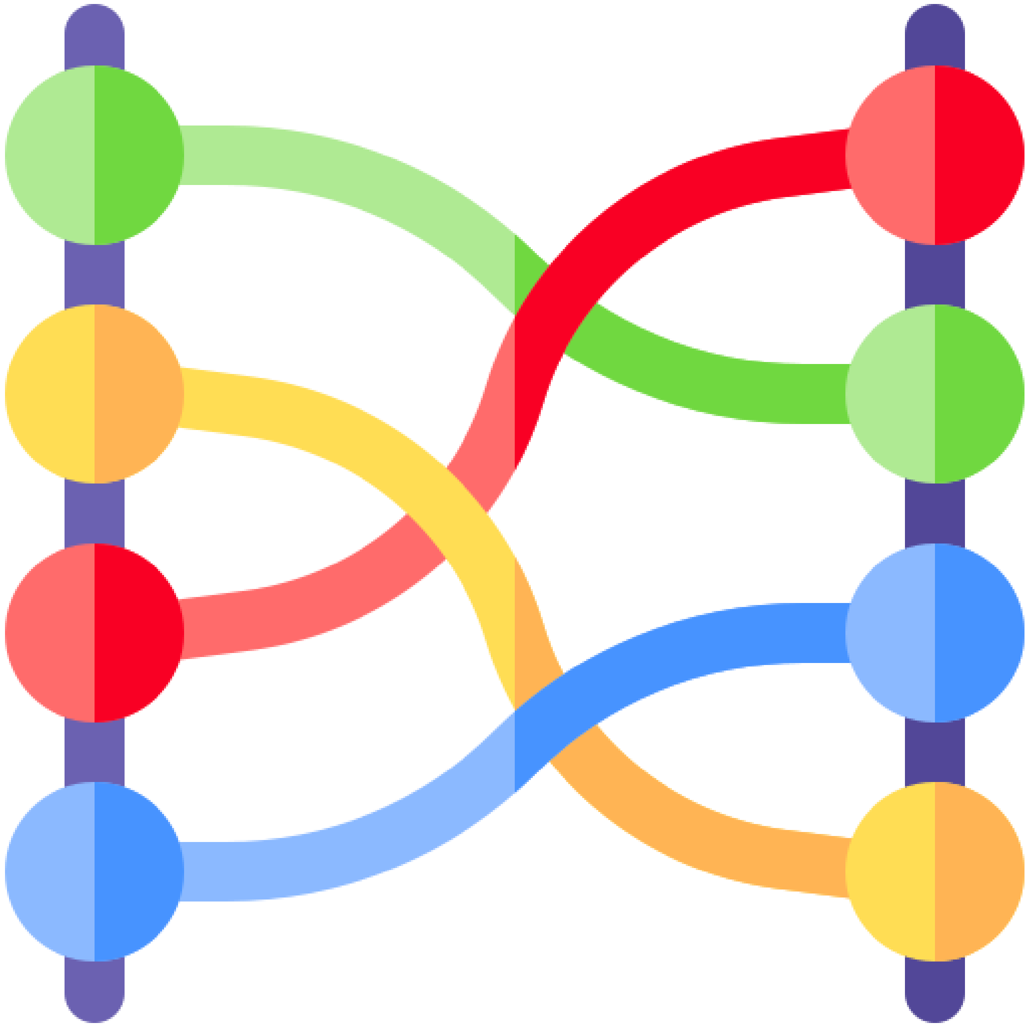
MLP





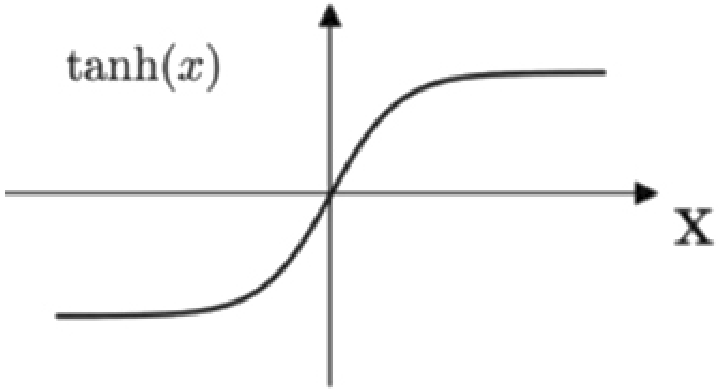
MLP



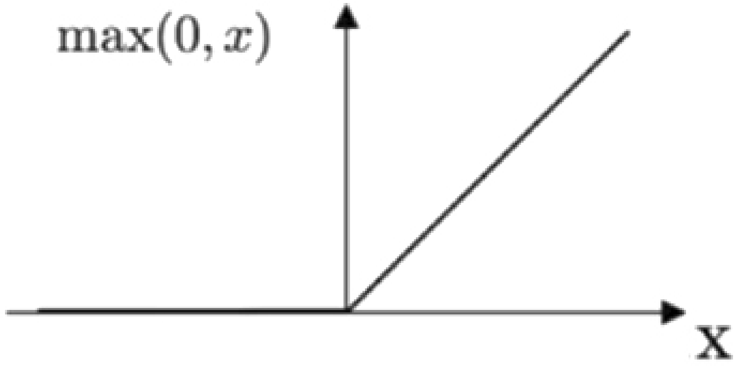


Activation

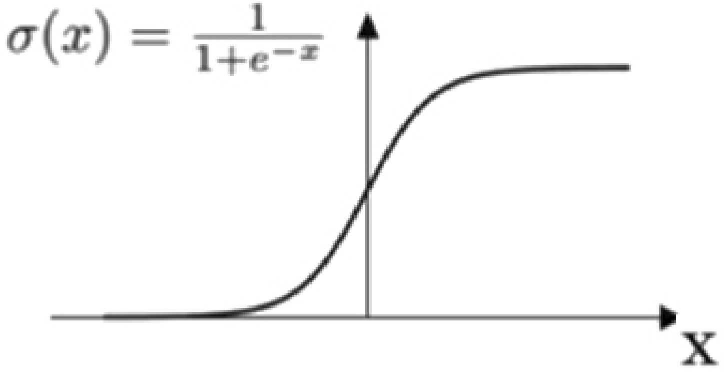
Tanh



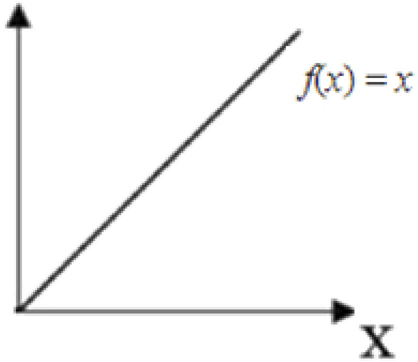
ReLU



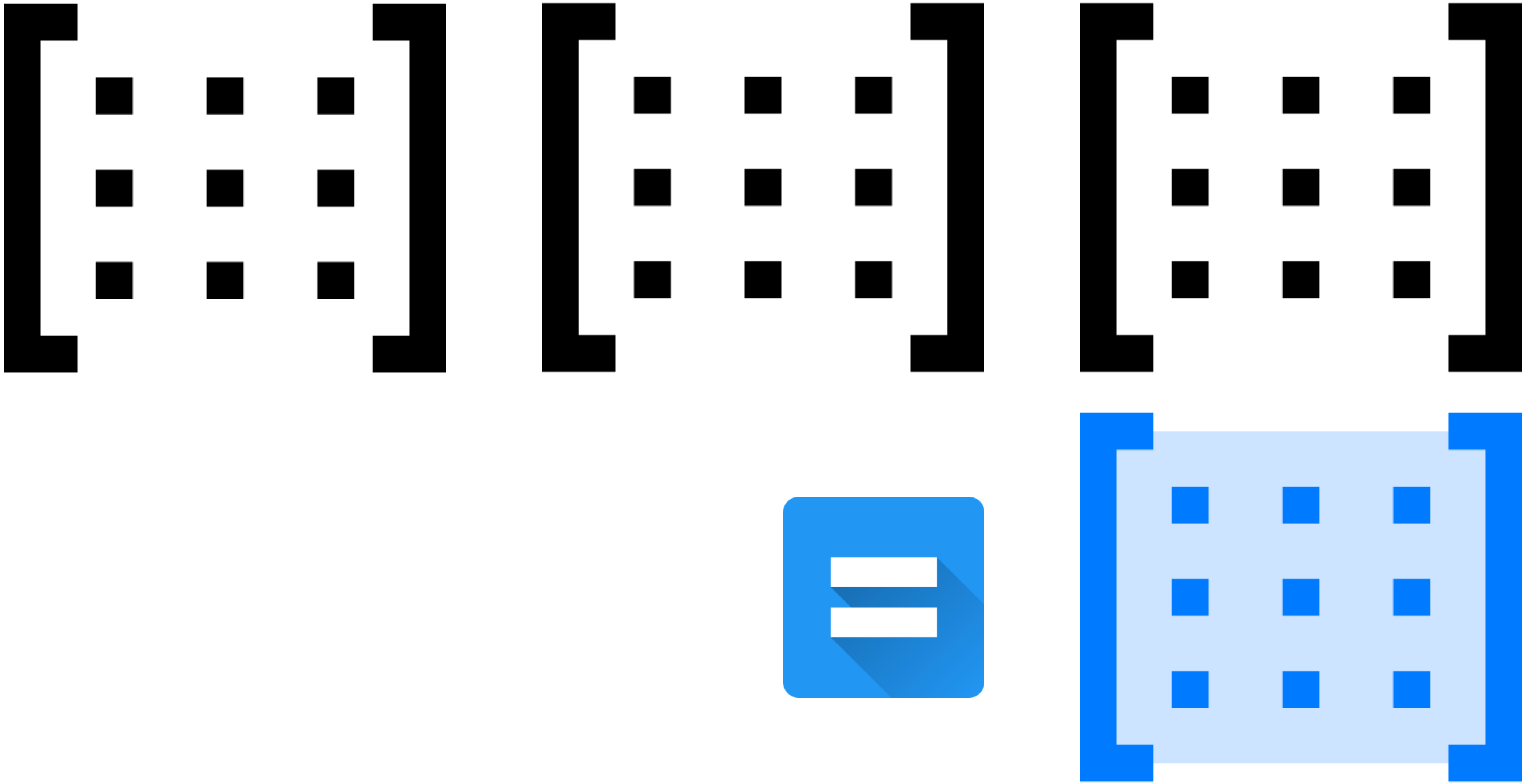
Sigmoid



Linear



Activation



Part 2. Learning

Seongchan Kim

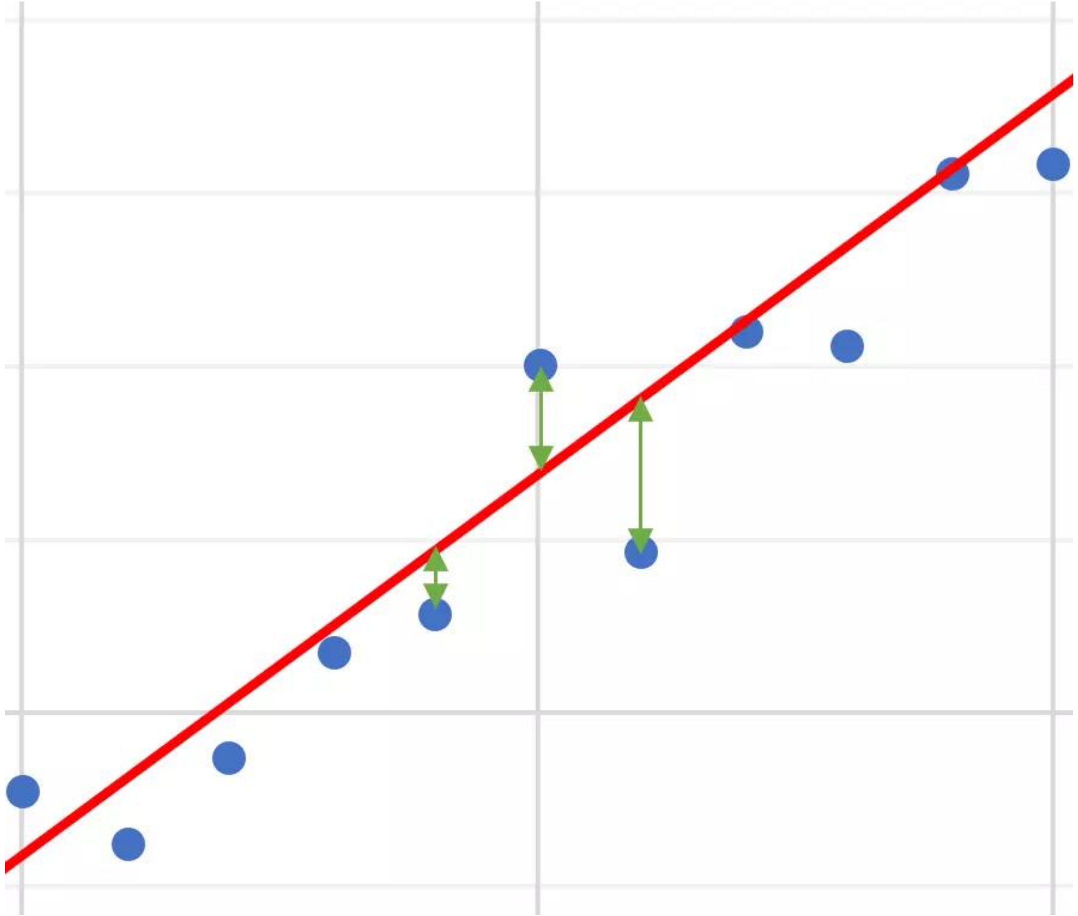
2020320120@korea.ac.kr

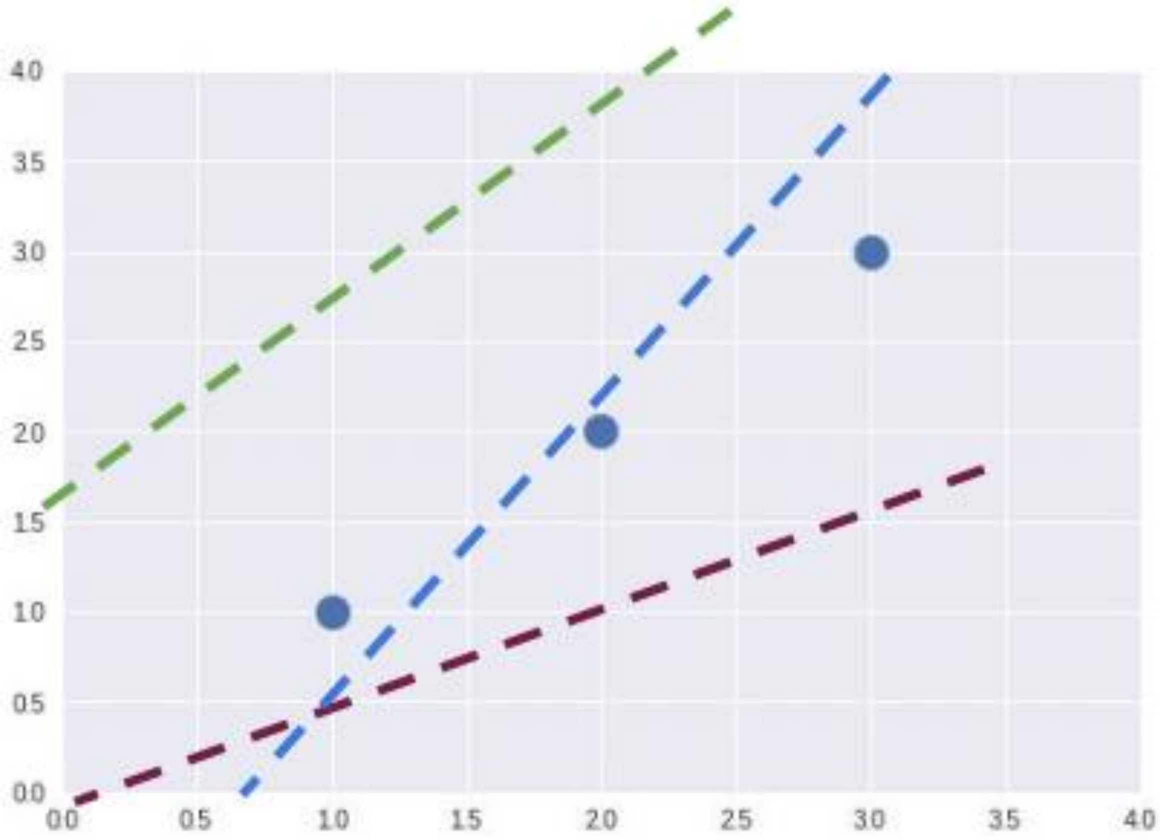
Artificial Intelligence in KU (AIKU)

Department of Computer Science and Engineering, Korea University

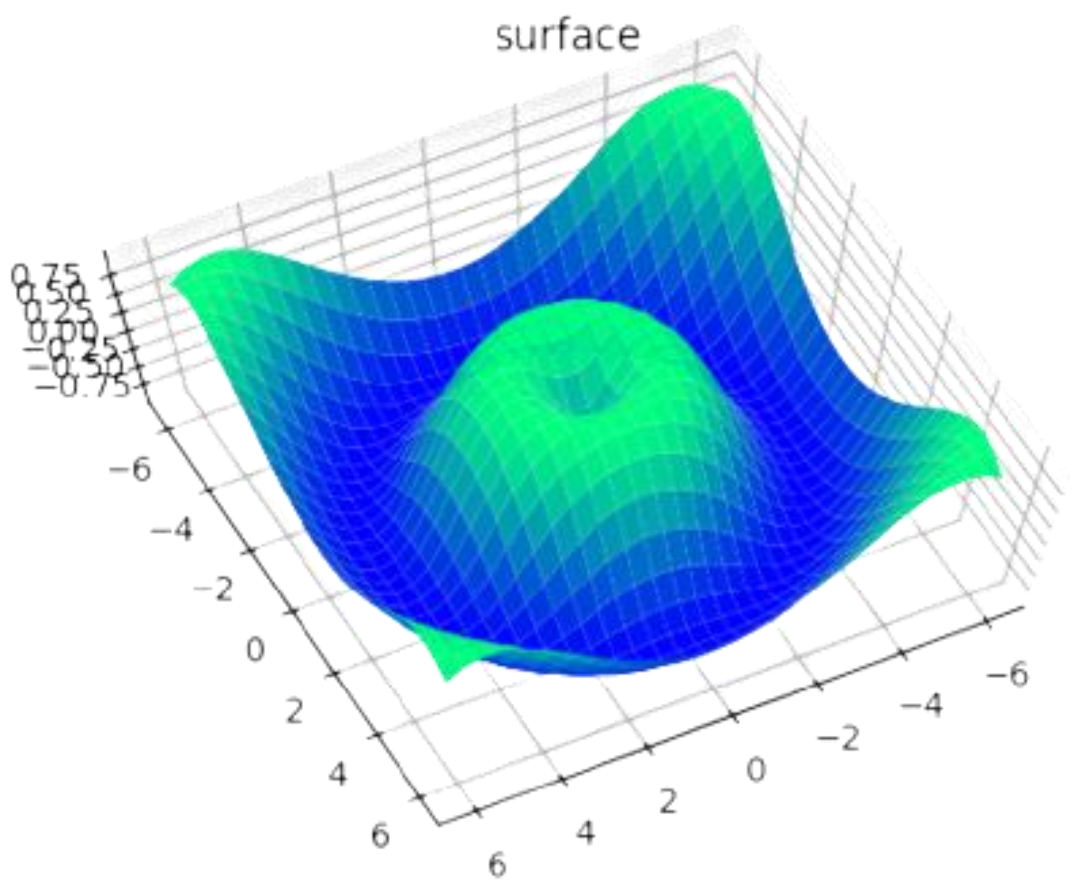
AIKU

What is Loss?

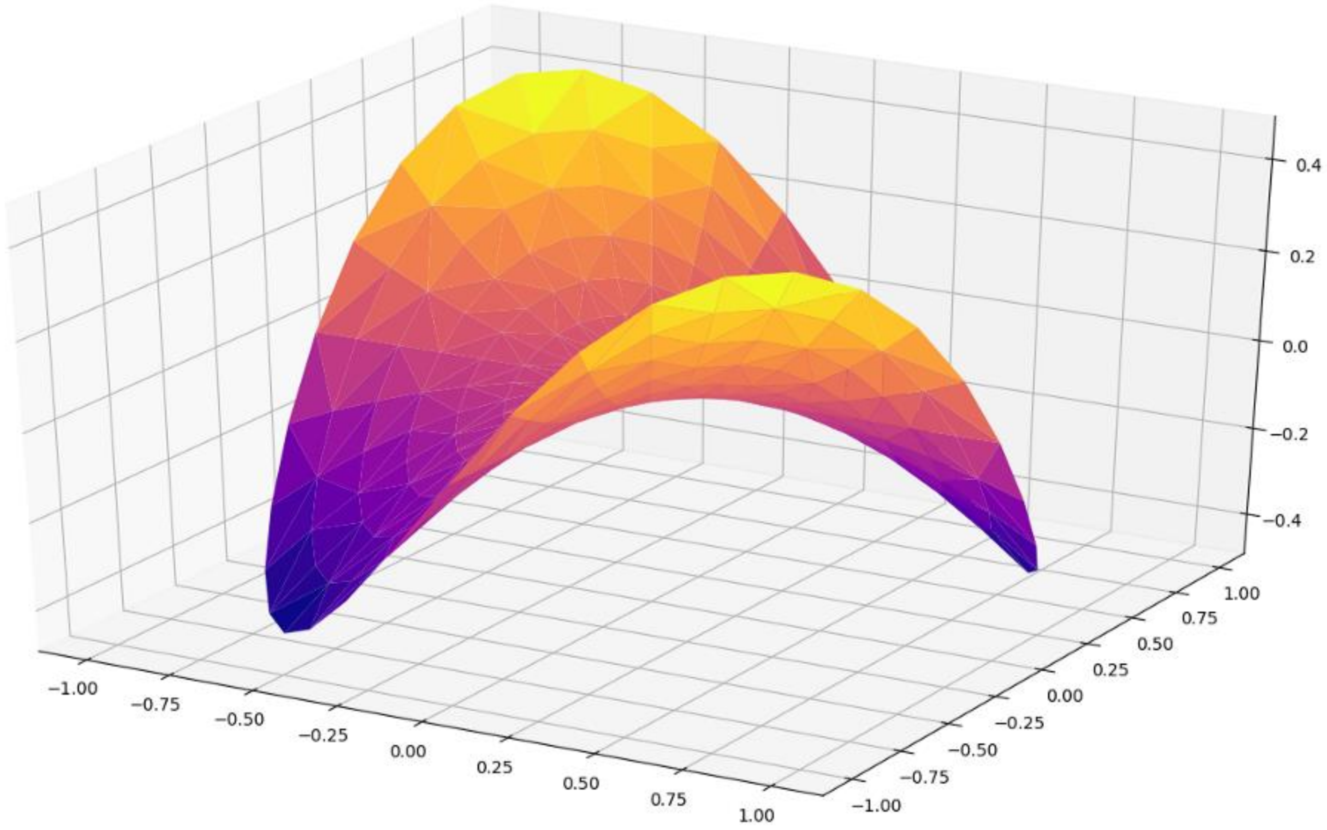




What is Loss?

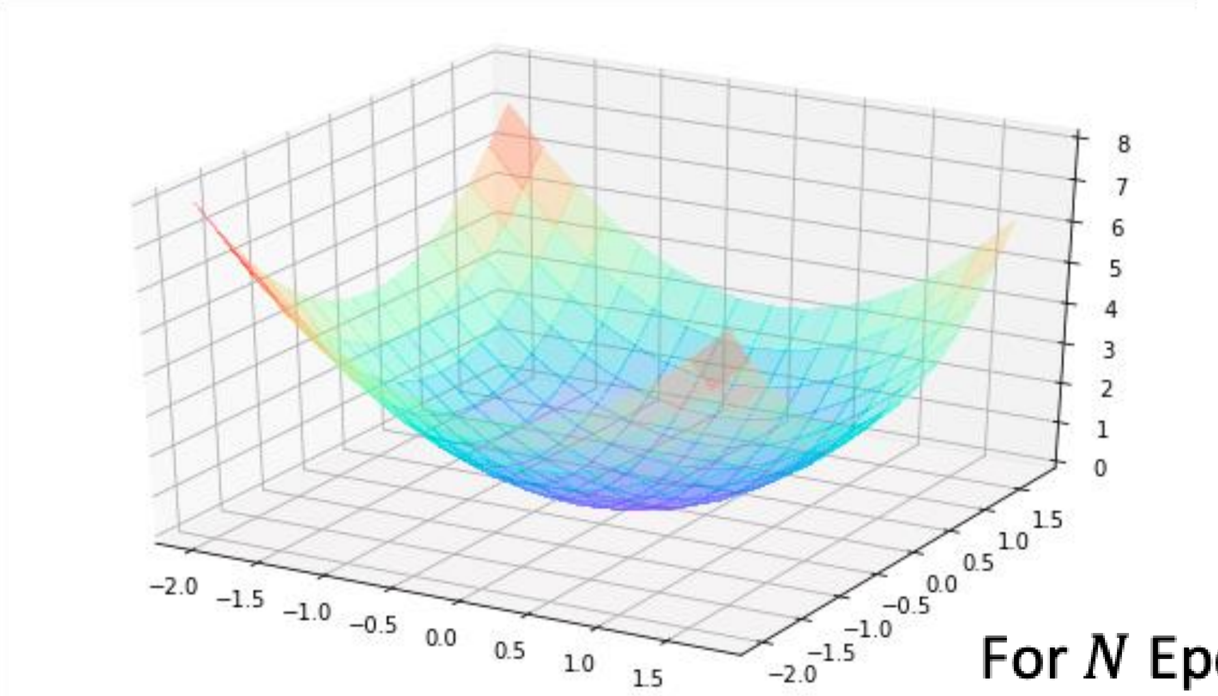


surface
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−6
−4
−2
0
2
4
6



0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

What is Loss?



For N Epochs{

For each training batch $\{(\mathrm{x}_b, y_b)\}_{b=1}^{B}$ {

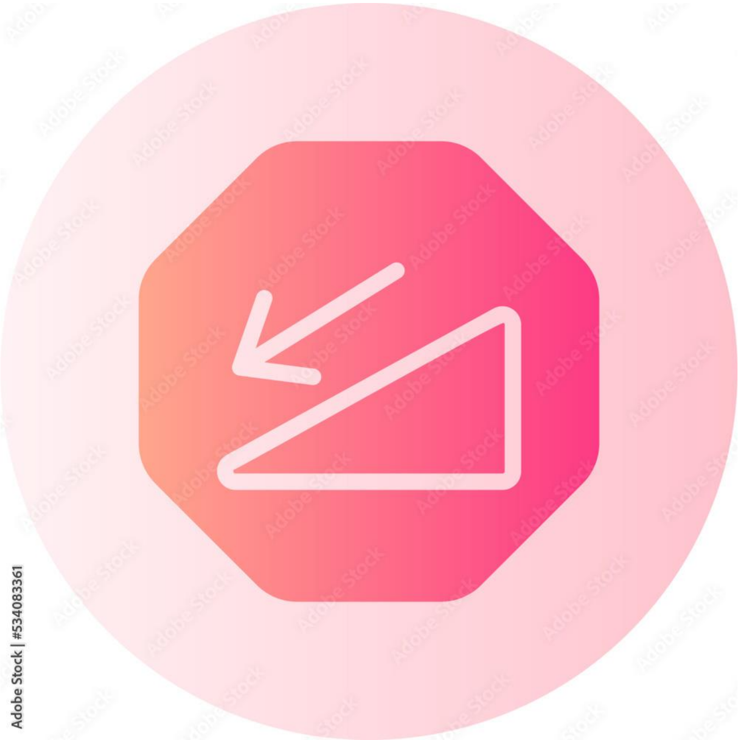
$$\mathrm{w} \leftarrow \mathrm{w} - \alpha \frac{\partial}{\partial \mathrm{w}} L(\mathrm{w})$$

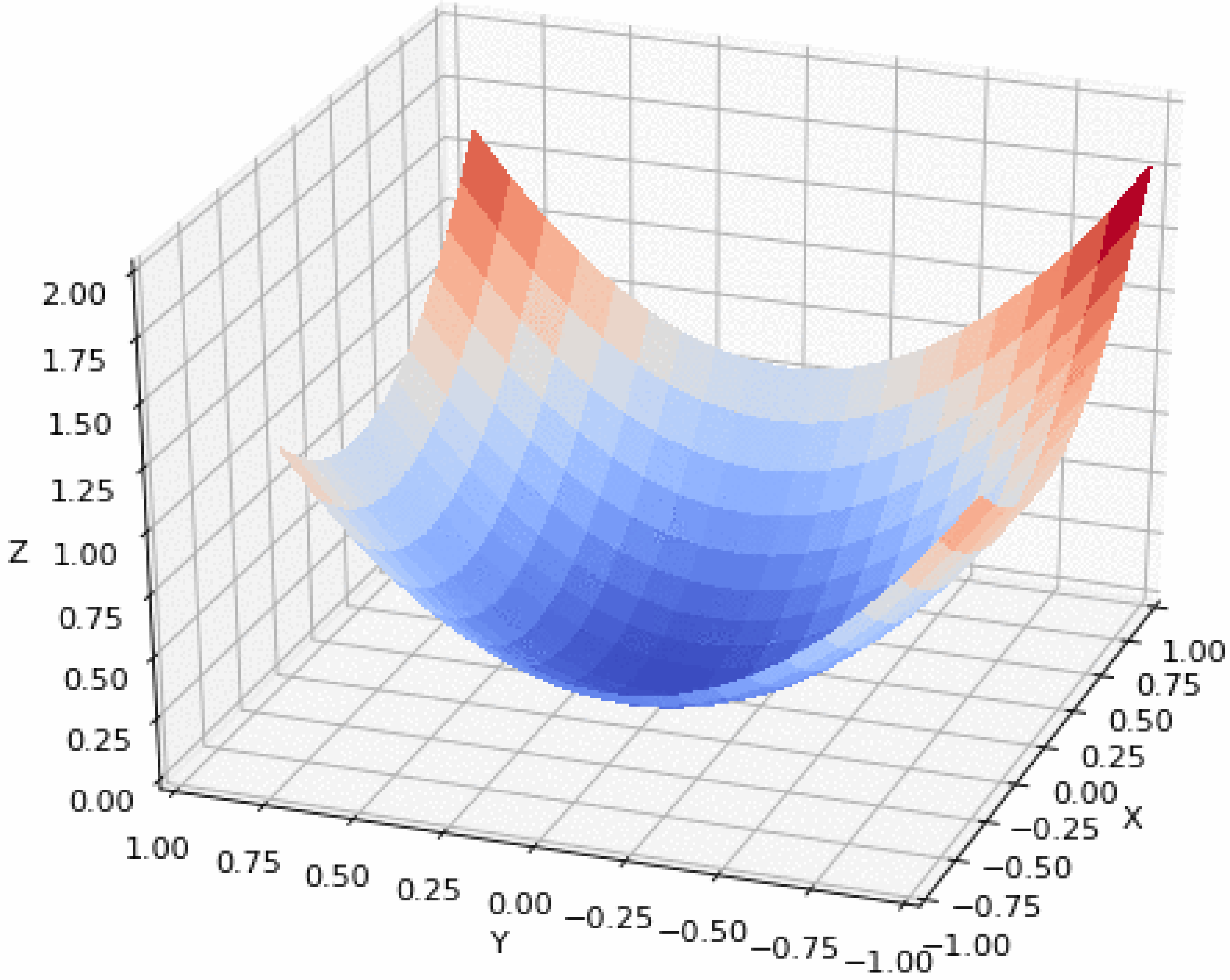
}

}

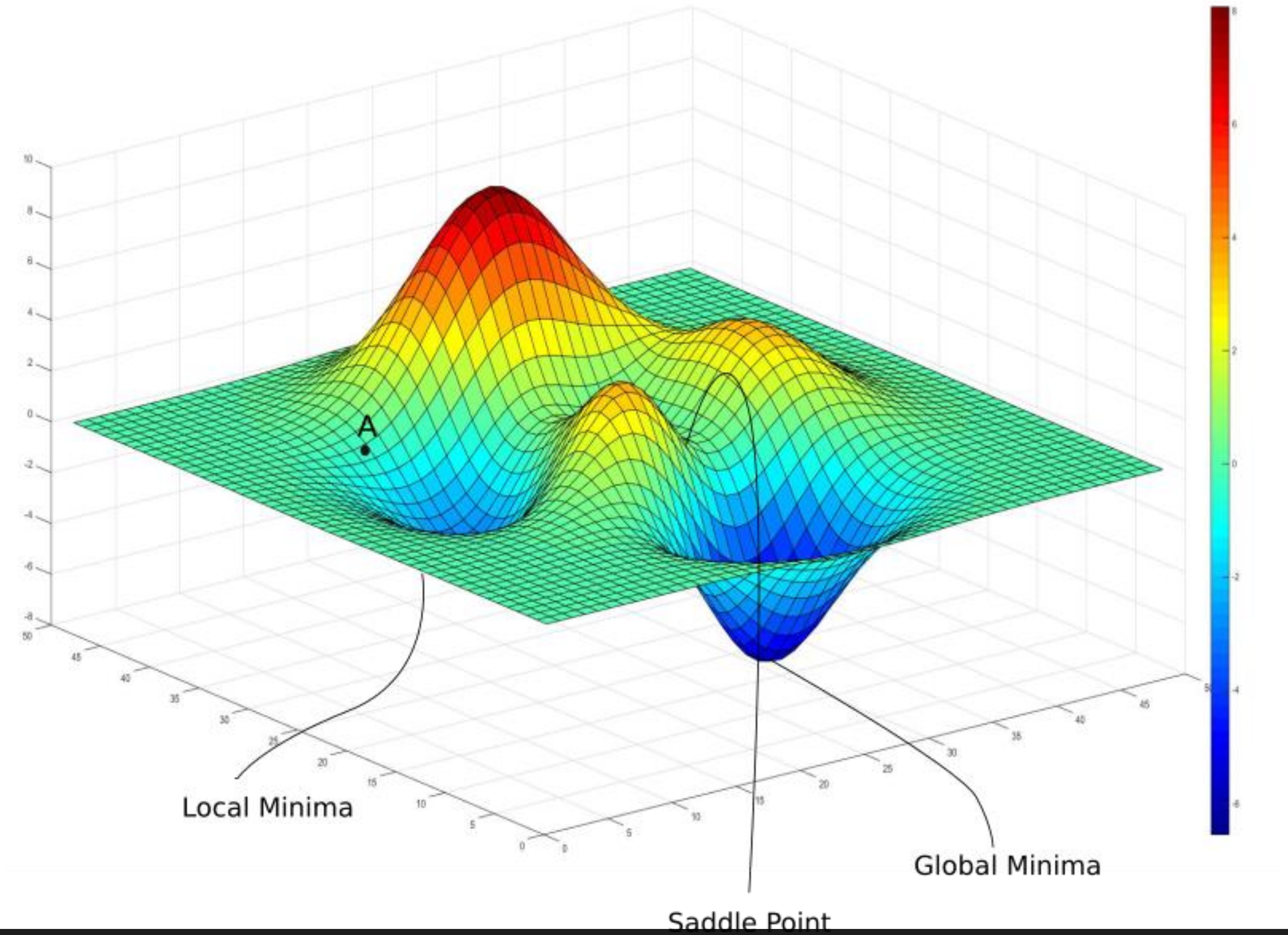
Gradient Descent

손실을 어떻게 감소시킬 것인가?



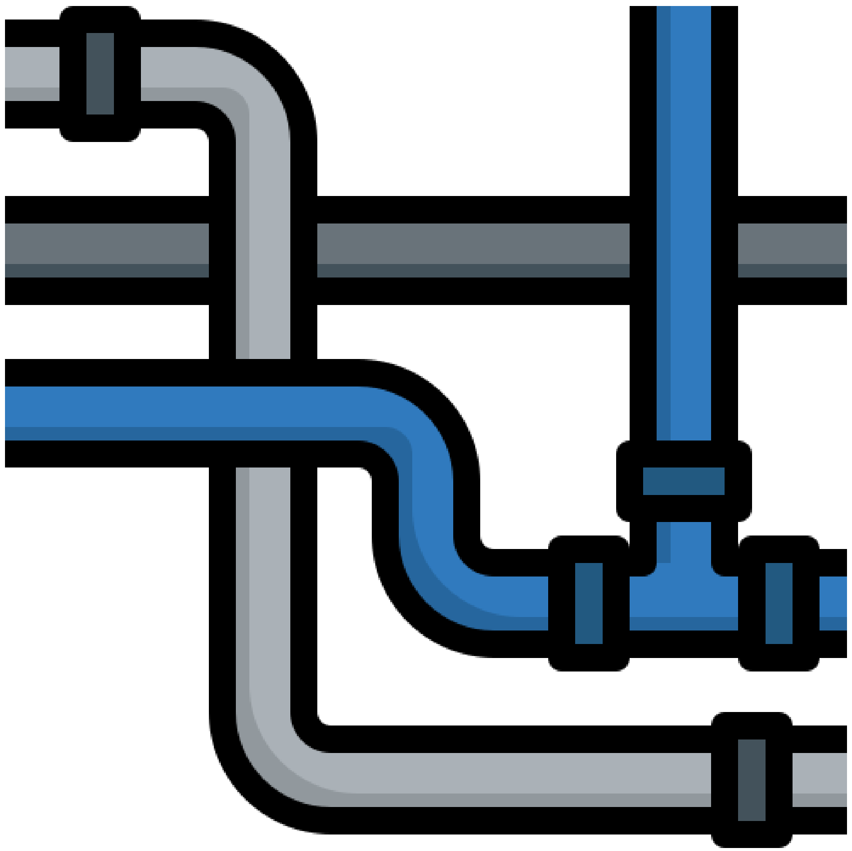


Gradient Descent



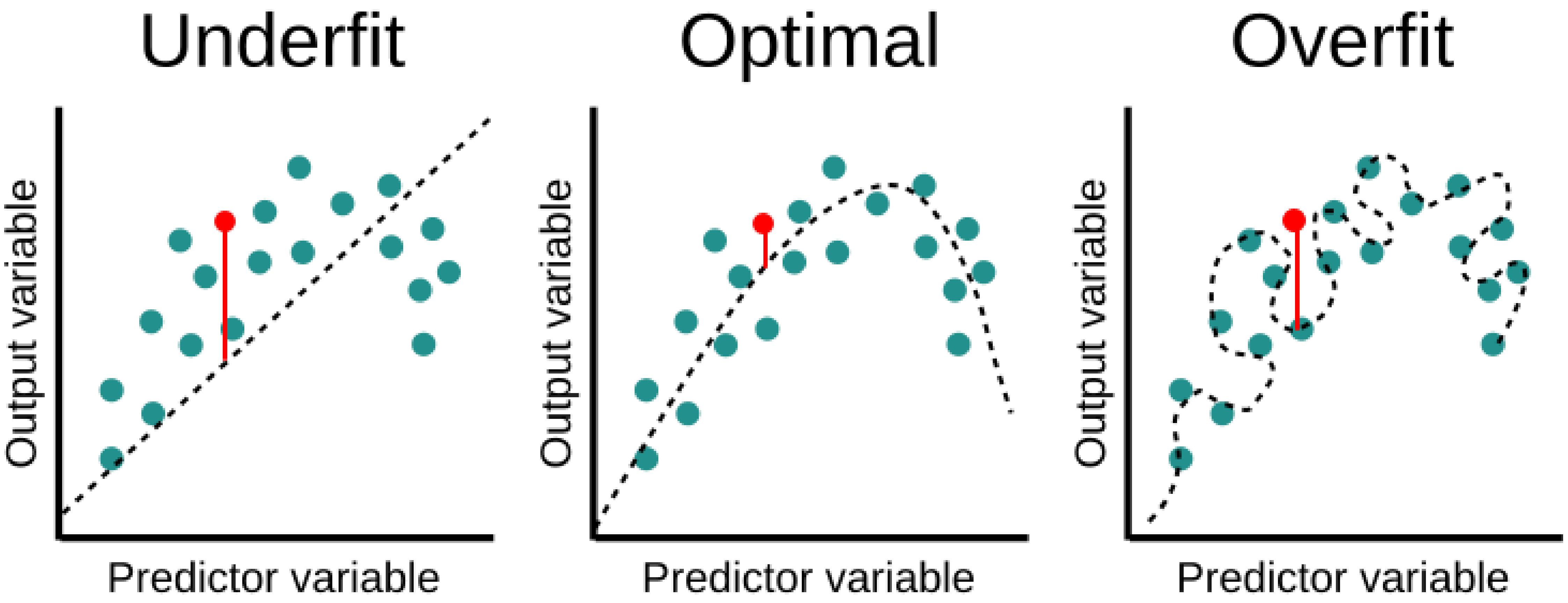
Deep Learning Pipeline

1. **Dataset**
2. **Model**
3. **Cost function**
4. **Train until converge**
    1. Forward
    2. Compute Loss
    3. Backward
    4. Gradient Descent

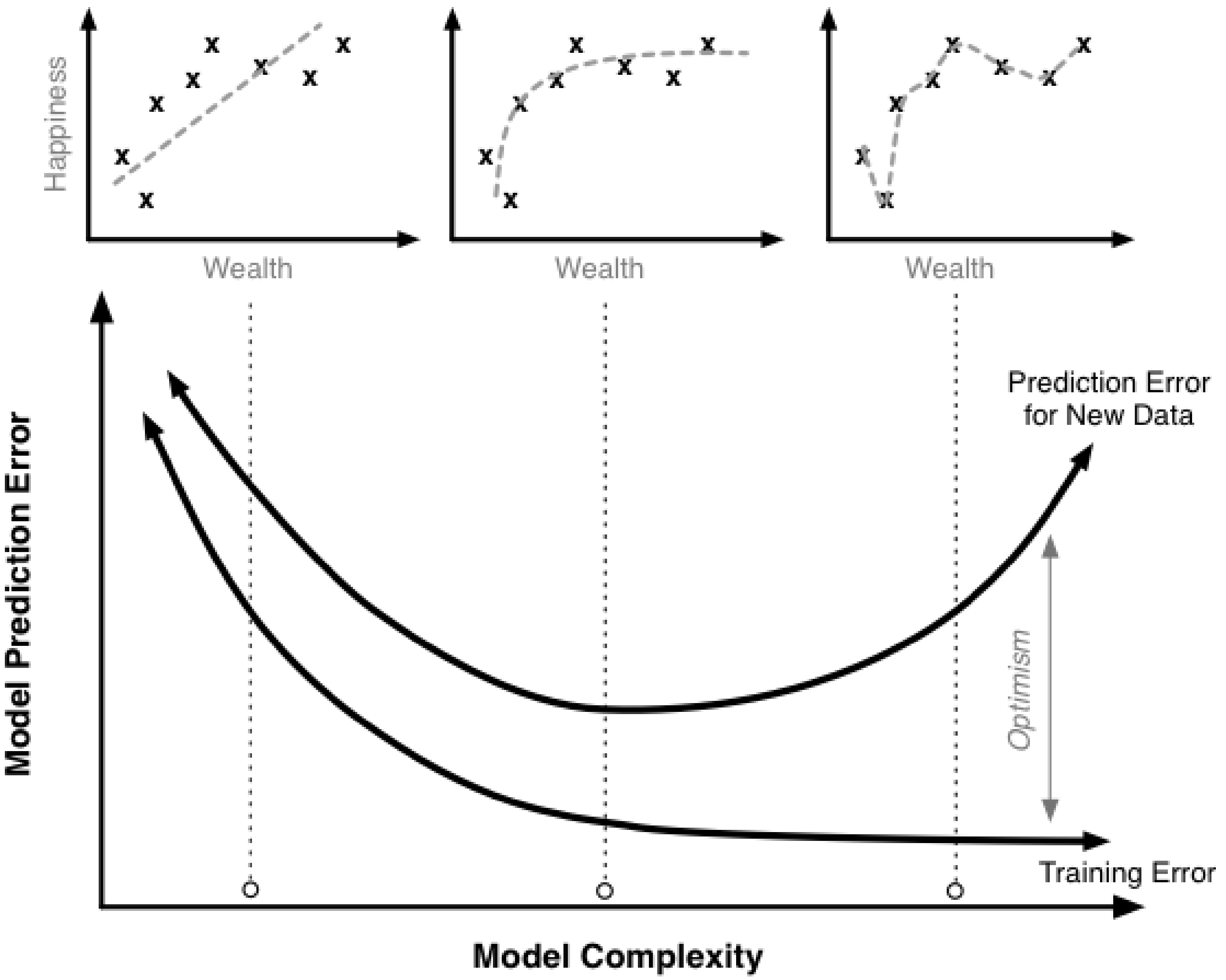


$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Underfitting and Overfitting



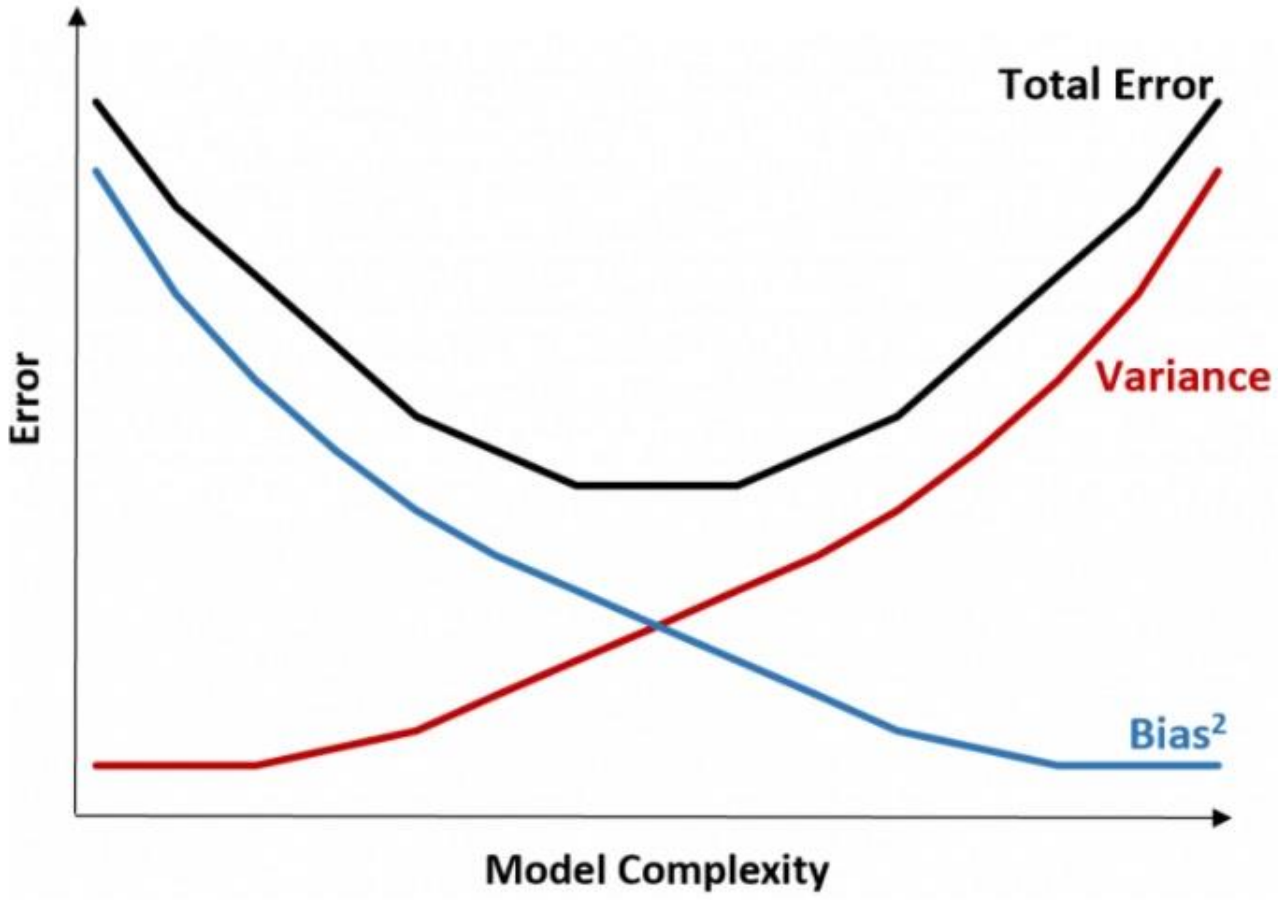
Underfitting and Overfitting

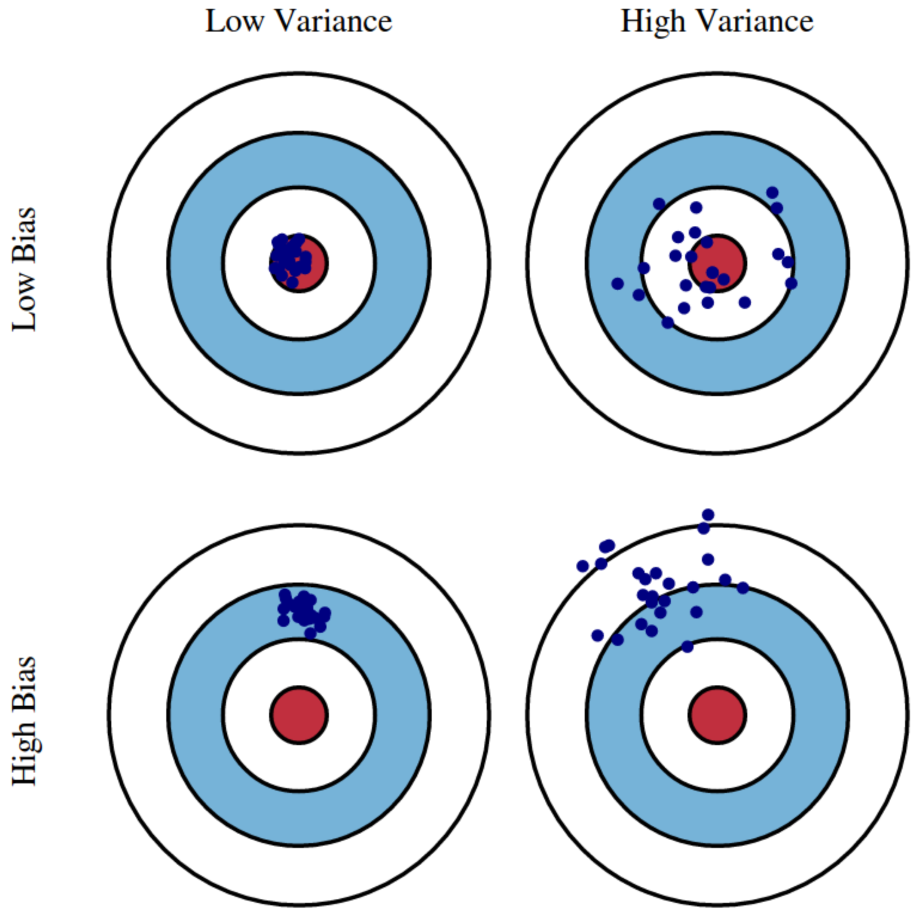


Bias and Variance

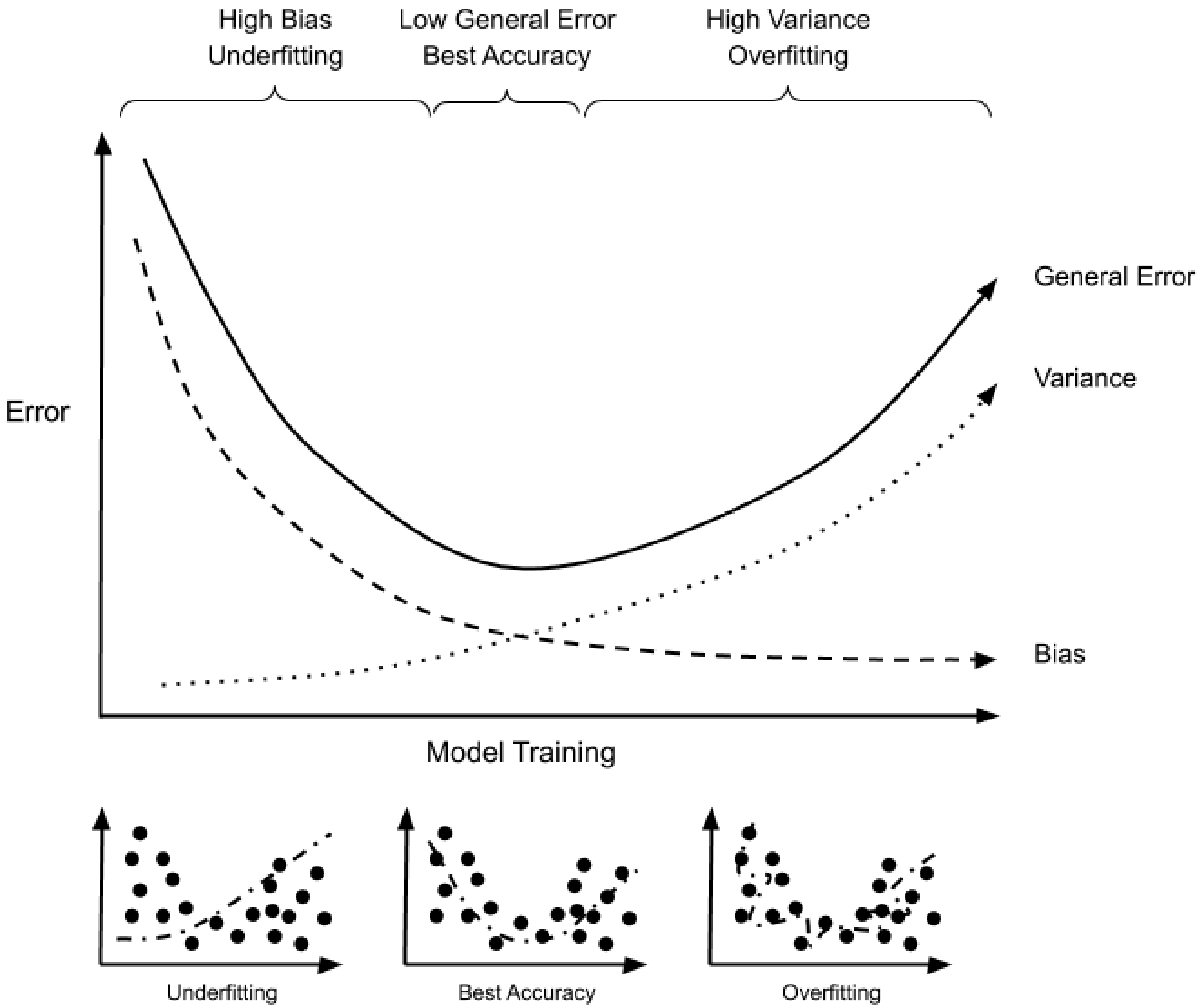
$$
\begin{aligned}
E[(y-\hat{f})^2] &= E[y^2+\hat{f}^2-2y\hat{f}] \\
&= E[y^2]+E[\hat{f}^2]-2E[y\hat{y}] \\
&= Var[y]+\{E[y]\}^2+Var[\hat{f}]+\{E[\hat{f}]\}^2-2E[(f+e)\hat{f}] \\
&= Var[y]+Var[\hat{f}]+f^2+\{E[\hat{f}]\}^2-2E[f\hat{f}] \\
&= \sigma^2+Var[\hat{f}]+(f-E[\hat{f}])^2 \\
&= \sigma^2+Var[\hat{f}]+bias[\hat{f}]^2
\end{aligned}
$$

Bias and Variance





Bias and Variance



Underfitting and Overfitting

How to solve underfitting

- **복잡한 모델 사용**
- **학습 시간 증가**

- **더 좋은 특성 (Machine Learning)**

Underfitting and Overfitting

How to solve overfitting

- **더 많은 데이터 수집**
- **데이터의 Noise 감소**
- **Regularization**
- **Early Stopping**

- **불필요한 특성 제거 (Machine Learning)**

Regularization

1. **L1 Regularization**

$$\Omega(W) = \lambda \|W\|_1 = \sum_i |W_i|$$

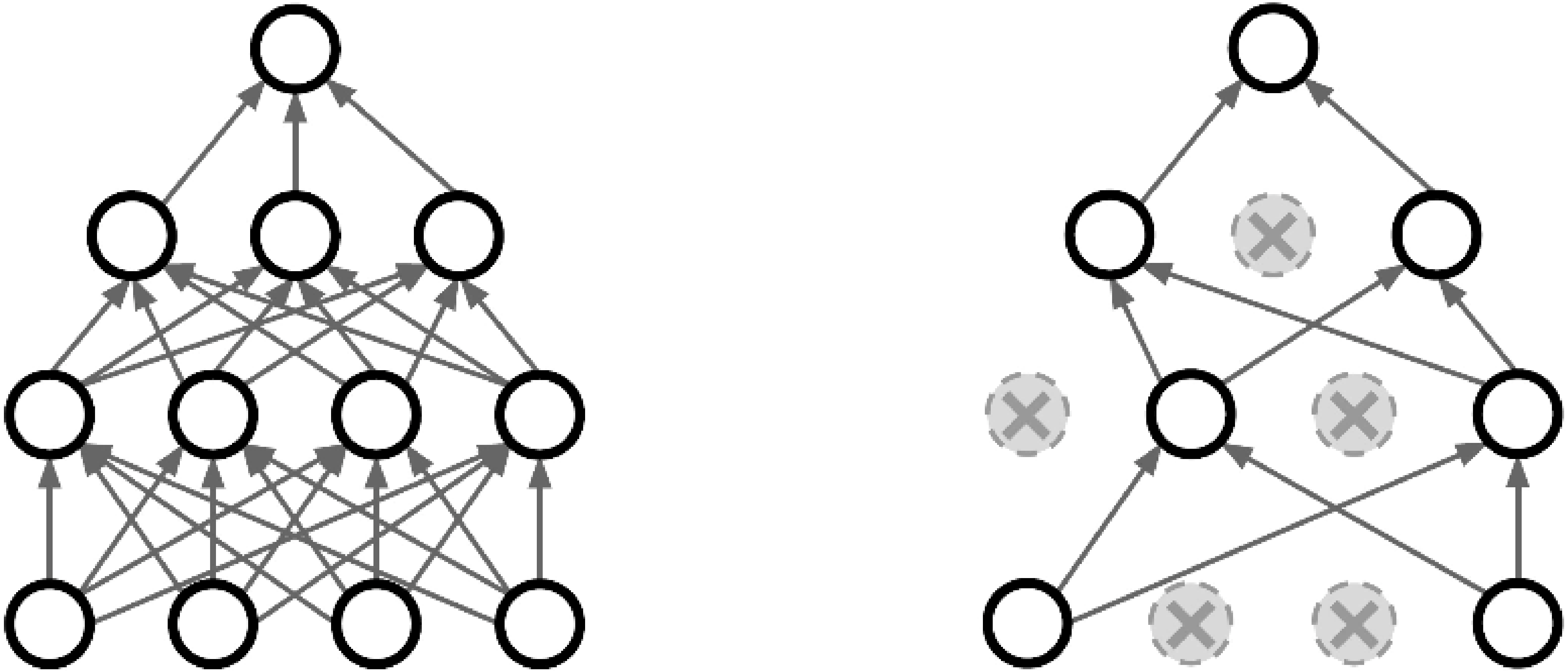
2. **L2 Regularization**

$$\Omega(W) = \lambda \|W\|_2^2 = \sum_i W_i^2$$

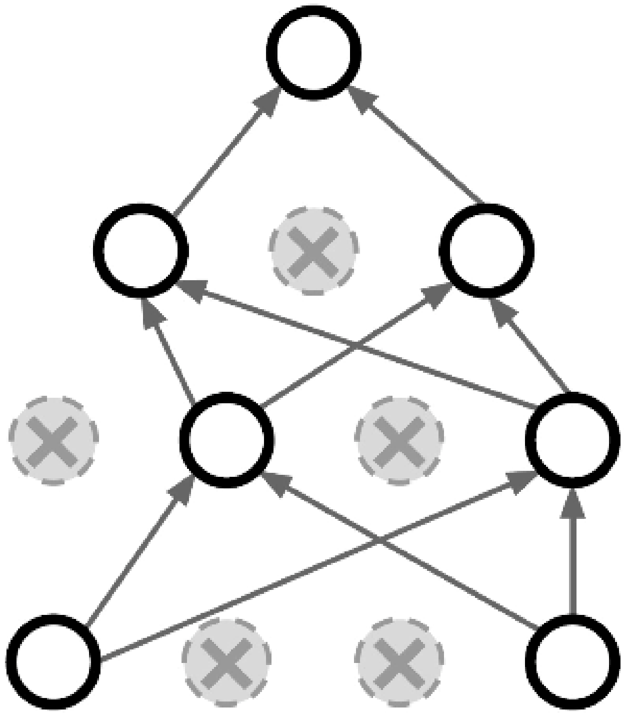
3. **Loss + Regularization**

$$\arg\min_W (L(W) + \Omega(W))$$

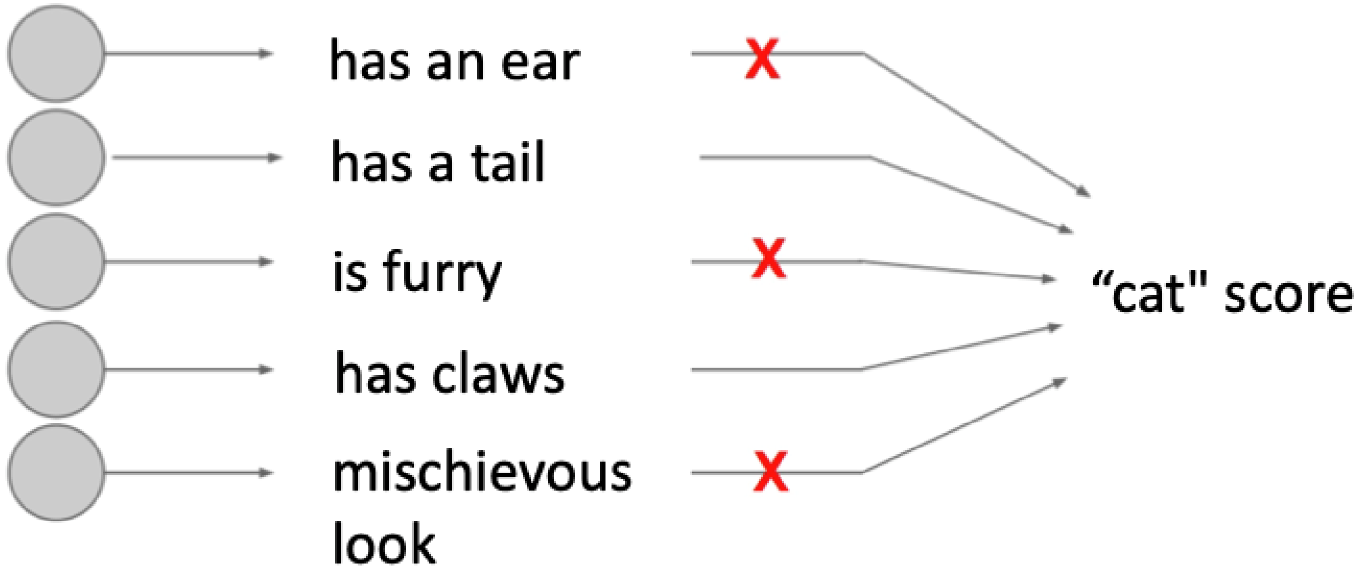
Regularization: Dropout



Regularization: Dropout

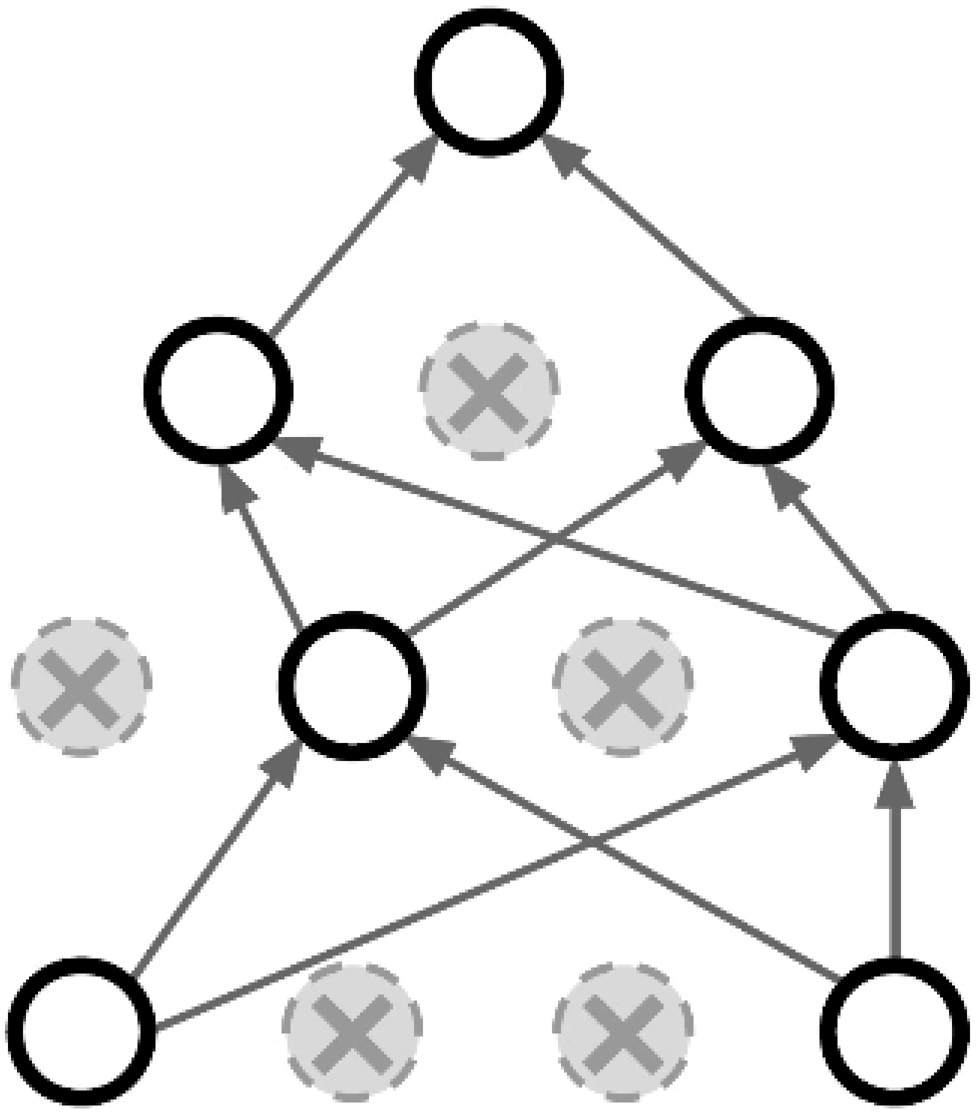


Forces the network to have a redundant representation;
Prevents co-adaptation of features



Regularization: Dropout

Dropout이 파라미터를 공유하는 큰 Ensemble 모델을 학습하는 것과 같다고 보는 관점도 있다.



Loss

1. **L1 Loss**
2. **MSE Loss**
3. **CrossEntropy Loss**
4. **NLL Loss**
5. **BCE Loss**
6. **CosineEmbedding Loss**
7. **HuberLoss**

L1 Loss

L1LOSS

CLASS torch.nn.L1Loss(*size_average=None*, *reduce=None*, *reduction='mean'*) [SOURCE]

Creates a criterion that measures the mean absolute error (MAE) between each element in the input x and target y.

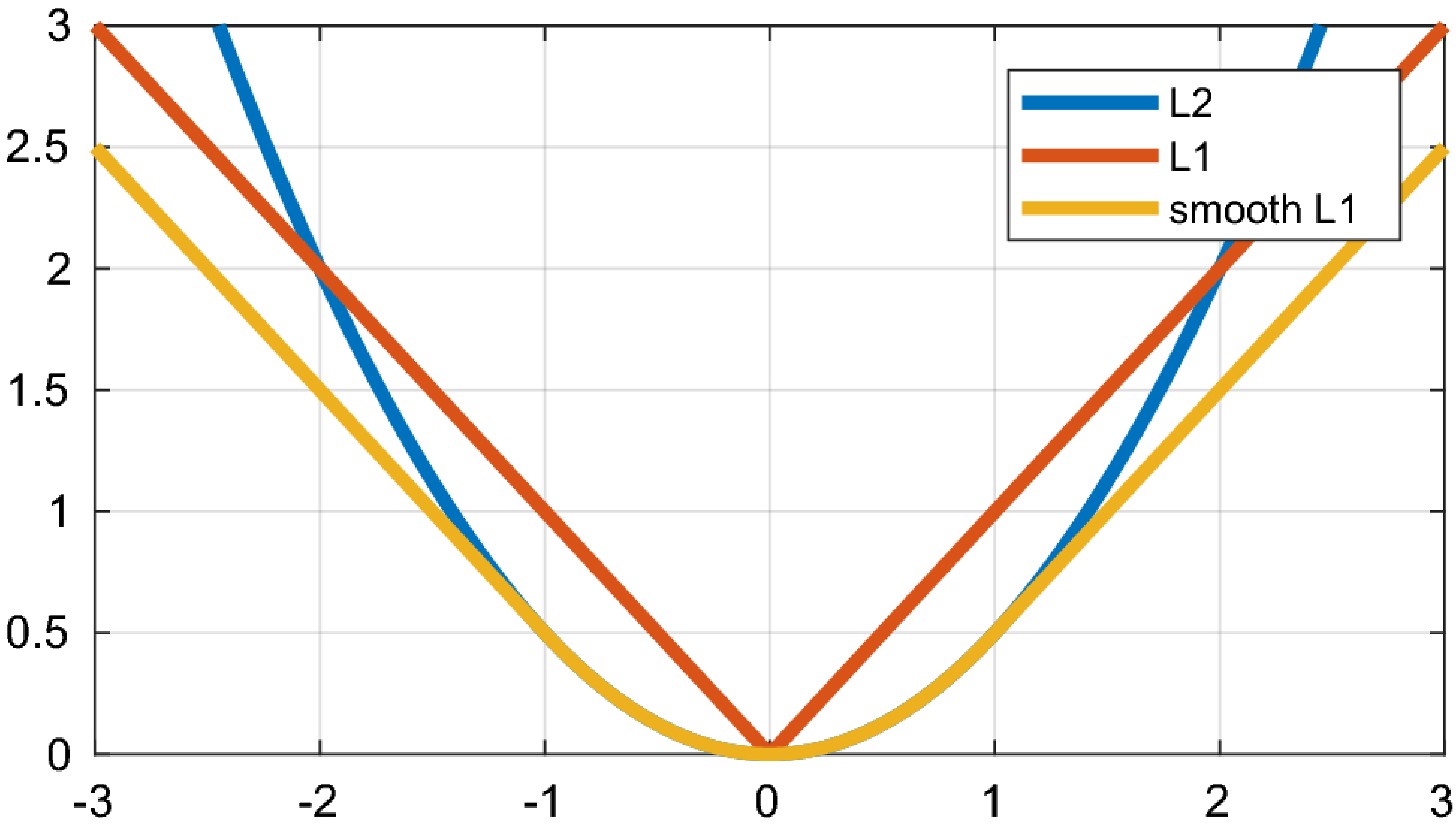
The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = \left| x_n - y_n \right|,$$

where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then:

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{`mean';} \\ \operatorname{sum}(L), & \text{if reduction} = \text{`sum'.} \end{cases}$$

L1 Loss



MSE Loss

MSELOSS

CLASS torch.nn.MSELoss(*size_average=None*, *reduce=None*, *reduction='mean'*) [SOURCE]

Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.

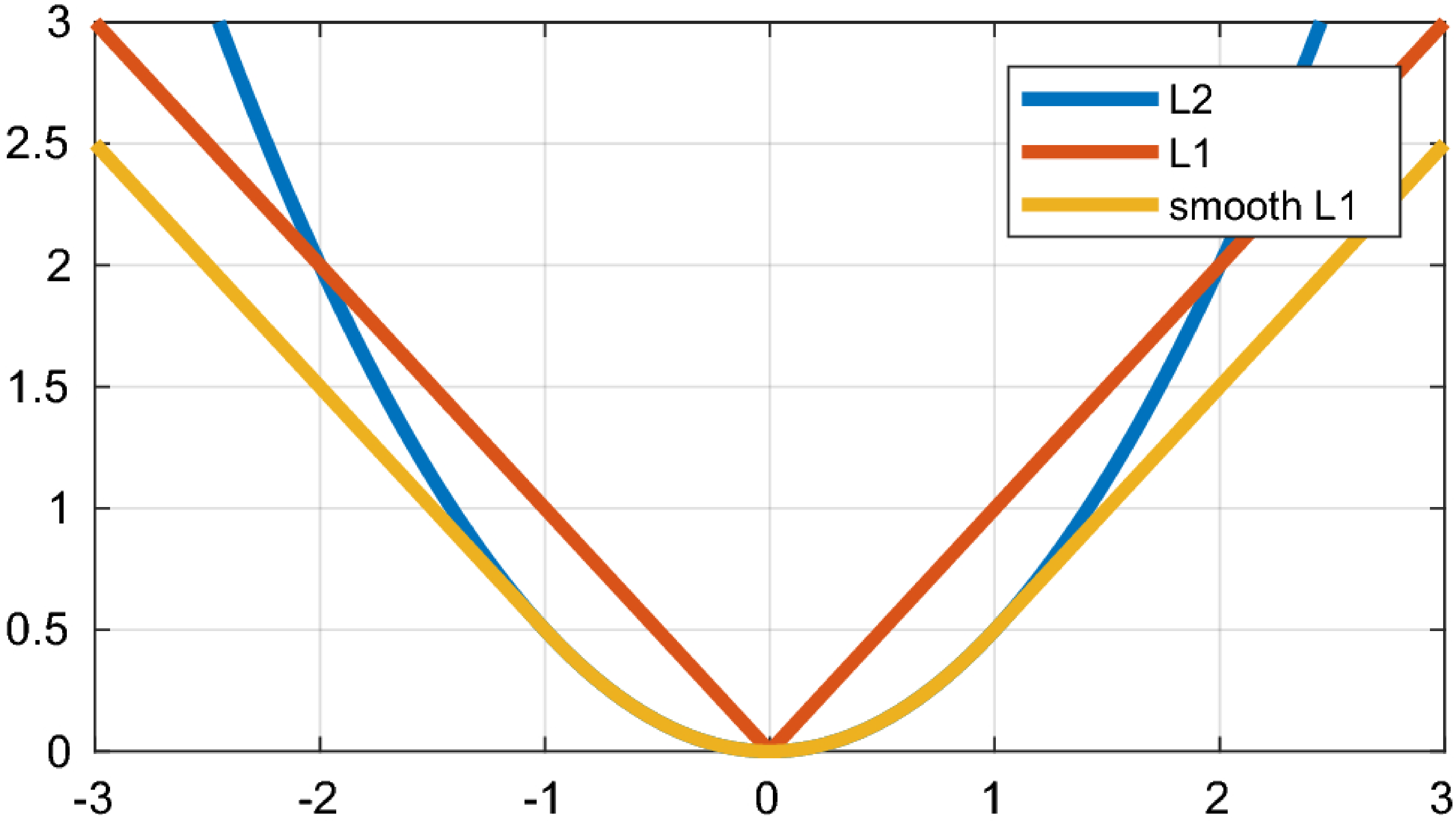
The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$$

where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then:

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{`mean';} \\ \operatorname{sum}(L), & \text{if reduction} = \text{`sum'.} \end{cases}$$

MSE Loss



Huber Loss

HUBERLOSS

CLASS torch.nn.HuberLoss(*reduction='mean'*, *delta=1.0*) [SOURCE]

Creates a criterion that uses a squared term if the absolute element-wise error falls below delta and a delta-scaled L1 term otherwise. This loss combines advantages of both `L1Loss` and `MSELoss`; the delta-scaled L1 region makes the loss less sensitive to outliers than `MSELoss`, while the L2 region provides smoothness over `L1Loss` near 0. See Huber loss for more information.

For a batch of size N, the unreduced loss can be described as:

$$\ell(x, y) = L = \{l_1, ..., l_N\}^T$$

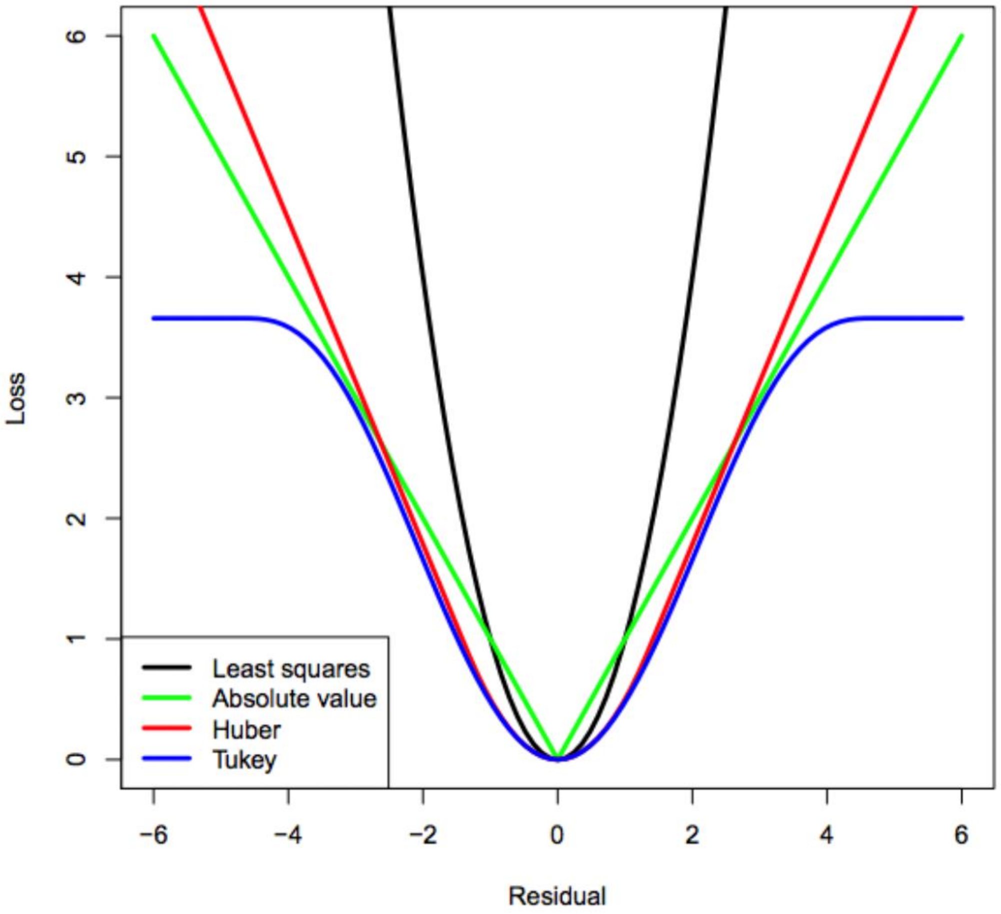
with

$$l_n = \begin{cases} 0.5(x_n - y_n)^2, & \text{if } |x_n - y_n| < delta \\ delta * (|x_n - y_n| - 0.5 * delta), & \text{otherwise} \end{cases}$$

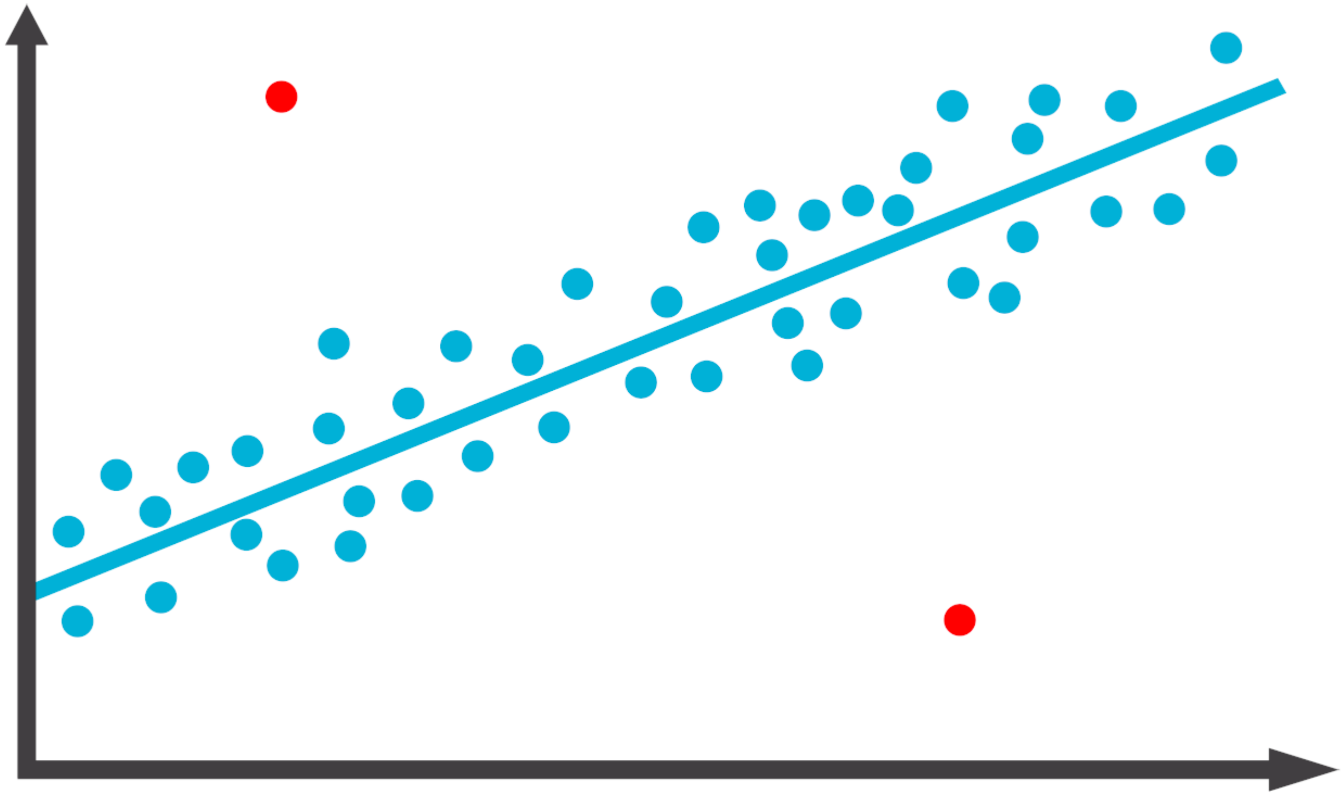
If *reduction* is not *none*, then:

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{`mean'}; \\ \operatorname{sum}(L), & \text{if reduction} = \text{`sum'}. \end{cases}$$

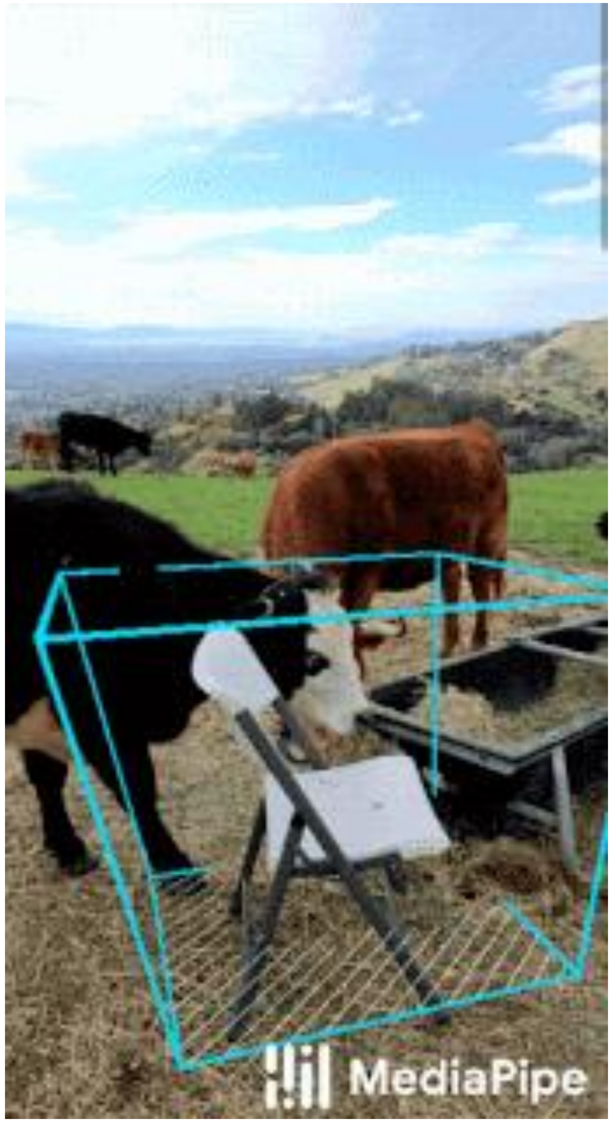
Huber Loss



Loss
Residual
−6
−4
−2
0
2
4
6
0
1
2
3
4
5
6
Least squares
Absolute value
Huber
Tukey



Loss for Regression



Loss for Regression

Demo: zero-shot depth estimation with DPT

Demo for Intel's DPT, a Dense Prediction Transformer for state-of-the-art dense prediction tasks such as semantic segmentation and depth estimation.

image

predicted depth

12.4s

CrossEntropy Loss

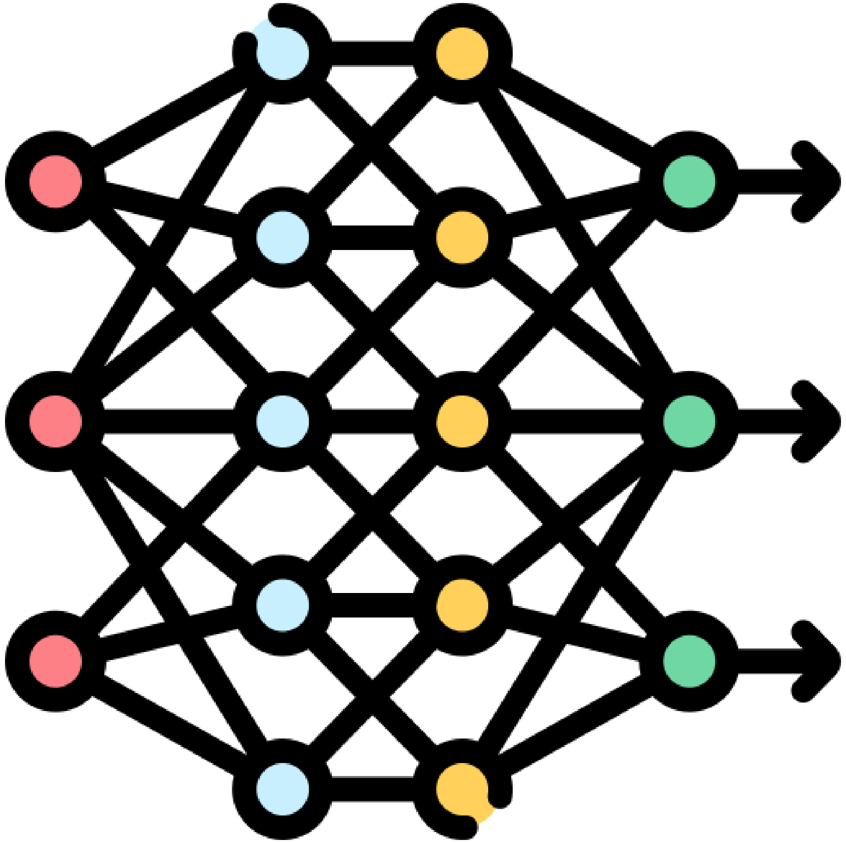
CROSSENTROPYLOSS

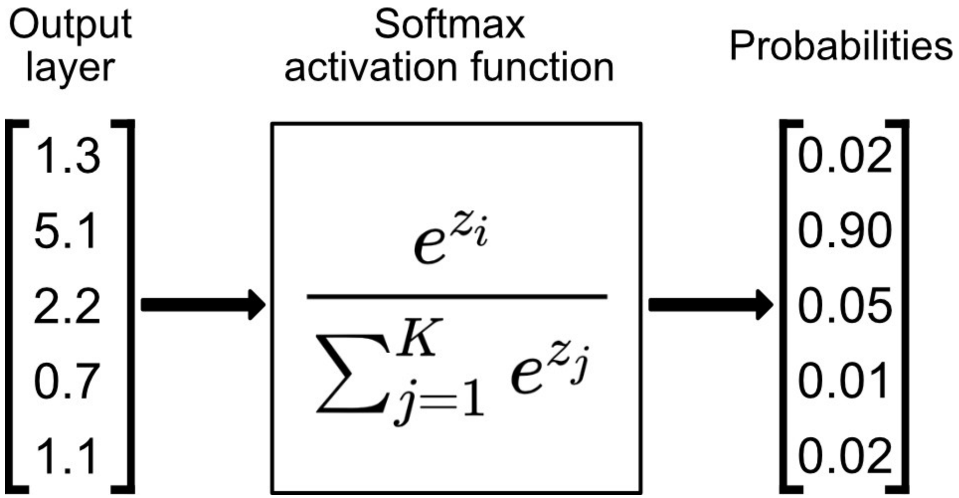
CLASS torch.nn.CrossEntropyLoss(*weight=None*, *size_average=None*, *ignore_index=- 100*, *reduce=None*, *reduction='mean'*, *label_smoothing=0.0*) [SOURCE]

This criterion computes the cross entropy loss between input logits and target.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

CrossEntropy Loss





$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

BCE Loss

BCELOSS

CLASS torch.nn.BCELoss(*weight=None*, *size_average=None*, *reduce=None*, *reduction='mean'*) [SOURCE]

Creates a criterion that measures the Binary Cross Entropy between the target and the input probabilities:

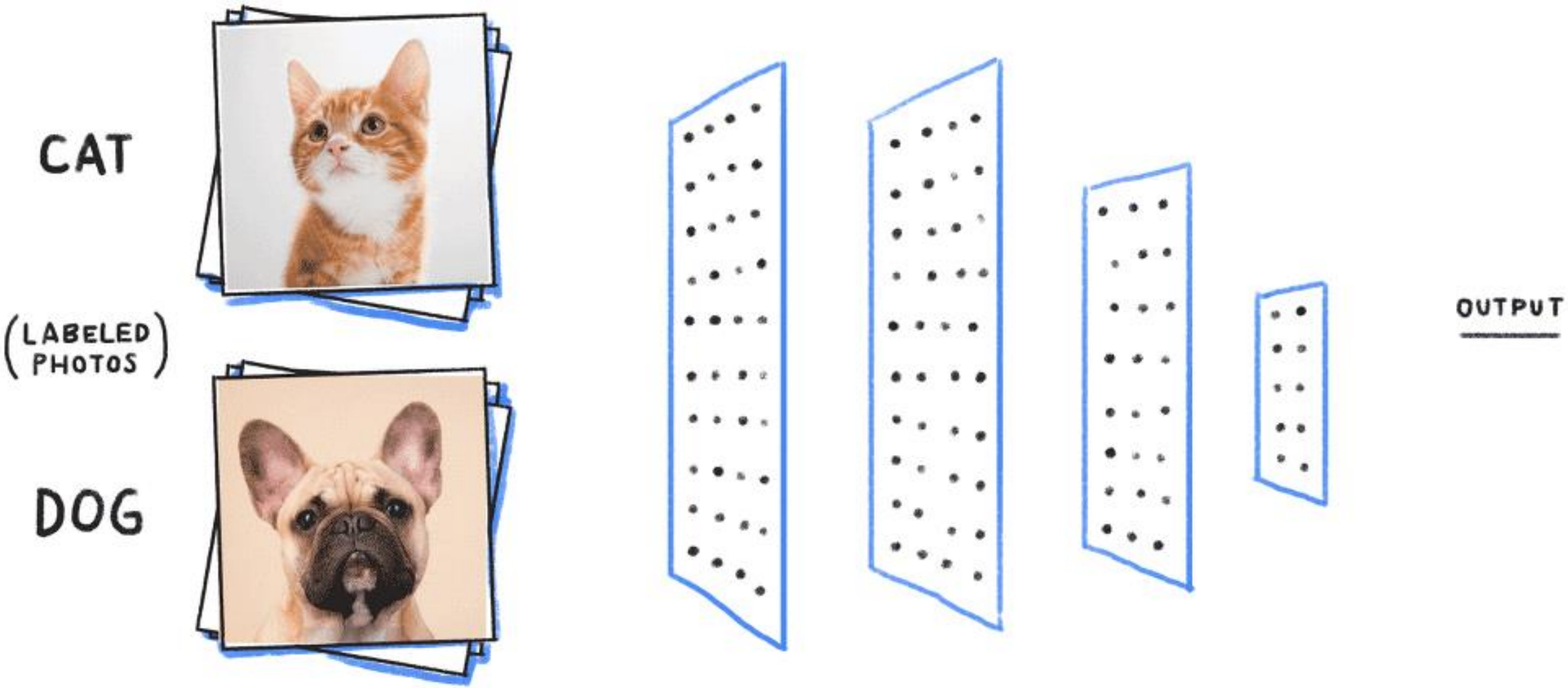
The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right],$$

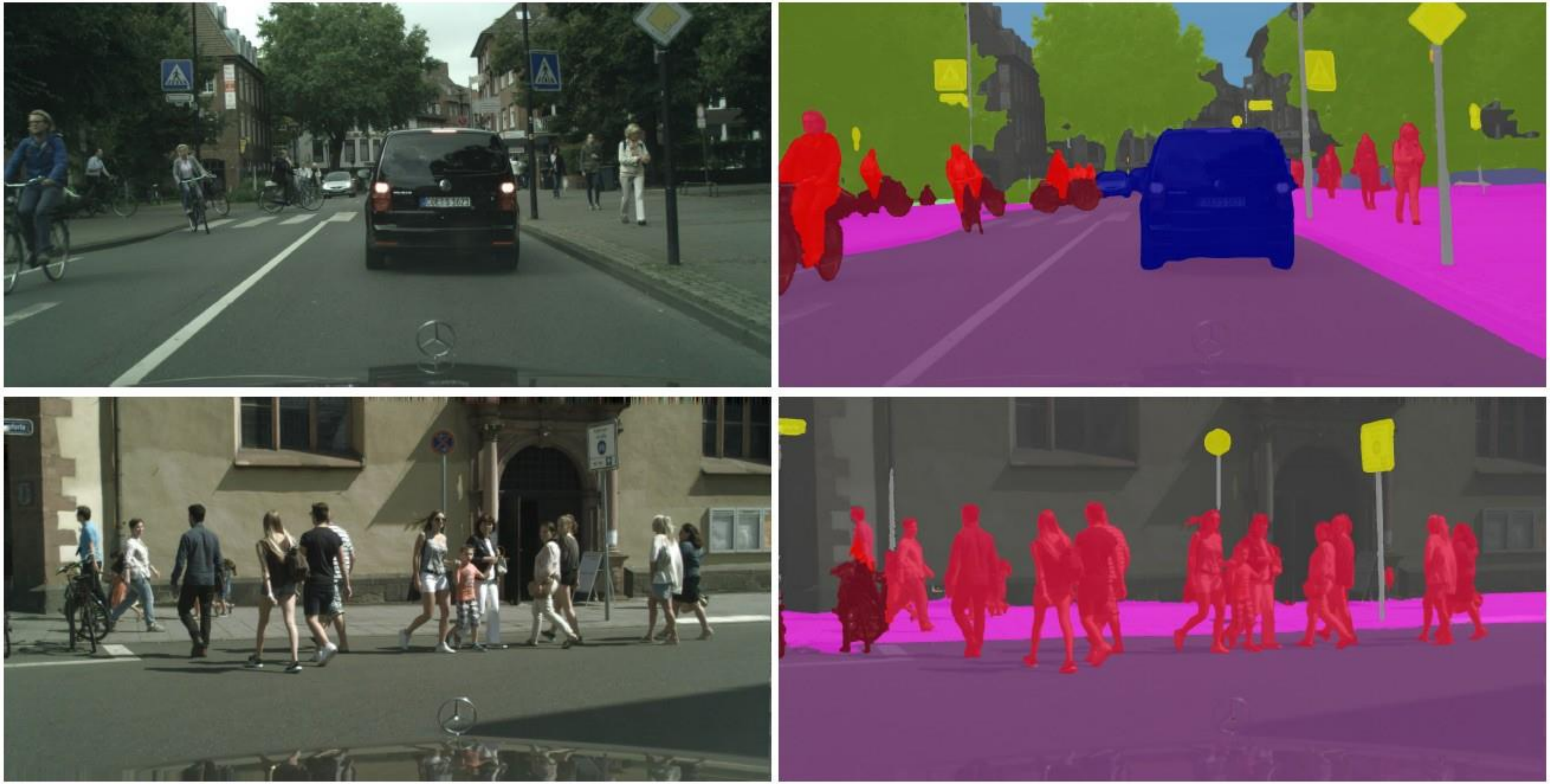
where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then

$$\ell(x, y) = \begin{cases} \operatorname{mean}(L), & \text{if reduction} = \text{`mean';} \\ \operatorname{sum}(L), & \text{if reduction} = \text{`sum'.} \end{cases}$$

BCE Loss

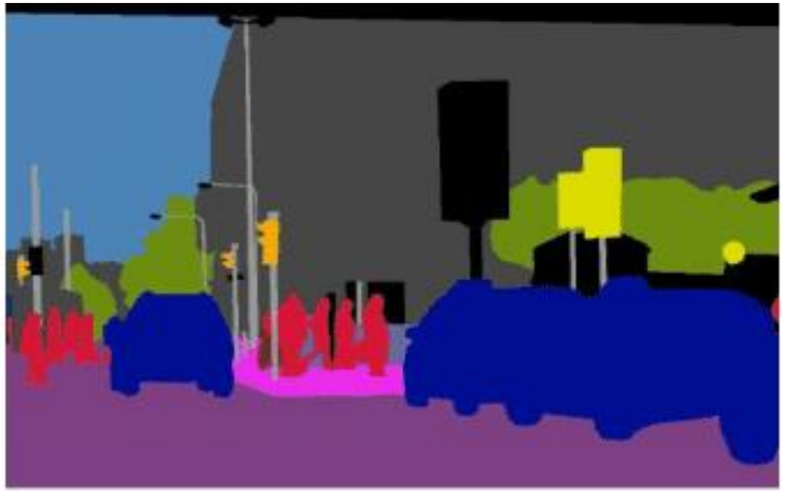


Loss for Classification



Loss for Classification


(a) Image


(b) Semantic Segmentation


(c) Instance Segmentation


(d) Panoptic Segmentation

CosineEmbedding Loss

COSINEEMBEDDINGLOSS

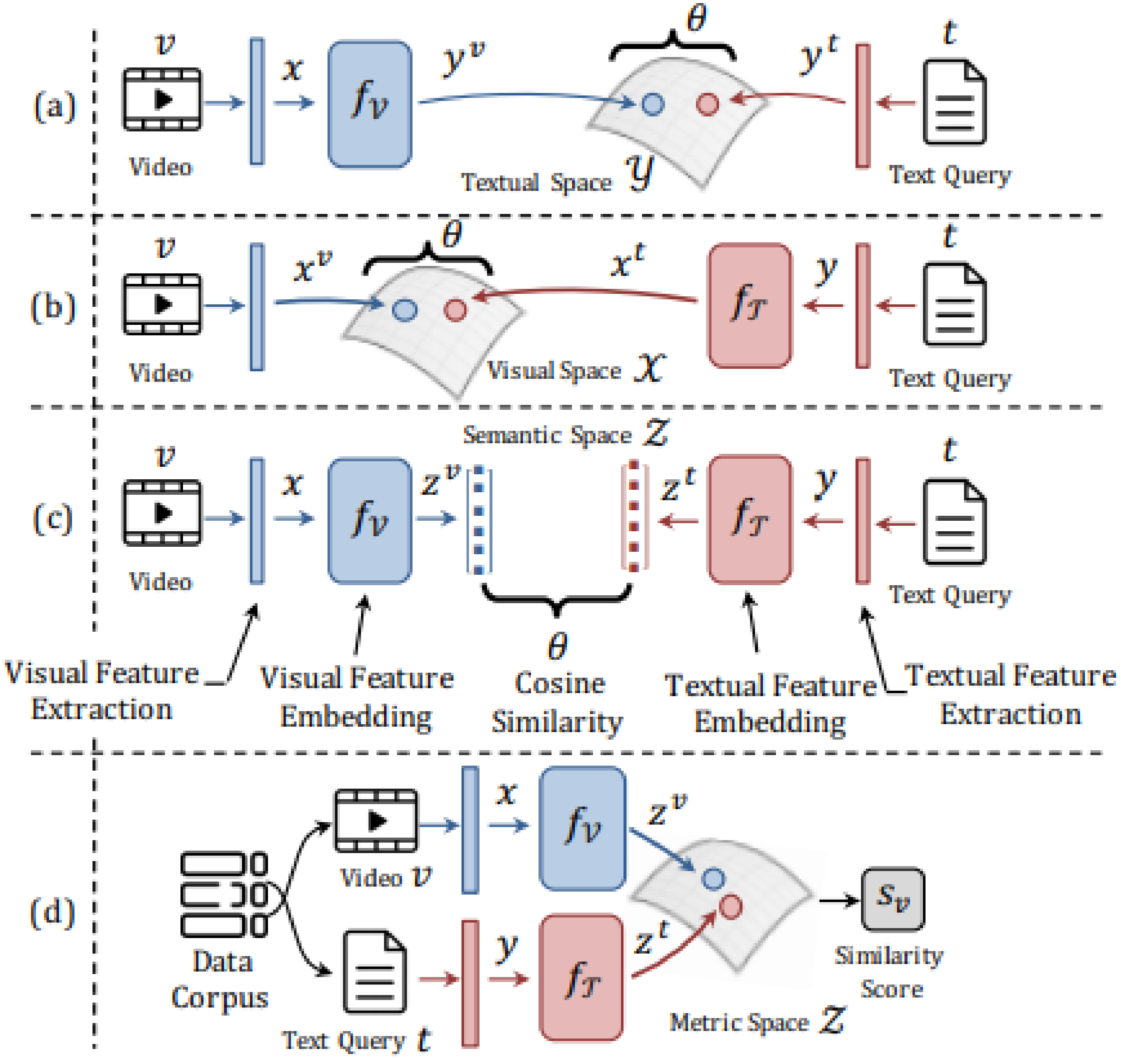
CLASS torch.nn.CosineEmbeddingLoss(*margin=0.0*, *size_average=None*, *reduce=None*, *reduction='mean'*) [SOURCE]

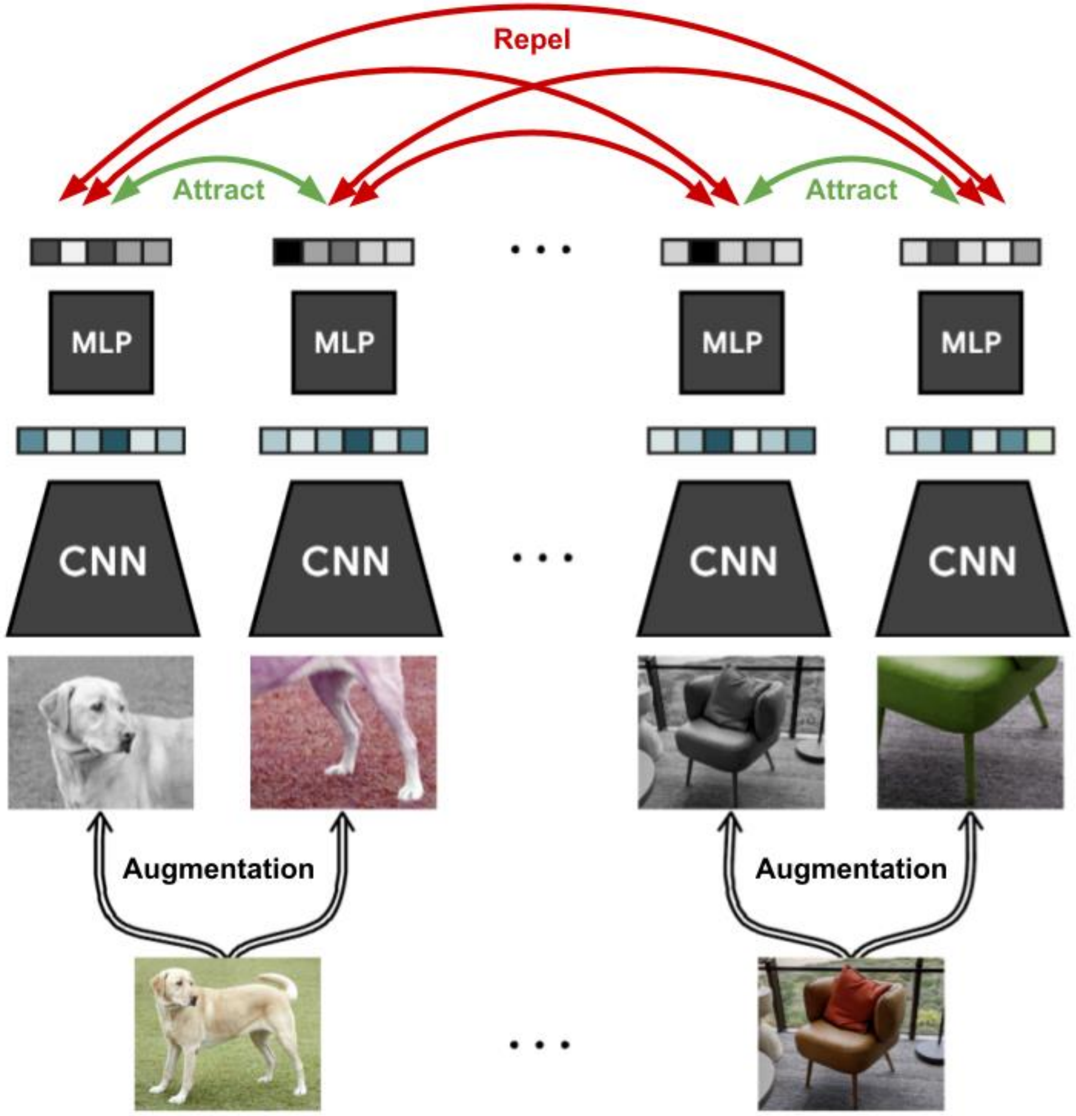
Creates a criterion that measures the loss given input tensors x_1, x_2 and a *Tensor* label y with values 1 or -1. This is used for measuring whether two inputs are similar or dissimilar, using the cosine similarity, and is typically used for learning nonlinear embeddings or semi-supervised learning.

The loss function for each sample is:

$$\text{loss}(x, y) = \begin{cases} 1 - \cos(x_1, x_2), & \text{if } y = 1 \\ \max(0, \cos(x_1, x_2) - \text{margin}), & \text{if } y = -1 \end{cases}$$

CosineEmbedding Loss





Thank you!
Q & A