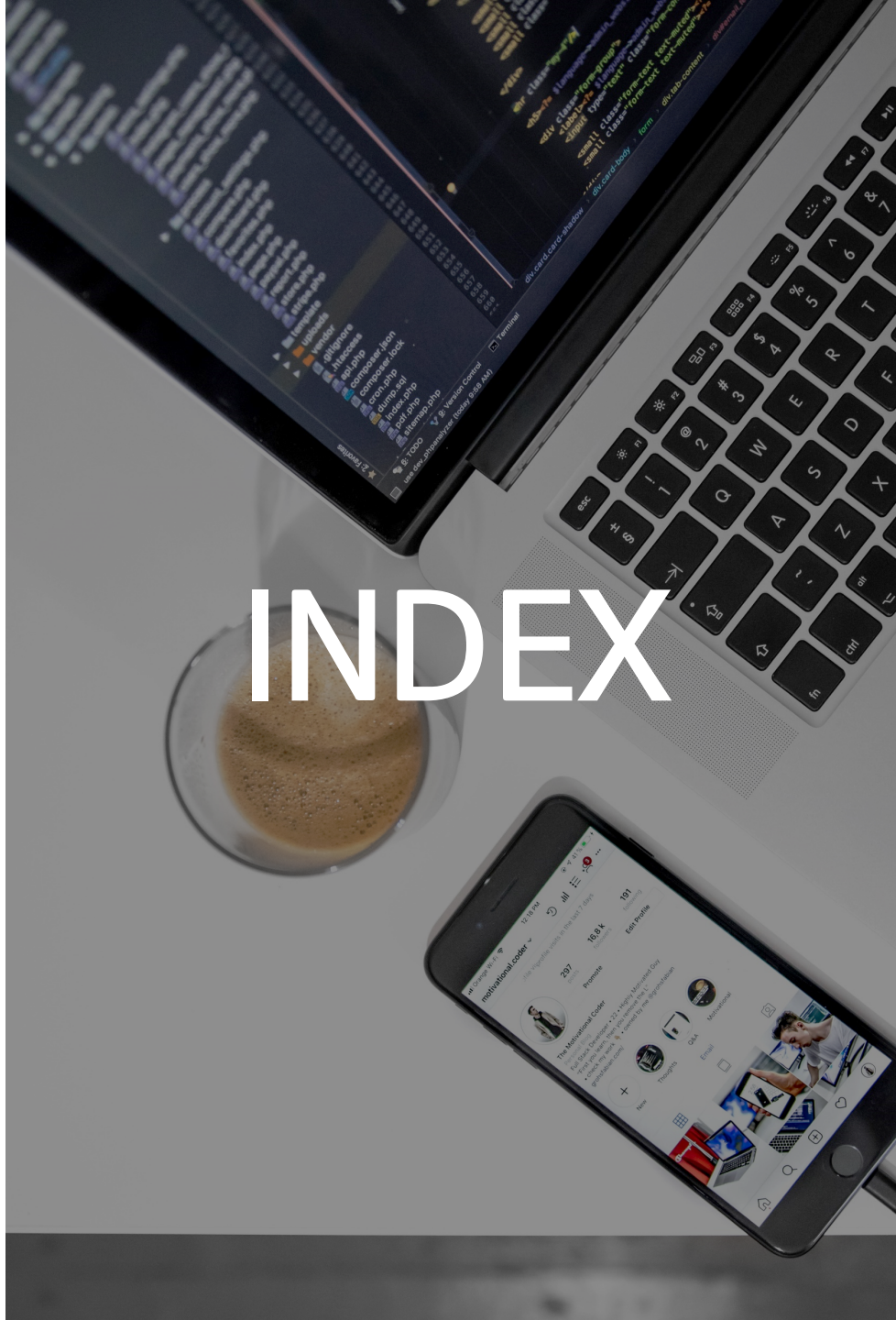
Summer NLP

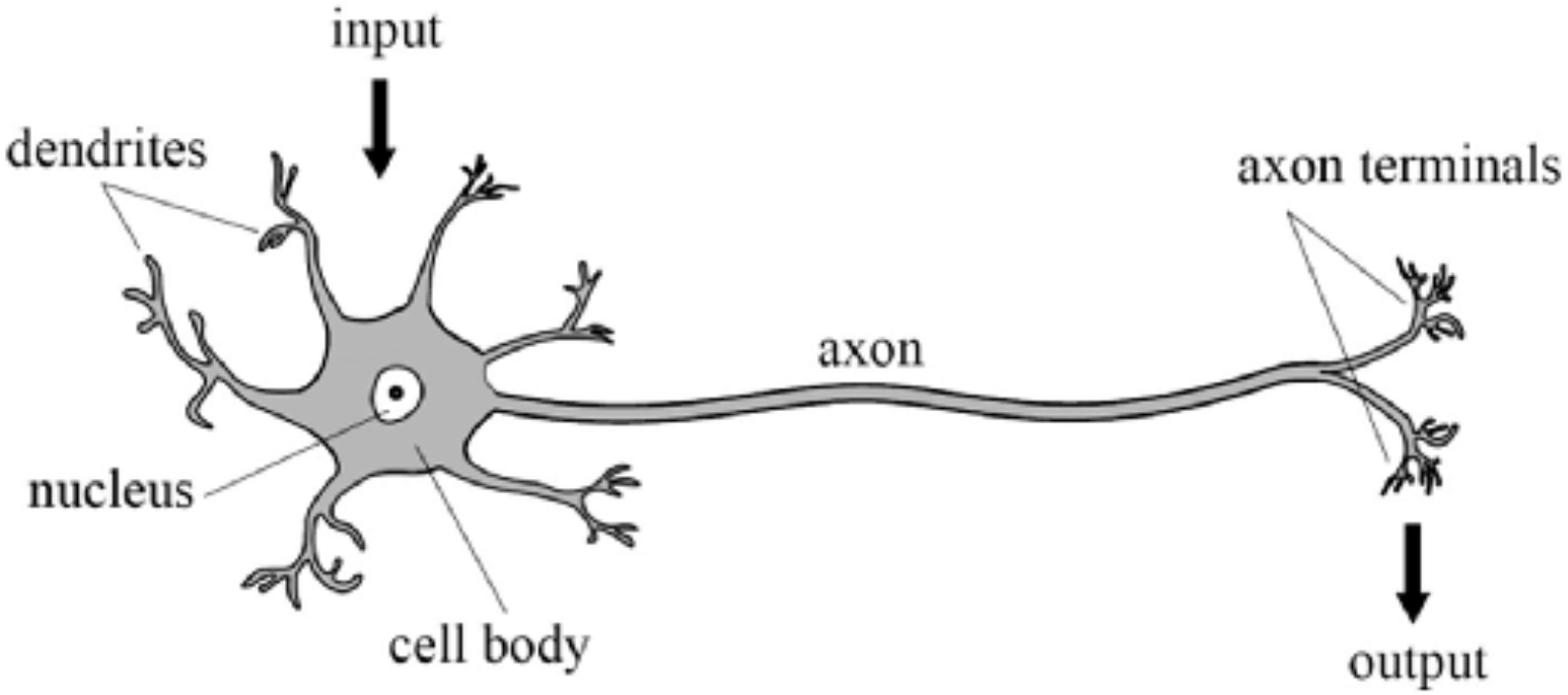
Lec 02.Perceptron for Sentiment Classification

파이토치로 배우는 자연어 처리 CHAPTER 1~2

INDEX



퍼셉트론



생물학적 뉴런

전기신호를 입력으로, 역치(Threshold)를 넘으면 다음 뉴런으로 신호 전달

퍼셉트론

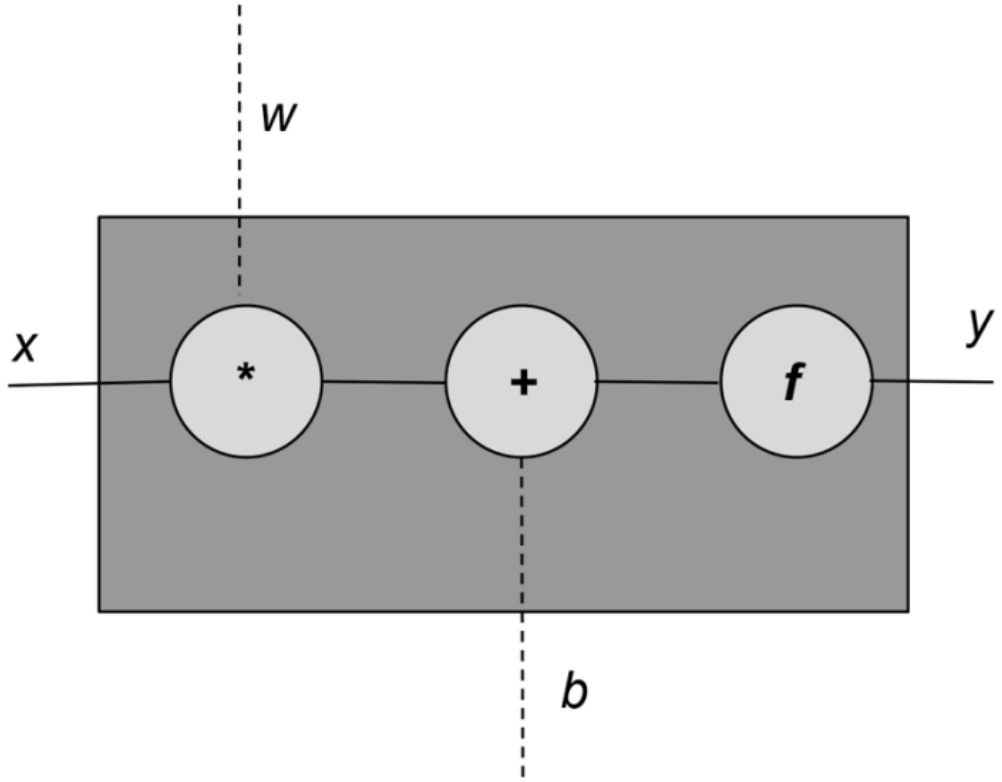


Figure 3-1. The computational graph for a perceptron with an input (x) and an output (y). The weights (w) and bias (b) constitute the parameters of the model.

퍼셉트론

입력에 **가중치(w)**를 곱하고, **편향(b)**을 더하여, **활성화 함수(f)**로 변환

퍼셉트론

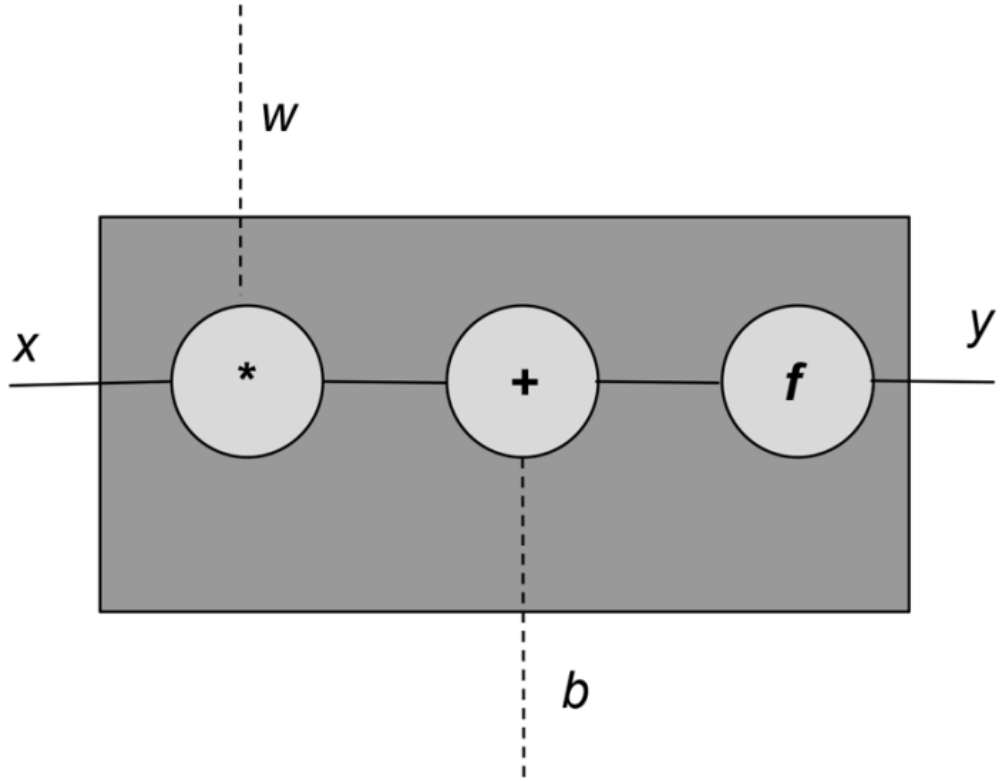
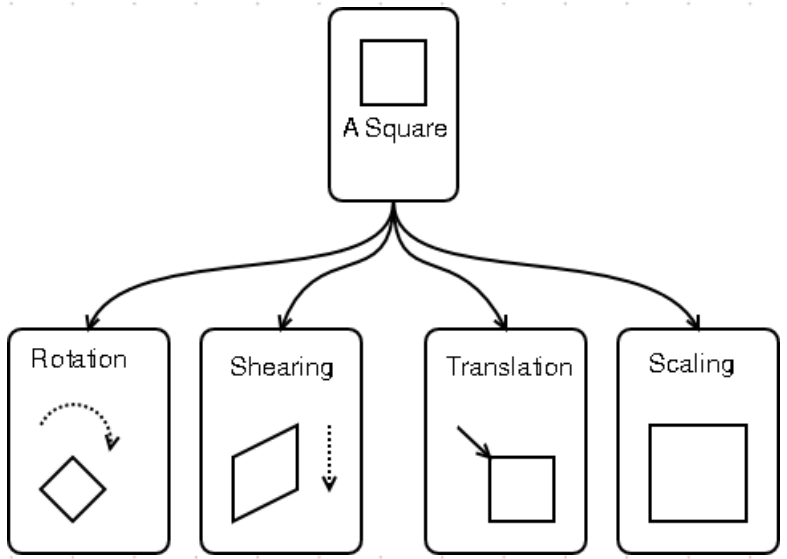


Figure 3-1. The computational graph for a perceptron with an input (x) and an output (y). The weights (w) and bias (b) constitute the parameters of the model.



퍼셉트론

wx+b는 'Affine Transform'이라 불리며, 이 변환 후 비선형 활성화함수를 거친 것

퍼셉트론

Example 3-1. Implementing a perceptron using PyTorch

```
import torch
import torch.nn as nn

class Perceptron(nn.Module):
    """ A perceptron is one linear layer """
    def __init__(self, input_dim):
        """
        Args:
            input_dim (int): size of the input features
        """
        super(Perceptron, self).__init__()
        self.fc1 = nn.Linear(input_dim, 1)

    def forward(self, x_in):
        """The forward pass of the perceptron

        Args:
            x_in (torch.Tensor): an input data tensor
                x_in.shape should be (batch, num_features)
        Returns:
            the resulting tensor. tensor.shape should be (batch,).
        """
        return torch.sigmoid(self.fc1(x_in)).squeeze()
```

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`

Shape:

- 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 $H_{out} = \text{out\_features}$.

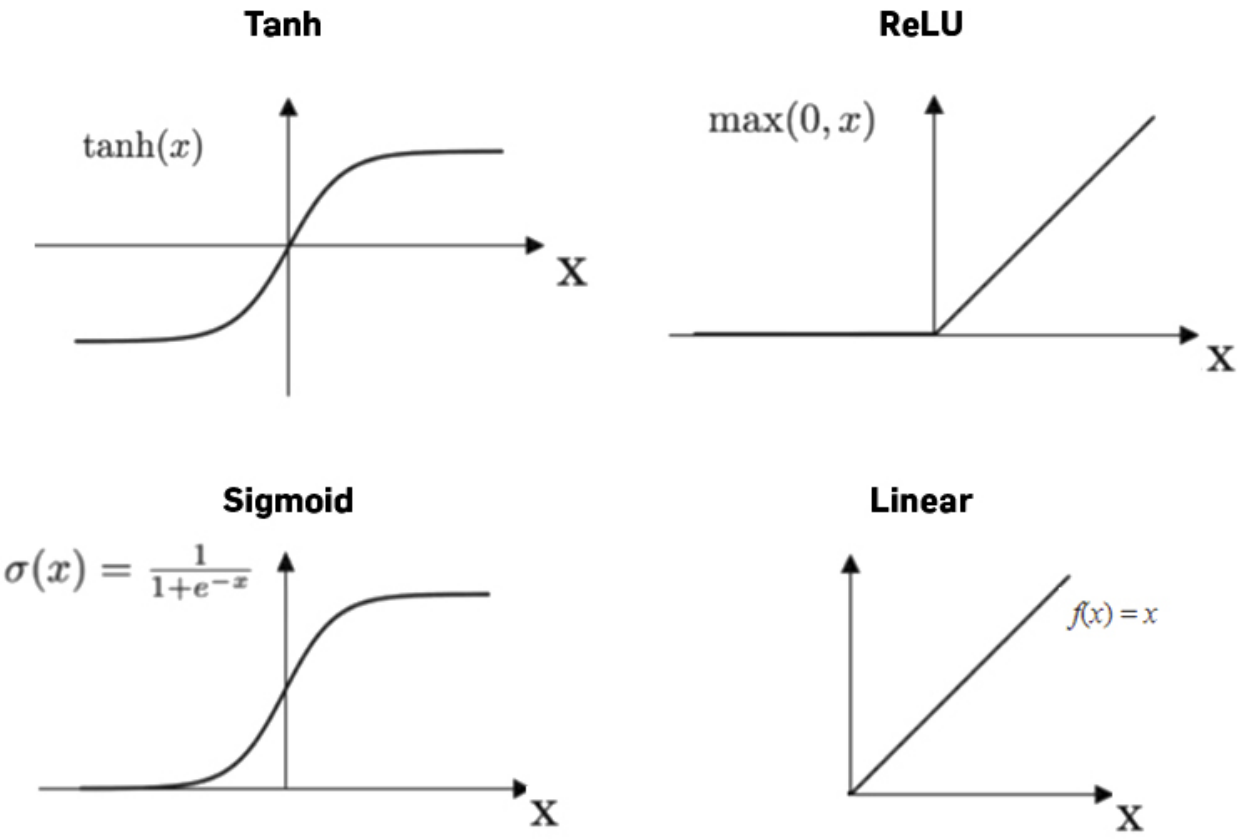
Variables

- **~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 $k = \frac{1}{\text{in\_features}}$

퍼셉트론

PyTorch로 Perceptron 구현

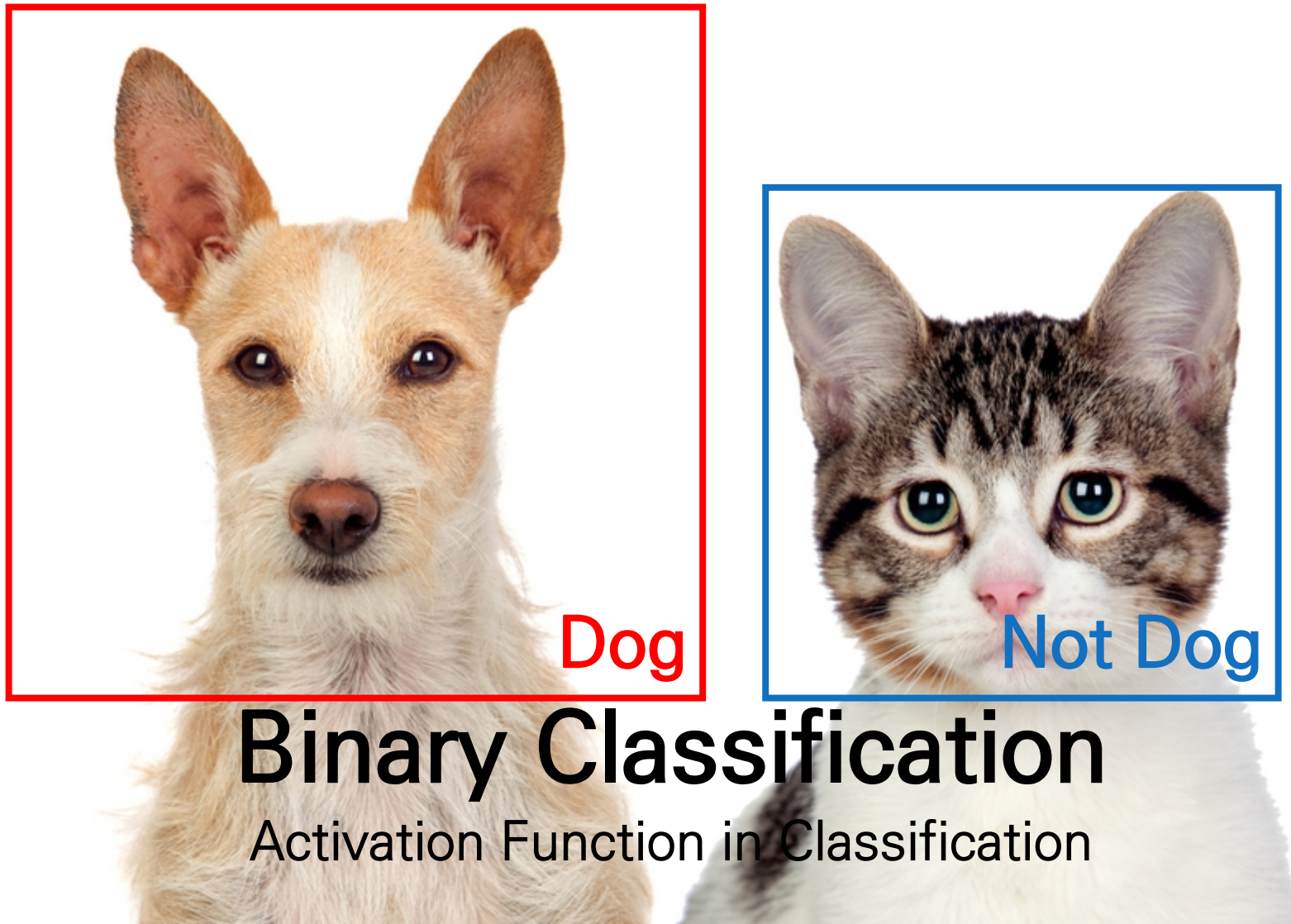
활성화 함수



활성화 함수

비선형 함수로, 데이터의 복잡한 관계를 포착하기 위해 사용

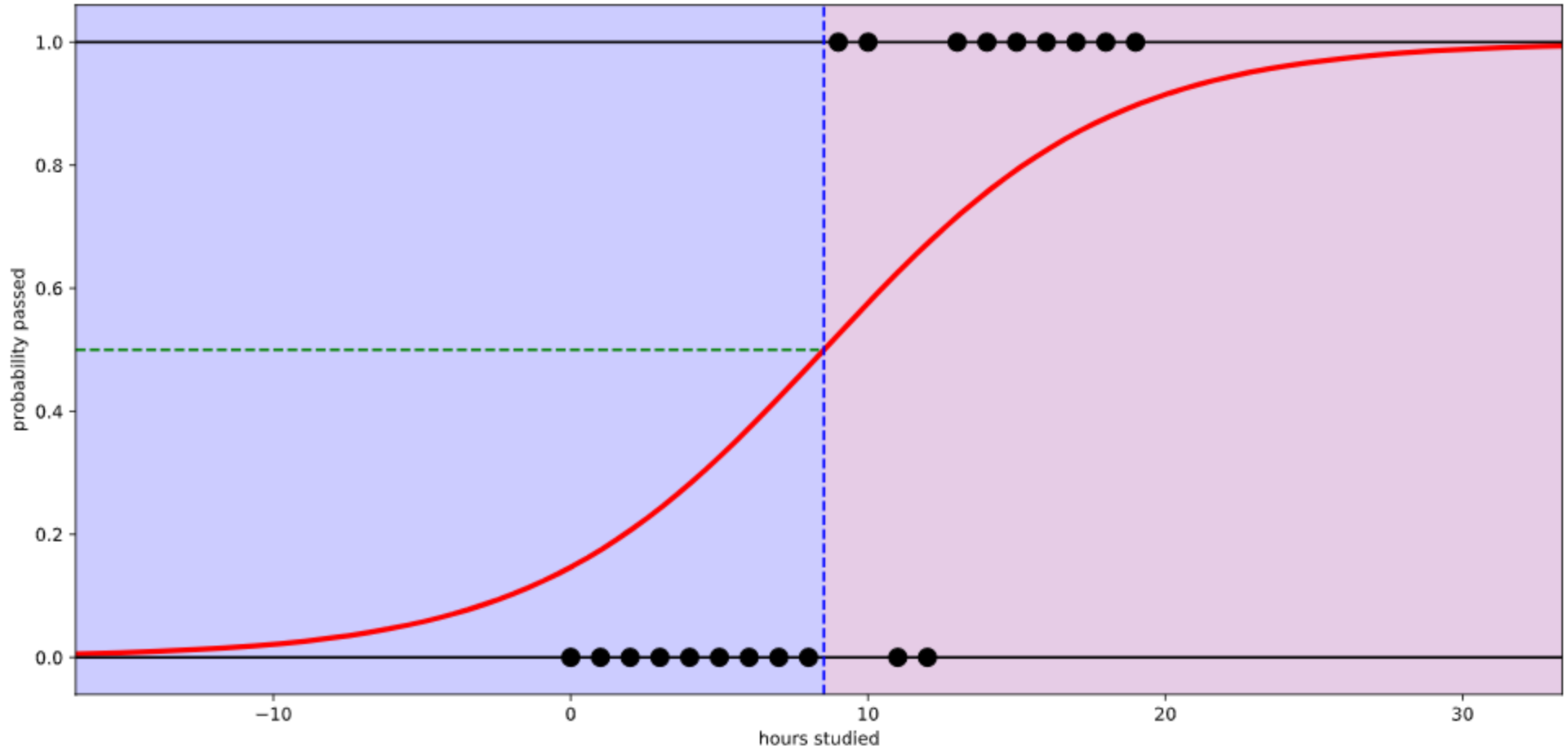
활성화 함수 (Review)



Binary Classification

Activation Function in Classification

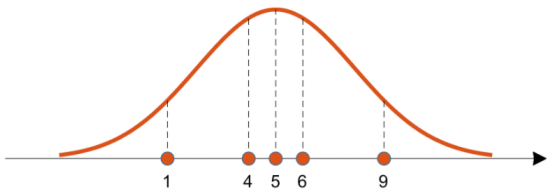
활성화 함수 (Review)



활성화 함수

Binary Classification에서는 ‘Sigmoid’를 사용. (출력값을 0~1의 확률로!)

활성화 함수 (Review)



$$P(x|\theta) = \prod_{k=1}^{n} P(x_k|\theta)$$

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X)}$$

1. Likelihood 계산 : 주어진 확률분포에서의 데이터에 대한 가능도

$$a_k = ln(P(X|Y_k)P(Y_k))$$

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)} = \frac{e^{a_1}}{e^{a_1} + e^{a_2}}$$

$$P(Y_1|X) = \frac{1}{1 + e^{a_2 - a_1}}$$

2. 식 변형: likelihood와 prior의 곱을 a_k로 설정

$$P(Y_1|X) = \frac{1}{1 + e^{-a}}$$

3. 활성화 함수 정의: a를 선형함수로 추정하면,
정리된 식을 활성화 함수로 사용 가능.

활성화 함수

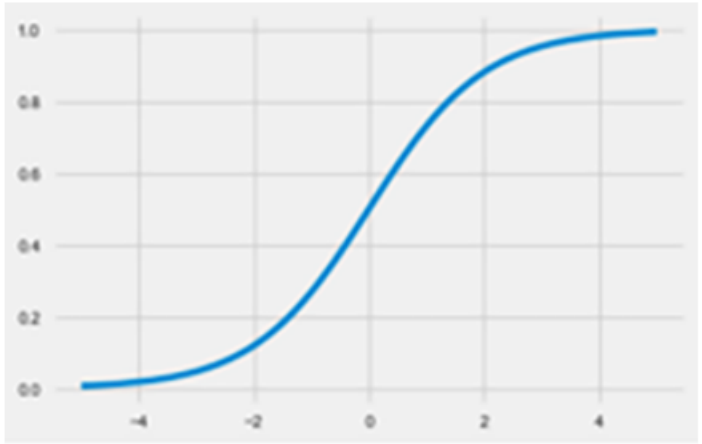
Binary Classification에서는 ‘Sigmoid’를 사용. (출력값을 0~1의 확률로!)

활성화 함수

Example 3-2. Sigmoid activation

```
import torch
import matplotlib.pyplot as plt

x = torch.range(-5., 5., 0.1)
y = torch.sigmoid(x)
plt.plot(x.numpy(), y.numpy())
plt.show()
```



$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid

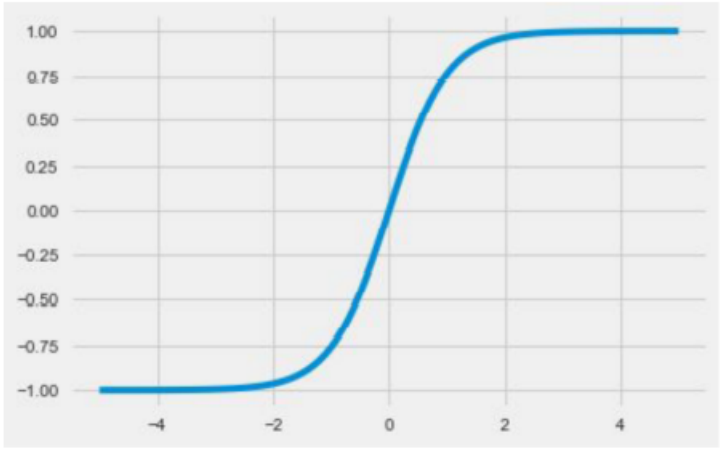
임의의 실숫값을 받아 0과 1 사이의 범위로 압축

활성화 함수

Example 3-3. Tanh activation

```
import torch
import matplotlib.pyplot as plt

x = torch.range(-5., 5., 0.1)
y = torch.tanh(x)
plt.plot(x.numpy(), y.numpy())
plt.show()
```



$$f(x) = \tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

tanh (Hyperpolic Tangent)

임의의 실숫값을 받아 −1과 1 사이의 범위로 압축 (Sigmoid의 선형 변환)

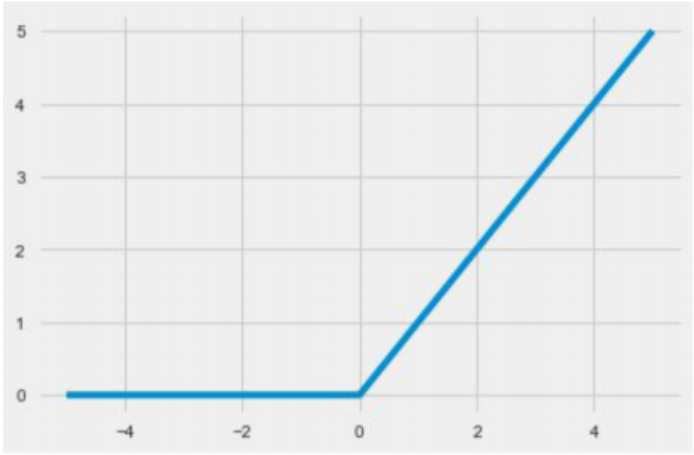
활성화 함수

Example 3-4. ReLU activation

```
import torch
import matplotlib.pyplot as plt

relu = torch.nn.ReLU()
x = torch.range(-5., 5., 0.1)
y = relu(x)

plt.plot(x.numpy(), y.numpy())
plt.show()
```



$$f(x) = \max(0, x)$$

ReLU

음수값을 0으로만 자르고, 나머지는 그대로

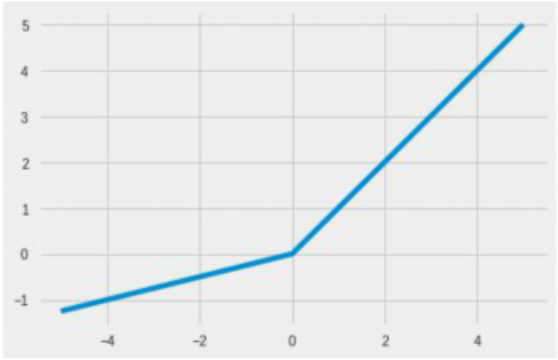
활성화 함수

Example 3-5. PReLU activation

```
import torch
import matplotlib.pyplot as plt

prelu = torch.nn.PReLU(num_parameters=1)
x = torch.range(-5., 5., 0.1)
y = prelu(x)

plt.plot(x.numpy(), y.numpy())
plt.show()
```

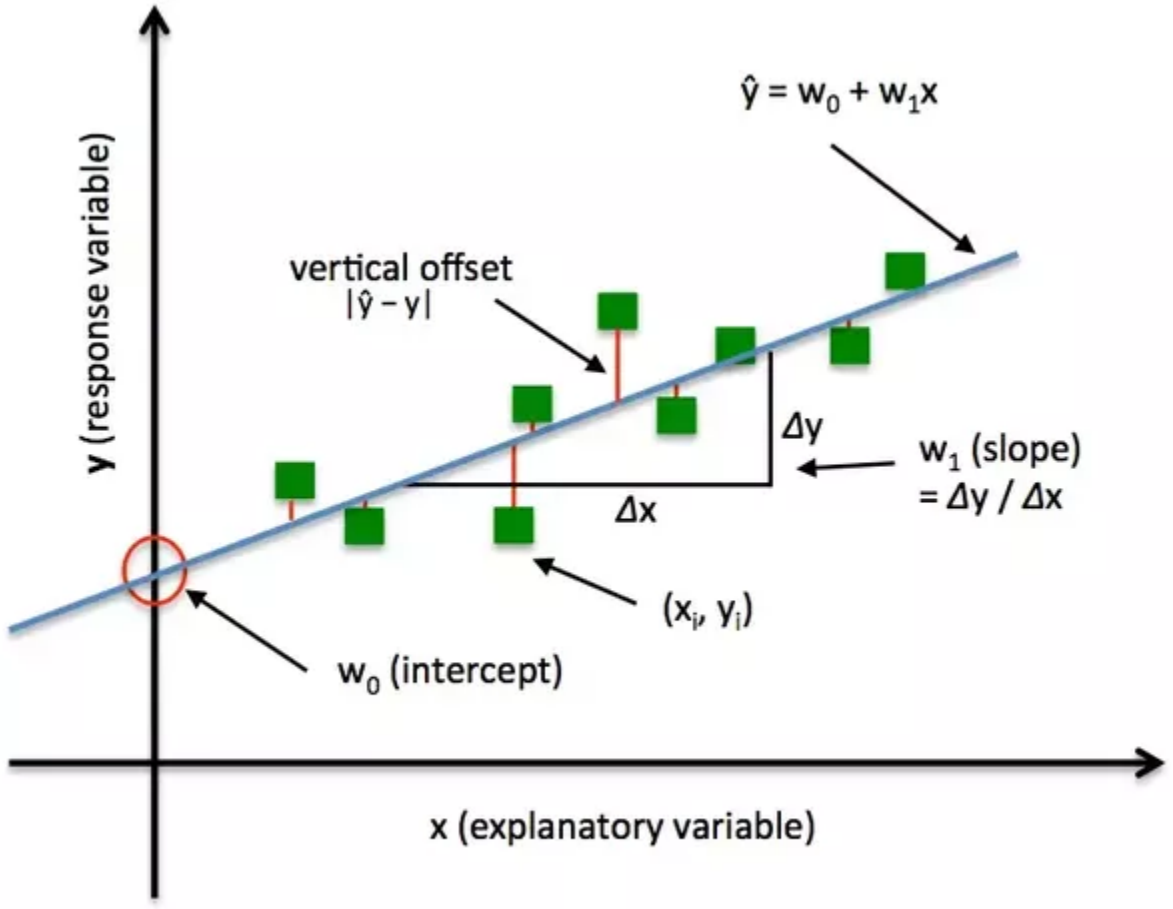


$$f(x) = \max(x, ax)$$

PReLU

음수값의 기울기는 a, 나머지는 그대로

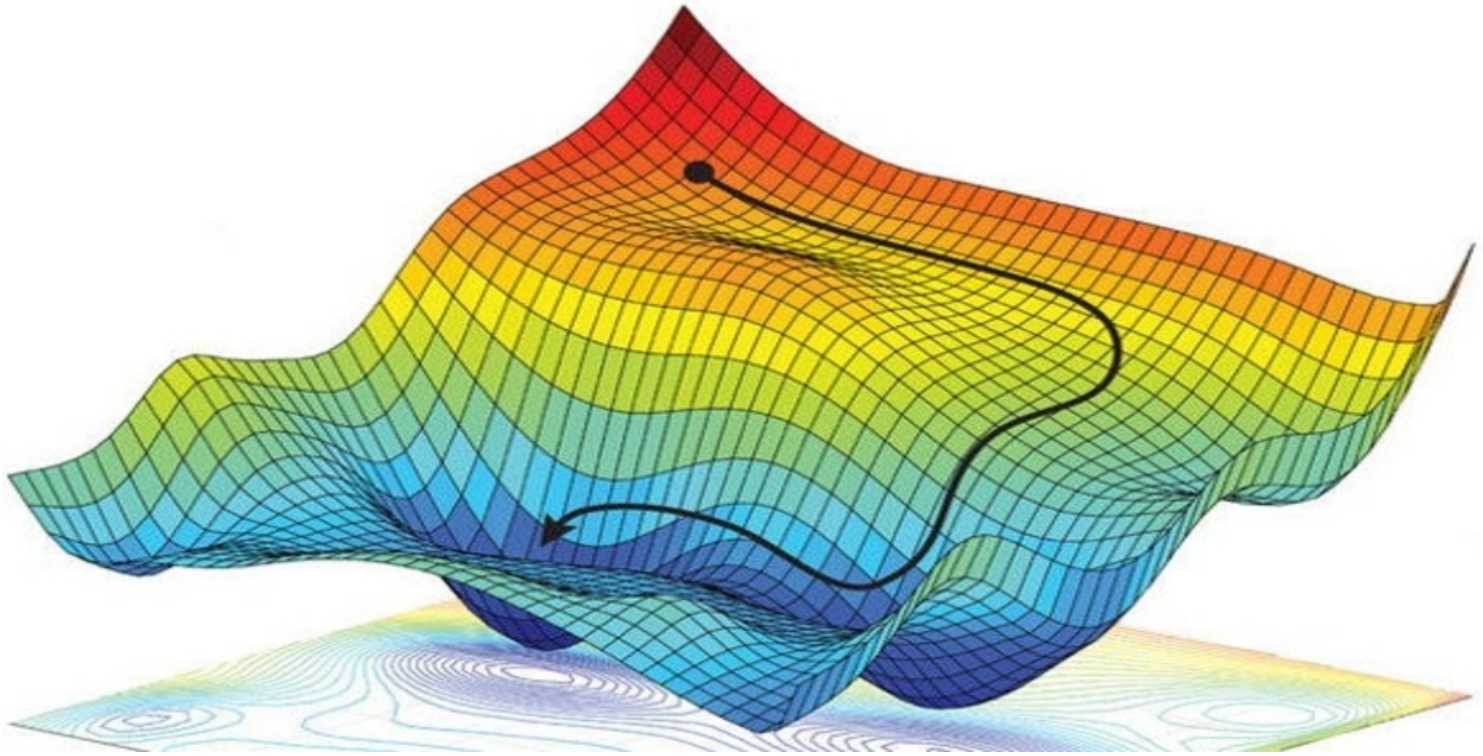
손실 함수와 지도학습



손실 함수 (Loss Func.)

모델의 예측값과 실제 값의 차이를 비교

손실 함수와 지도학습



손실 함수 (Loss Func.)

Gradient Descent Algorithm을 통해 가장 낮은 Loss를 찾는 것이 목표

손실 함수와 지도학습

Input[0]

```
import torch
import torch.nn as nn

mse_loss = nn.MSELoss()
outputs = torch.randn(3, 5, requires_grad=True)
targets = torch.randn(3, 5)
loss = mse_loss(outputs, targets)
print(loss)
```

Output[0]

```
tensor(3.8618)
```

$$L_{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2$$

Mean Squared Error

Regression 문제에서 사용하며, Euclidean Distance라고도 불림.

손실 함수와 지도학습

Example 3-8. Cross-entropy loss

Input[0]

```
import torch
import torch.nn as nn

ce_loss = nn.CrossEntropyLoss()
outputs = torch.randn(3, 5, requires_grad=True)
targets = torch.tensor([1, 0, 3], dtype=torch.int64)
loss = ce_loss(outputs, targets)
print(loss)
```

Output[0]

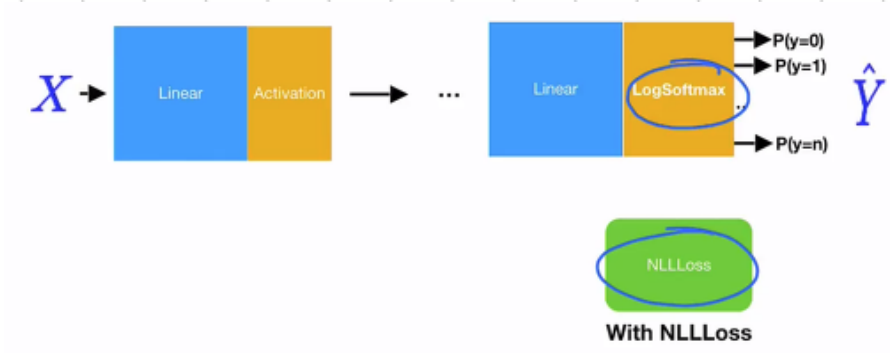
```
tensor(2.7256)
```

$$L_{cross\_entropy}\left(y,\hat{y}\right) = -\sum_i y_i \log\left(\hat{y}_i\right)$$

Cross Entropy

Classification 문제에서 사용되며, Information Theory에서 유도됨.

손실 함수와 지도학습



CROSSENTROPYLOSS

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

This criterion combines `LogSoftmax` and `NLLLoss` in one single class.

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

Cross Entropy

Softmax에서의 Exponential 연산은 연산범위가 크므로, 내부적으로 Log를 씌움.

손실 함수와 지도학습

Input[0]

```
bce_loss = nn.BCELoss()
sigmoid = nn.Sigmoid()
probabilities = sigmoid(torch.randn(4, 1, requires_grad=True))
targets = torch.tensor([1, 0, 1, 0],  dtype=torch.float32).view(4, 1
 oss = bce_loss(probabilities, targets)
print(probabilities)
print(loss)
```

Output[0]

```
tensor([[ 0.1625],
        [ 0.5546],
        [ 0.6596],
        [ 0.4284]])
tensor(0.9003)
```

$$L_{cross\_entropy}\left(y,\hat{y}\right)=-\sum_i y_i \log\left(\hat{y}_i\right)$$

Binary Cross Entropy

Softmax를 사용하지 않는 이진분류에서는 BCELoss를 사용

손실 함수와 지도학습

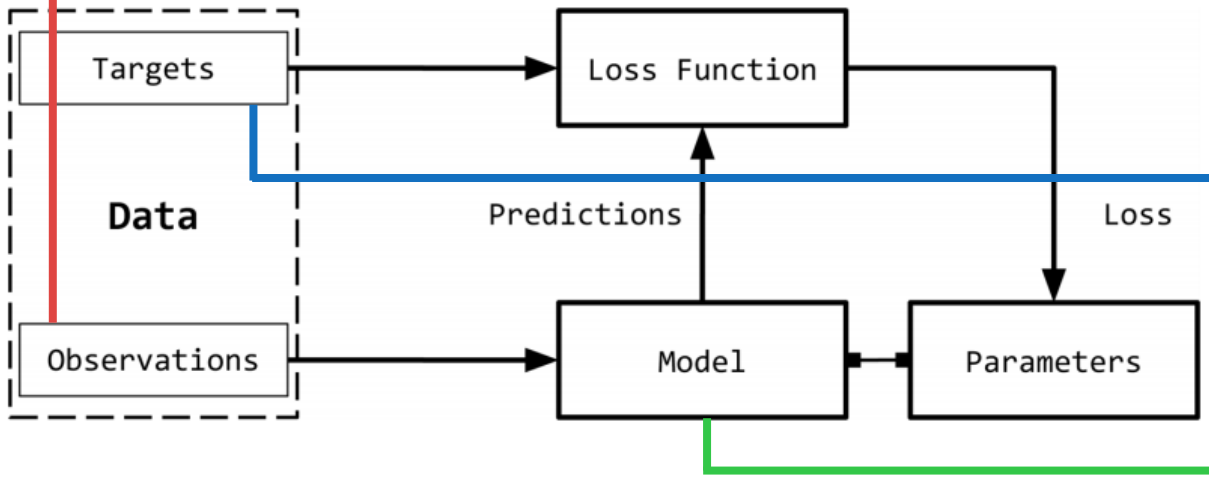


Figure 1-1. The supervised learning paradigm, a conceptual framework for learning from labeled input data.

샘플 (x, Observations)

예측에 사용하는 아이템으로, x로 표기한다.
샘플은 모델 학습의 '입력(input)'이 된다.

타깃 (y, Targets)

샘플에 상응하는 레이블으로, 예측되는 대상이다.
y로 표기하며, 학습의 '정답(Ground Truth)'가 된다.

모델 (f, Model)

수학식이나, 샘플 x를 받아 타깃 레이블값을 예측하는 함수이다.

Supervised Learning

지정된 타깃에 새로운 샘플을 매핑하는 방법을 학습하는 문제

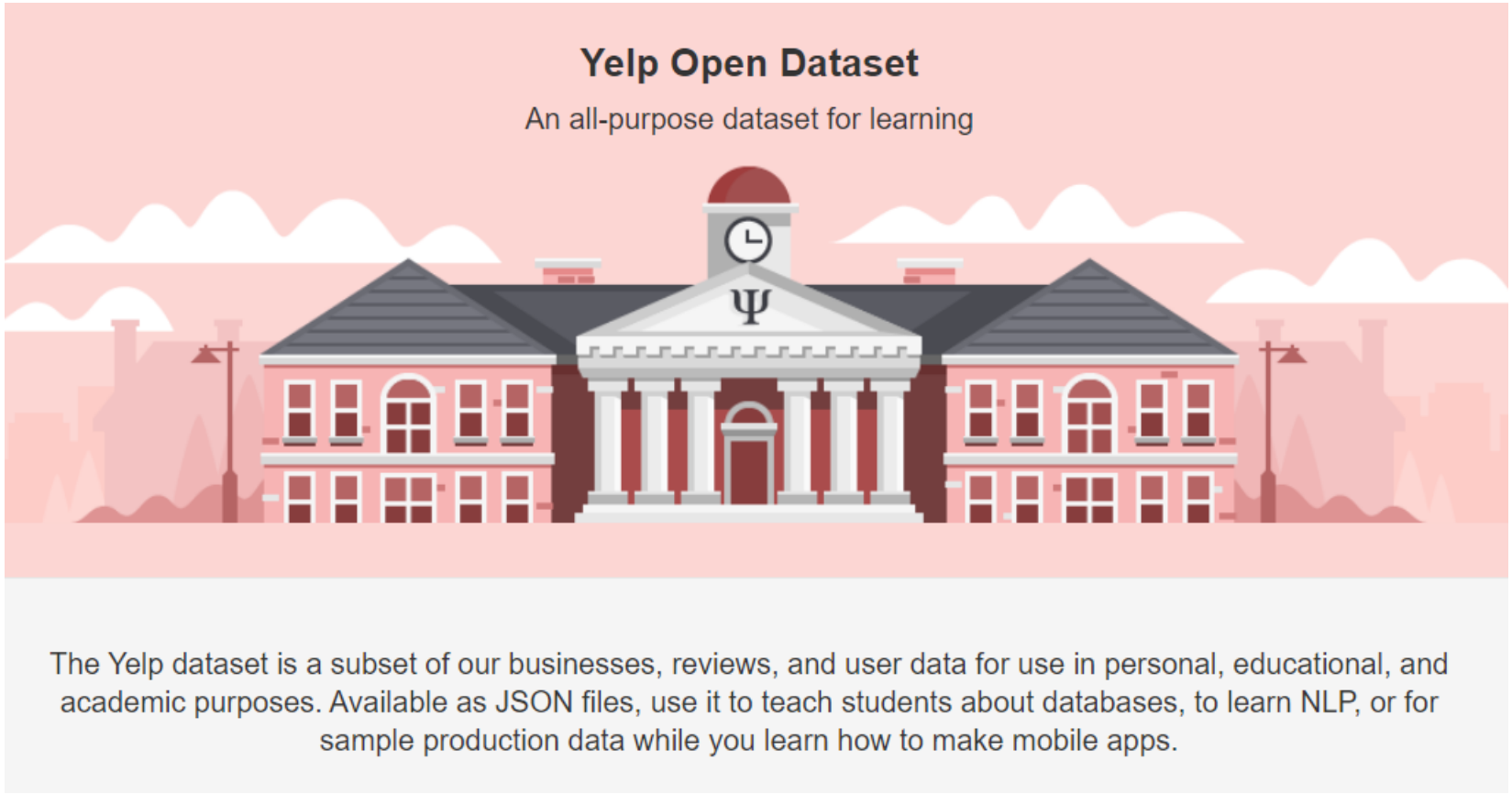
손실 함수와 지도학습

Notebook Link

Lets Code!

With Colab

레스토랑 리뷰 **감성 분류하기**



Yelp Dataset

리뷰와 관련된 데이터셋으로, 레스토랑 리뷰를 중점적으로 학습할 예정

레스토랑 리뷰 **감성 분류하기**

	rating	review
0	1	Unfortunately, the frustration of being Dr. Go...
1	2	Been going to Dr. Goldberg for over 10 years. ...
2	1	I don't know what Dr. Goldberg was like before...
3	1	I'm writing this review to give you a heads up...
4	2	All the food is great here. But the best thing...

Yelp Dataset

리뷰와 관련된 데이터셋으로, 레스토랑 리뷰를 중점적으로 학습할 예정

레스토랑 리뷰 **감성 분류하기**

```python
class Vocabulary(object):
    """ 매핑을 위해 텍스트를 처리하고 어휘 사전을 만드는 클래스 """

    def __init__(self, token_to_idx=None, add_unk=True, unk_token="<UNK>"):


    def to_serializable(self):
        """ 직렬화할 수 있는 딕셔너리를 반환합니다 """

    @classmethod
    def from_serializable(cls, contents):
        """ 직렬화된 딕셔너리에서 Vocabulary 객체를 만듭니다 """

    def add_token(self, token):
        """ 토큰을 기반으로 매핑 딕셔너리를 업데이트합니다 """


    def add_many(self, tokens):
        """ 토큰 리스트를 Vocabulary에 추가합니다. """

    def lookup_token(self, token):
        """ 토큰에 대응하는 인덱스를 추출합니다.
        토큰이 없으면 UNK 인덱스를 반환합니다. """

    def lookup_index(self, index):
        """ 인덱스에 해당하는 토큰을 반환합니다. """
```

Vocabulary

Vocabulary를 통해 문자열을 정수-토큰 매핑을 수행

레스토랑 리뷰 **감성 분류하기**

```python
class ReviewVectorizer(object):
    """ 어휘 사전을 생성하고 관리합니다 """
    def __init__(self, review_vocab, rating_vocab):

    def vectorize(self, review):
        """ 리뷰에 대한 윗-핫 벡터를 만듭니다 """

    @classmethod
    def from_dataframe(cls, review_df, cutoff=25):
        """ 데이터셋 데이터프레임에서 Vectorizer 객체를 만듭니다 """

    @classmethod
    def from_serializable(cls, contents):
        """ 직렬화된 딕셔너리에서 ReviewVectorizer 객체를 만듭니다 """

    def to_serializable(self):
        """ 캐싱을 위해 직렬화된 딕셔너리를 만듭니다 """
```

Vectorizer

Vocabulary를 캡슐화하고, 입력 문자열을 수치 벡터로 반환

레스토랑 리뷰 **감성 분류하기**

```python
class ReviewDataset(Dataset):
    def __init__(self, review_df, vectorizer):

    @classmethod
    def load_dataset_and_make_vectorizer(cls, review_csv):
        """ 데이터셋을 로드하고 새로운 ReviewVectorizer 객체를 만듭니다 """

    @classmethod
    def load_dataset_and_load_vectorizer(cls, review_csv, vectorizer_filepath):
        """ 데이터셋을 로드하고 새로운 ReviewVectorizer 객체를 만듭니다.
        캐시된 ReviewVectorizer 객체를 재사용할 때 사용합니다. """

    @staticmethod
    def load_vectorizer_only(vectorizer_filepath):
        """ 파일에서 ReviewVectorizer 객체를 로드하는 정적 메서드 """

    def save_vectorizer(self, vectorizer_filepath):
        """ ReviewVectorizer 객체를 json 형태로 디스크에 저장합니다 """

    def get_vectorizer(self):
        """ 벡터 변환 객체를 반환합니다 """

    def set_split(self, split="train"):
        """ 데이터프레임에 있는 열을 사용해 분할 세트를 선택합니다 """

    def __len__(self):

    def __getitem__(self, index):
        """ 파이토치 데이터셋의 주요 진입 메서드 """

    def get_num_batches(self, batch_size):
        """ 배치 크기가 주어지면 데이터셋으로 만들 수 있는 배치 개수를 반환합니다 """
```

Dataset

Vectorizer를 활용하여 PyTorch에서 사용할 수 있는 Dataset으로 구현

손실 함수와 지도학습

Notebook Link

Lets Code!

With Colab


Thank You