Deep structured learning in the cybersecurity domain [AICS323]
1D-CNN and BiLSTM and Practice (W04)

Prof. Mee Lan Han (aeternus1203@gmail.com)

고려대학교

인공지능사이버보안학과

KOREA UNIVERSITY

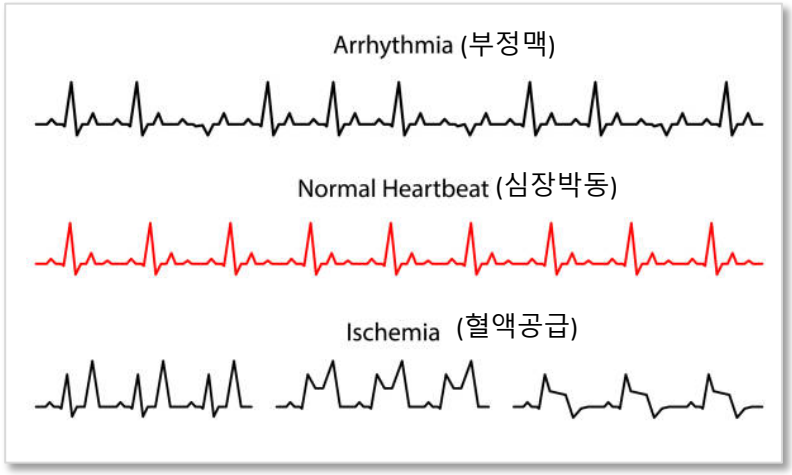
CONTENTS

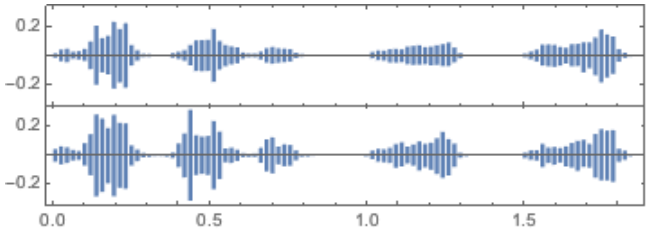
Time series CNN

■ Time series CNN

□ What is time series data?

- Time series data 를 Time-stamped data라고도 함
- 시간 순서대로 색인된 일련의 데이터 포인트





□ CNN in time seires data

- **Convolution은 일반적으로 image 같은 2D 데이터와 공간 데이터에 활용**됨
- 1D sequence 데이터에 대한 Convolution 연산도 수행함
  - **1D sequence는 실제로는 Convolution 연산이 아닌 Correlation 연산 수행**임

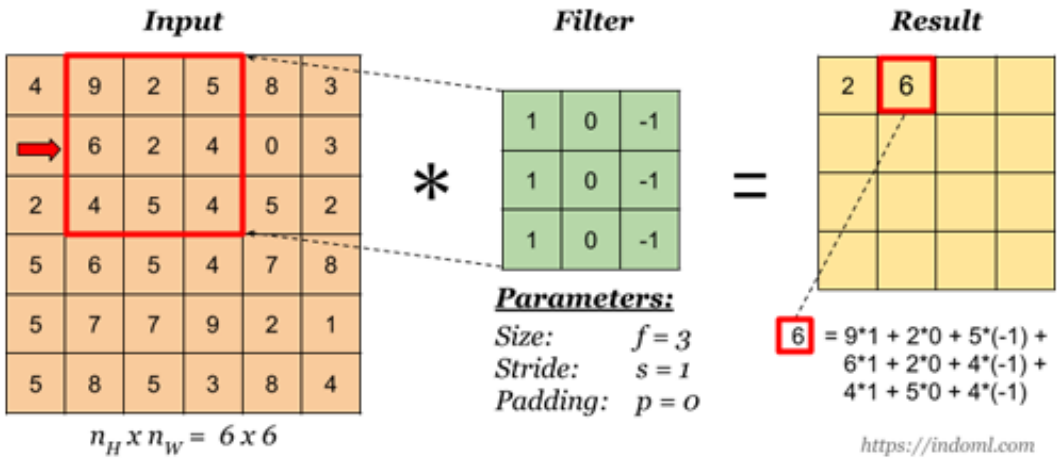
Time series CNN

■ Time series CNN

□ CNN in time series data

- 1D sequence: Convolution 연산이 아닌 **Correlation 연산 수행**임

1) 영상처리에서 Filter 를 적용하기 위해서는 합성곱(Convolution) 연산 방법을 사용하게 됨



원본 이미지에서 픽셀 선택 후, 픽셀 범위만큼 원본 이미지와 Filter를 곱하고,
모두 더한 값을 하나의 픽셀 결과값으로 도출함

Time series CNN

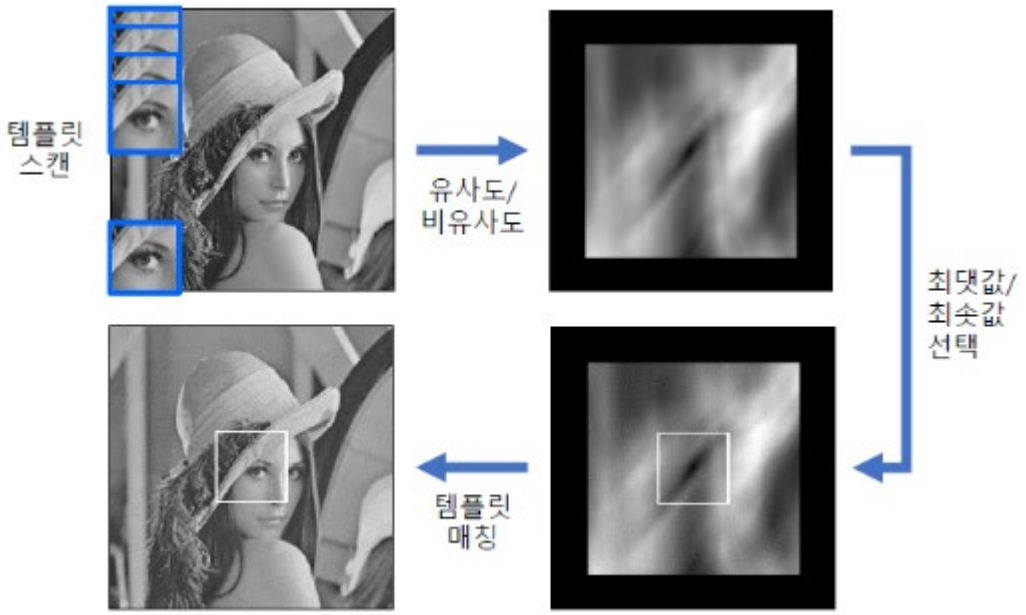
■ Time series CNN

□ CNN in time series data

2) 상관연산 (**Correlation**)은 Filter를 있는 그대로 합성곱 연산을 수행함

- Convolution은 주로 필터링과 특징 추출과 관련이 있으며, **Correlation은 주로 템플릿 매칭**과 관련

- 템플릿 매칭(Template matching)
  - 입력 영상에서 (작은 크기의) 템플릿 영상과 일치하는 부분을 찾는 기법
  - 템플릿: 찾을 대상이 되는 작은 영상. 패치(patch)



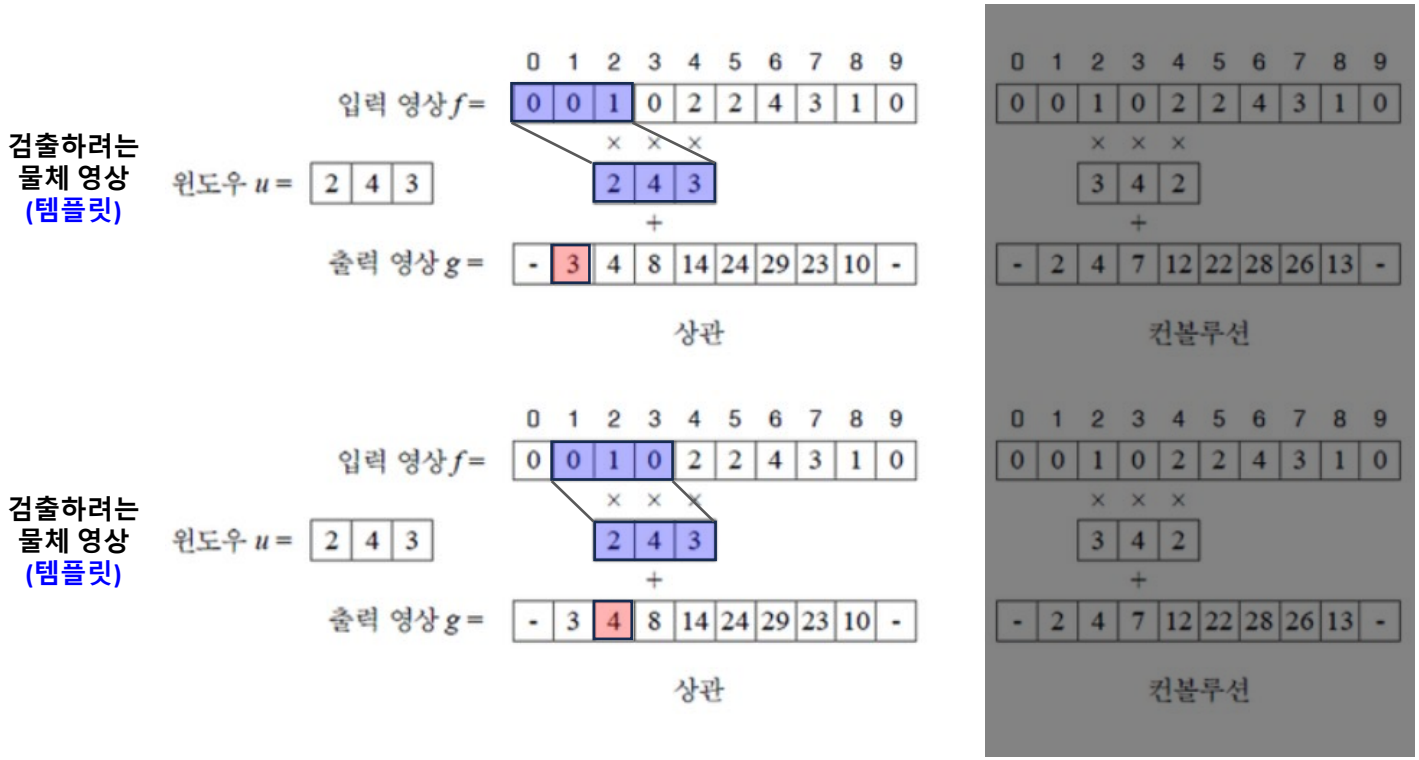
Time series CNN

■ Time series CNN

□ CNN in time series data

2) 상관연산 (**Correlation**)은 Filter를 있는 그대로 합성곱 연산을 수행함

- Convolution은 주로 필터링과 특징 추출과 관련이 있으며, **Correlation은 주로 템플릿 매칭**과 관련
- Convolution은 커널을 뒤집어서 사용하고, Correlation은 뒤집지 않고 직접 사용



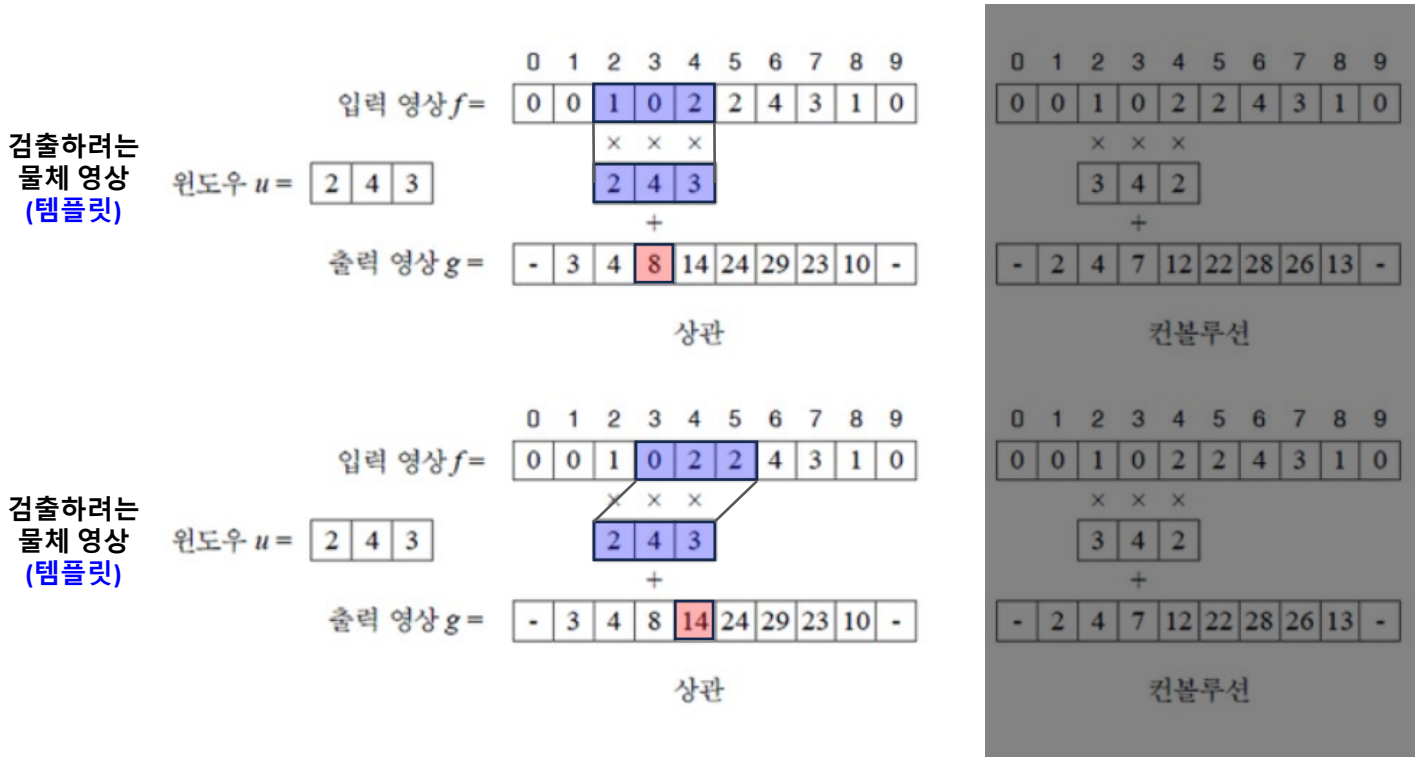
Time series CNN

■ Time series CNN

□ CNN in time series data

2) 상관연산 (**Correlation**)은 Filter를 있는 그대로 합성곱 연산을 수행함

- Convolution은 주로 필터링과 특징 추출과 관련이 있으며, **Correlation은 주로 템플릿 매칭**과 관련
- Convolution은 커널을 뒤집어서 사용하고, Correlation은 뒤집지 않고 직접 사용



Time series CNN

■ Time series CNN

□ Time series data 에 CNN을 먼저 고민하는 이유?

- CNN은 RNN보다 sequential operation (순차 계산)이 적어 계산 효율성이 더 높은 편임
- 즉, 같은 **계산 비용으로 더 긴 history를 track할 수 있음**

Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$

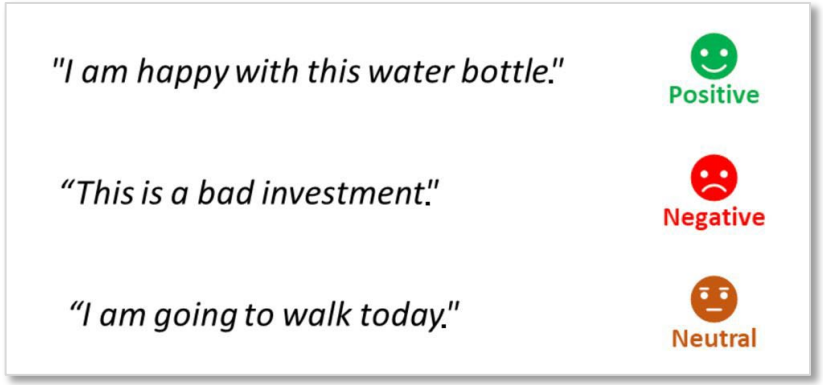
- RNN: time series의 '모든' point는 '모든' 이전 timestep에 의존한다는 가정이 필요
- CNN: time window 내에서 pattern을 학습하므로 해당 가정이 필요하지 않음
- 시간 정보가 누락된 데이터인 경우 유리함 (time dependence가 RNN보다 낮음)
  - (문장에서 단어를 추측하는 데에 있어 CNN은 문장의 전체 텍스트를 알 필요 없음)

Time series CNN

■ Time series CNN

□ When to use CNN?

- CNN (1D Convolutions; Temporal Convolutions)
  - Sentiment Analysis (감정 분석) ex) 영화 리뷰가 **긍정 / 부정**인지 분류
  - 텍스트에서도 유의미한 feature를 추출하며, 단어 수준의 NLP Task인 경우 RNN보다 CNN이 성능이 좋음
  - **Convolution window 내에 특정 지역의 local pattern을 효율적으로 학습**함



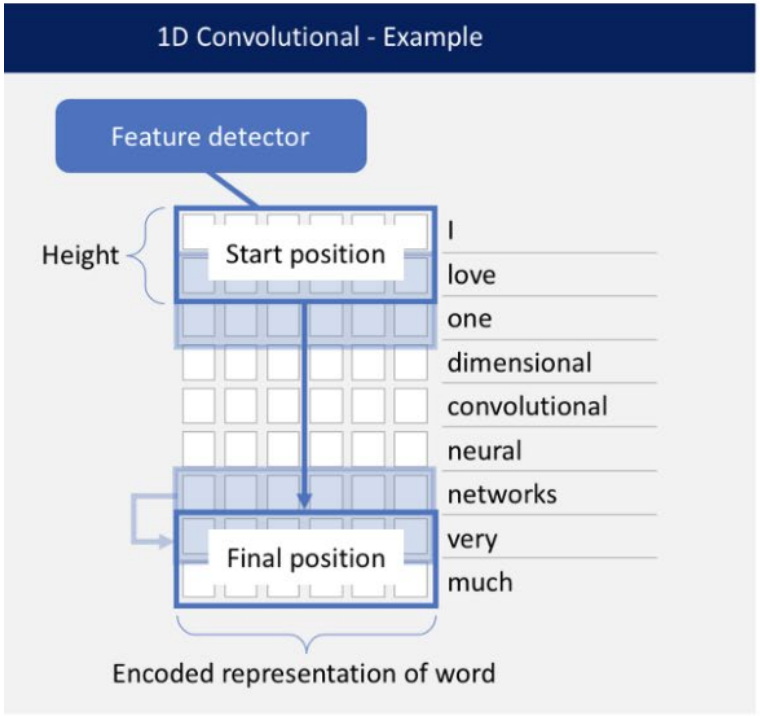
- RNN
  - Machine Translation (기계 번역) ex) **한국어 ↔ 영어 번역**
  - 전체 words sequence가 있을 때 유용하며, 단어들 간의 관계와 순서를 CNN보다 더 잘 기억함

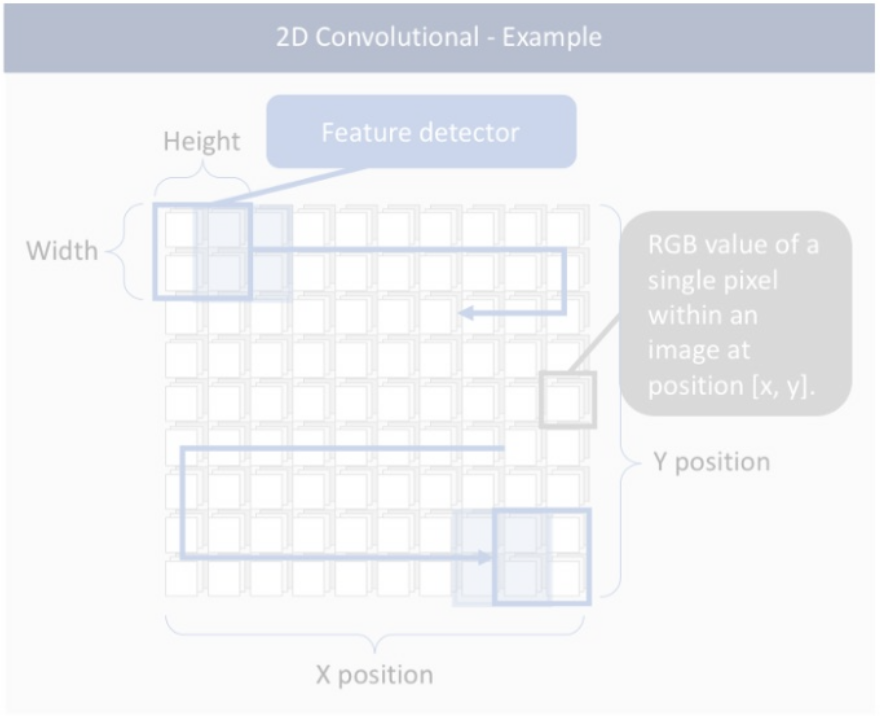
Time series CNN

■ Time series CNN

□ 1D-CNN vs. 2D-CNN

- 시계열 데이터를 다룰 때 1D CNN이 적합
  (**1차원 형태의 데이터의 상관관계를 효과적으로 추출할 수 있는 구조**)
- 데이터 내에서 단순한 패턴을 식별하는 데 좋고 Time sequence 분석에 유용함
- 변수 간의 지엽적인 특징을 추출할 수 있음 (커널 슬라이딩을 통해 공간적 상관 관계를 추출)





[참조] https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

Time series CNN

■ Time series CNN

□ 1D-CNN vs. 2D-CNN

- x축과 y축의 공간 관계를 고려
- **2차원(2D) 형태의 데이터의 상관관계를 효과적으로 추출할 수 있는 구조**
  (2D 형태의 공간적 특징을 가지는 영상데이터)

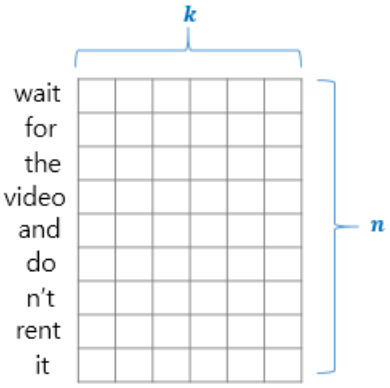
[참조] https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

Time series CNN

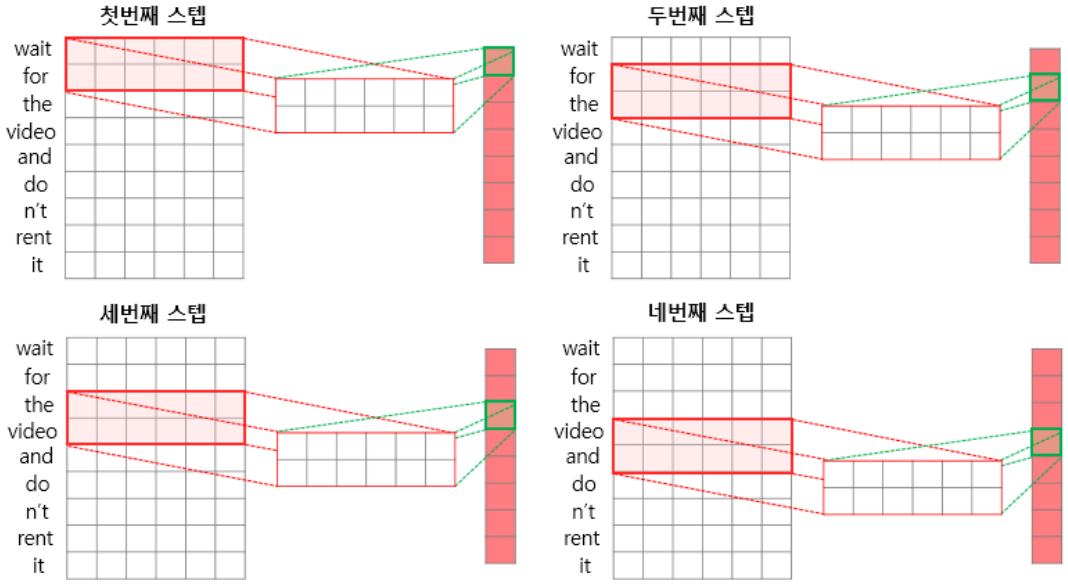
■ Time series CNN

□ 1D-CNN

- 1차원 데이터를 처리하는 CNN의 한 종류 (**하나의 축만 고려하는 1D 필터를 사용**)
- 일반적으로 **1차원 값 시퀀스로 표시**되는 데이터의 1D 합성곱 연산 (숫자 데이터, 텍스트 데이터)
- 수치나 텍스트 처리에서는 **filter가 위/아래 한 방향으로만 이동**하는 1D Convolution을 사용
- **커널의 크기에 따라서 참고하는 단어의 묶음의 크기가 달라짐**



n은 문장의 길이,
k는 임베딩 벡터의 차원의 행렬



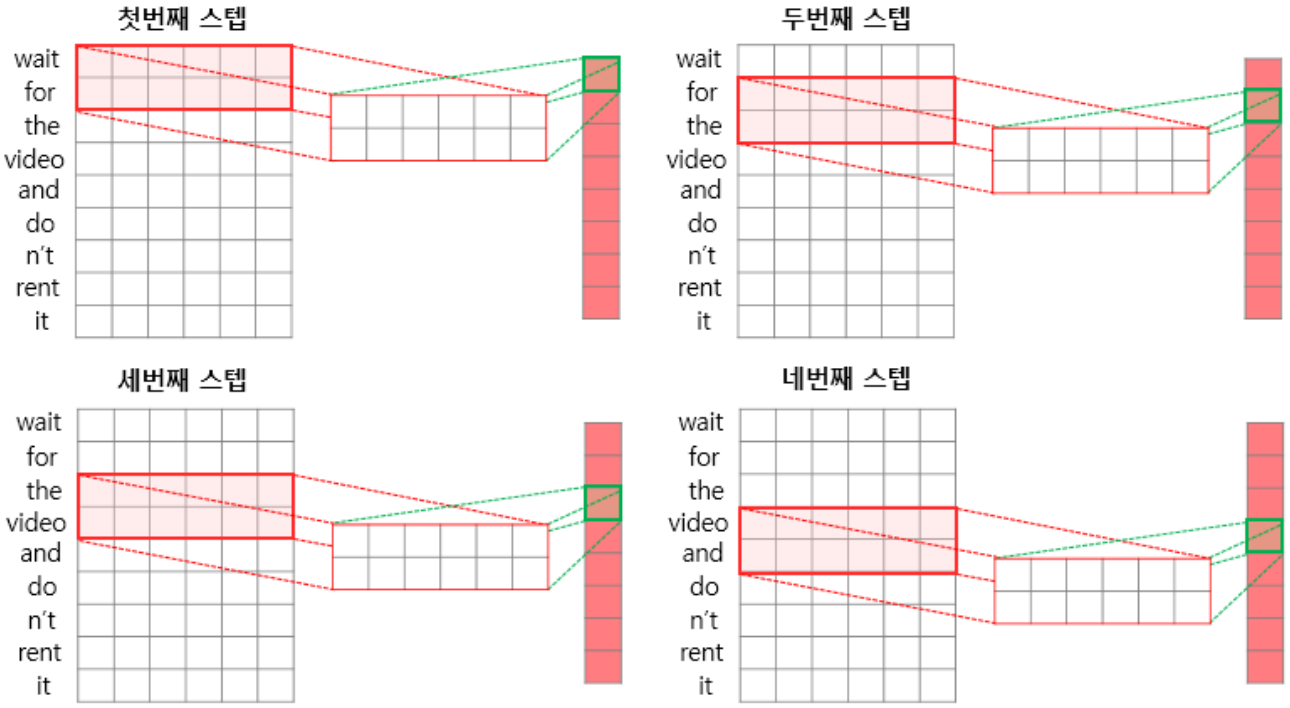
Time series CNN

■ Time series CNN

□ 1D-CNN

- 1D 합성곱 연산에서 커널의 너비는 문장 행렬에서의 임베딩 벡터의 차원과 동일하게 설정됨
- 1D 합성곱 연산에서는 커널의 높이만으로 해당 커널의 크기라고 간주
- 커널의 크기가 2인 경우 >>
  - 높이가 2, **너비가 임베딩 벡터의 차원**인 커널이 사용 (하단 그림)

너비 방향으로는 더 이상 움직일 곳이 없음 -> 1D 합성곱 연산에서는 커널이 문장 행렬의 높이 방향으로만 움직임



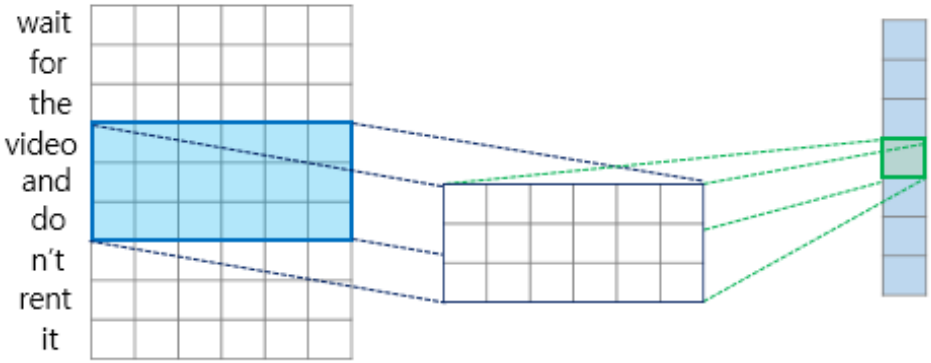
Time series CNN

■ Time series CNN

□ 1D-CNN

- 커널 크기 2인 경우
  - 여덟 번째 스텝까지 반복한 후, 결과적으로는 우측의 8차원 벡터를 1D 합성곱 연산의 결과로서 얻게됨

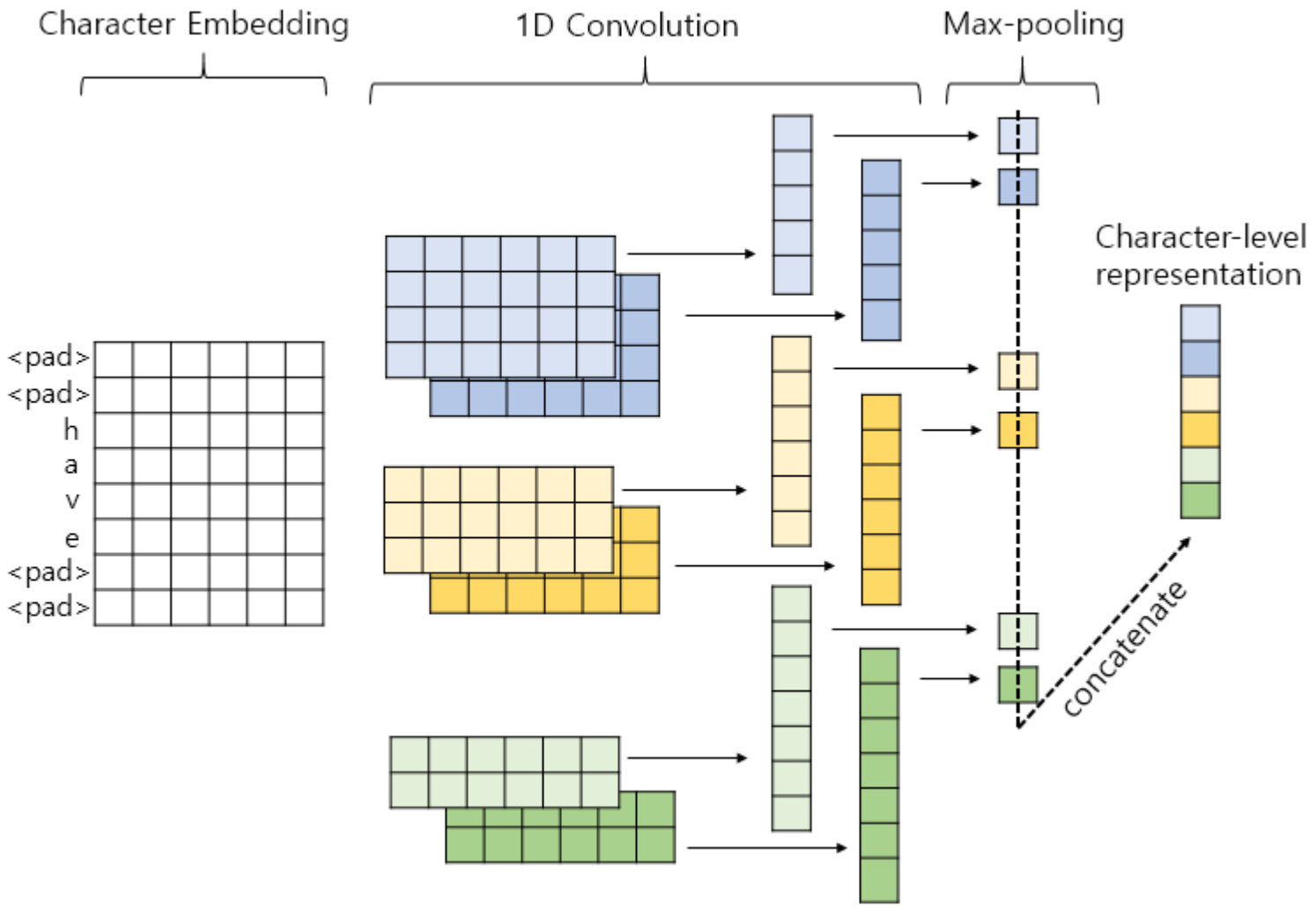
- 커널 크기 3인 경우
  - 1D 합성곱 연산에서도 커널의 크기는 사용자가 변경할 수 있음
  - 가령, 커널의 크기 3인 경우, 네번째 스텝에서의 연산은 아래의 그림과 같음



Time series CNN

■ Time series CNN

□ 1D-CNN 설계



Time series CNN

■ Time series CNN

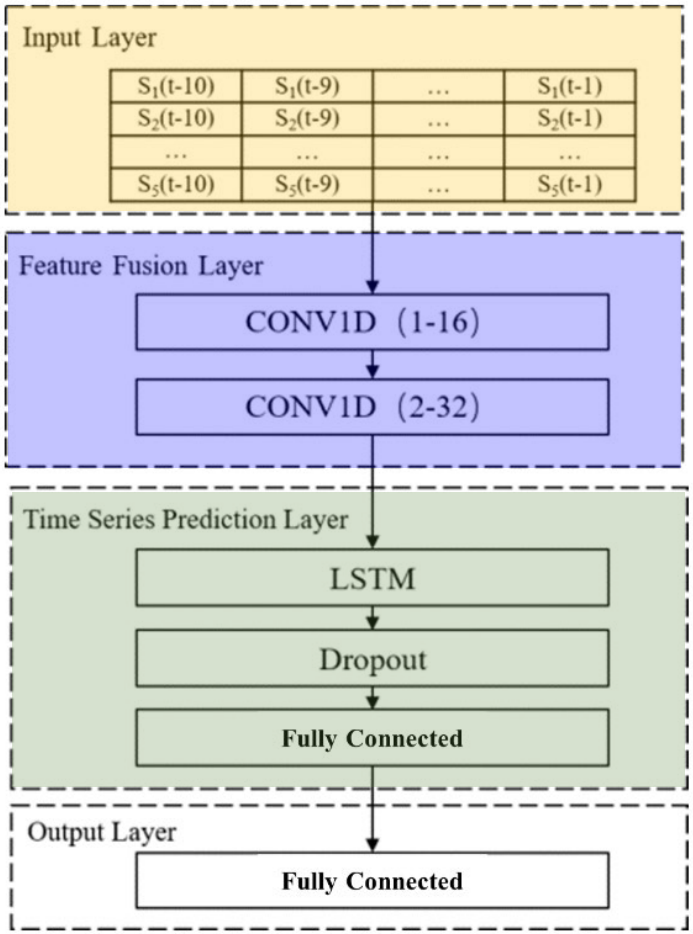
□ 1D-CNN 모델

```
from tensorflow.keras.layers import Dense, Conv1D, GlobalMaxPooling1D, Embedding, Dropout, MaxPooling1D
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

embedding_dim = 32
dropout_ratio = 0.3
num_filters = 32
kernel_size = 5

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(Dropout(dropout_ratio))
model.add(Conv1D(num_filters, kernel_size, padding='valid', activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dropout(dropout_ratio))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
mc = ModelCheckpoint('best_model.h5', monitor = 'val_acc', mode='max', verbose=1, save_best_only=True)

history = model.fit(X_train_padded, y_train, epochs=10, batch_size=64, validation_split=0.2, callbacks=[es, mc])

X_test_encoded = tokenizer.texts_to_sequences(X_test)
X_test_padded = pad_sequences(X_test_encoded, maxlen = max_len)
print("\n 테스트 정확도: %.4f" % (model.evaluate(X_test_padded, y_test)[1]))
```

Time series CNN

■ Time series CNN

□ 1D-CNN + LSTM 모델 (1/2)



- Input layer에는 그림 0과 같은 시공간적 특성행렬이 입력

- 유효 특징 추출을 담당
- 원본 데이터에서 벡터를 추출하고 원본 기능의 공간적 정보를 마이닝함
- 서로 다른 변수를 결합하고 변수 간의 공간적 상관 관계를 추출

- LSTM 기반 시계열 예측 레이어
- 과적합을 피하기 위해 dropout layer가 네트워크에 추가

Time series CNN

■ Time series CNN

□ 1D-CNN + LSTM 모델 (2/2)

```
import torch.nn as nn

class Conv1d_LSTM(nn.Module):

    def __init__(self, in_channel=3, out_channel=1)
        super(Conv1d_LSTM, self).__init__()
        self.conv1d_1 = nn.Conv1d(in_channels=in_channel,
                                  out_channels=16,
                                  kernel_size=3,
                                  stride=1,
                                  padding=1)
        self.conv1d_2 = nn.Conv1d(in_channels=16,
                                  out_channels=32,
                                  kernel_size=3,
                                  stride=1,
                                  padding=1)

        self.lstm = nn.LSTM(input_size=32,
                            hidden_size=50,
                            num_layers=1,
                            bias=True,
                            bidirectional=False,
                            batch_first=True)

        self.dropout = nn.Dropout(0.5)

        self.dense1 = nn.Linear(50, 32)
        self.dense2 = nn.Linear(32, out_channel)
```

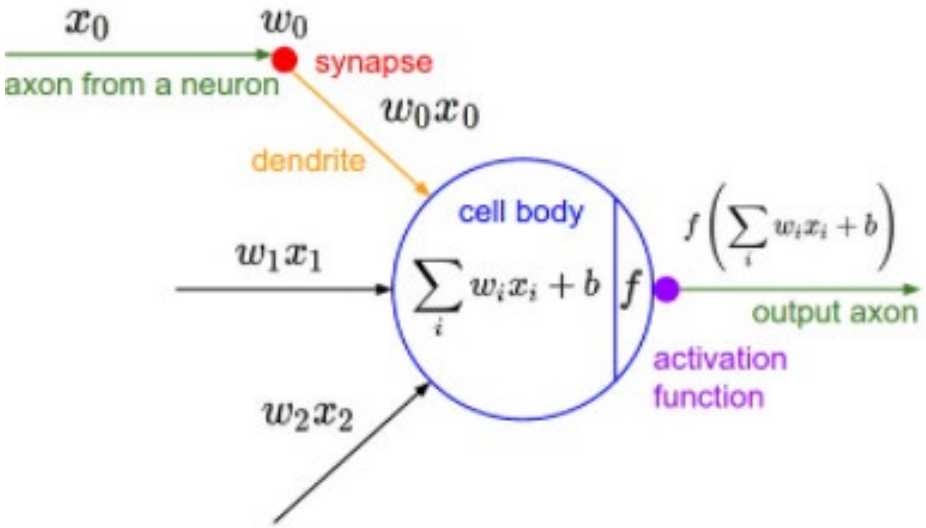
```
import torch.nn as nn

class Conv1d_LSTM(nn.Module):

    def forward(self, x):
        # Raw x shape : (B, S, F) => (B, 10, 3)

        # Shape : (B, F, S) => (B, 3, 10)
        x = x.transpose(1, 2)
        # Shape : (B, F, S) == (B, C, S) // C = channel => (B, 16, 10)
        x = self.conv1d_1(x)
        # Shape : (B, C, S) => (B, 32, 10)
        x = self.conv1d_2(x)
        # Shape : (B, S, C) == (B, S, F) => (B, 10, 32)
        x = x.transpose(1, 2)

        self.lstm.flatten_parameters()
        # Shape : (B, S, H) // H = hidden_size => (B, 10, 50)
        _, (hidden, _) = self.lstm(x)
        # Shape : (B, H) // -1 means the last sequence => (B, 50)
        x = hidden[-1]

        # Shape : (B, H) => (B, 50)
        x = self.dropout(x)

        # Shape : (B, 32)
        x = self.fc_layer1(x)
        # Shape : (B, O) // O = output => (B, 1)
        x = self.fc_layer2(x)

        return x
```

Performance improvement

■ Activation Function

□ 활성화 함수

- 입력 신호의 총합을 출력 신호로 변환하는 함수를 일반적으로 활성화 함수라고 함
- 입력 신호의 총합이 활성화를 일으키는지를 정하는 역할
- 가중치(w)가 달린 입력 신호(x)와 편향(b)의 총합을 계산하고 함수 f에 넣어 출력하는 흐름



Performance improvement

■ Activation Function

□ 활성화 함수 역할

- 신경망의 활성화 함수는 비선형 함수를 사용
- 선형 함수인 활성화 함수를 이용하면 여러층으로 구성하는 이점을 살릴 수 없음
- 딥러닝에서 층을 쌓기 위해서는 비선형 함수인 활성화 함수를 사용해야 함

- **선형함수?**
  - ✓ 출력이 입력의 상수 배만큼 변화는 함수(1개의 곧은 직선)
  - ✓ 신경망에서 선형 함수를 이용하면 신경망의 층을 깊게하는 의미가 없어짐
  - ✓ 선형함수 사용 시, 층을 아무리 깊게해도 은닉층이 없는 네트워크로도 똑같은 기능을 할 수 있음

  예) $h(x) = cx$ 를 활성화 함수로 사용한 3층 신경망 → $y(x)= h(h(h(x)))$

  이 계산은 $y(x)=c*c*c*x$ 를 수행하지만 사실 $y(x)=ax$ 와 동일함

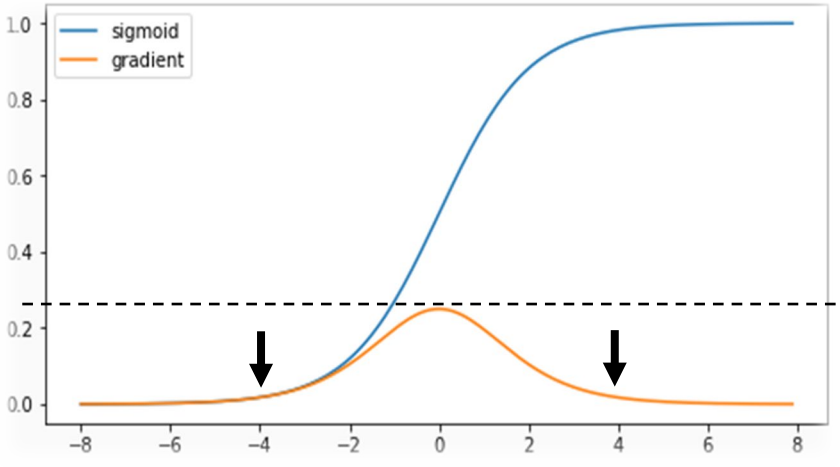
  → 은닉층이 없는 네트워크로 표현 가능

Performance improvement

■ Activation Function

(1) Sigmoid 활성함수

- 실수 값을 입력받아 0~1 사이의 값으로 압축
- 큰 음수 값일 수록 0에 가까워지고 큰 양수 값일 수록 1에 가까움
- 시그모이드 함수의 미분 최대치는 0.3, 즉, 1보다 작은 값
- 은닉층이 많아질수록 기울기는 점점 0에 가까워지게 됨



시그모이드 함수를 미분
한 값 최대치 0.3

- 활성화 (activation)들이 [−4,4] 범위에 들어가지 않게 되면 전체 곱의 기울기는 소멸
- 시그모이드 함수를 대체할 수 있는 활성화 함수를 사용하려면 시그모이드 함수와 모양이 비슷하고 경사가 소실되지 않는 함수를 사용 -> **ReLU 함수사용**

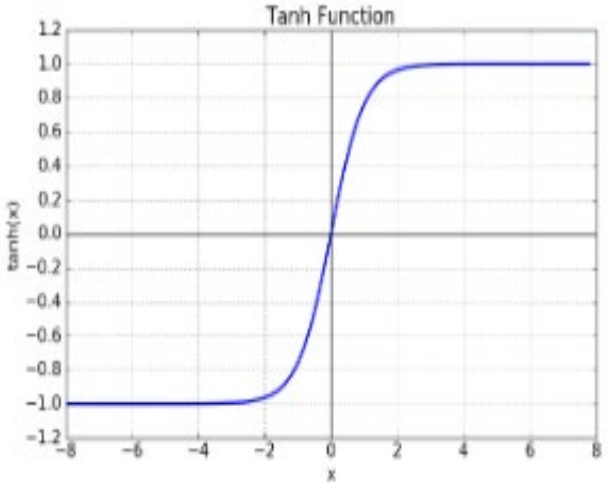
Performance improvement

■ Activation Function

(2-1) Tanh 활성함수 & ReLU (Rectified Linear Unit) 활성함수

□ 시그모이드와 비슷하지만 실수 값을 입력받아 -1~1 사이의 값으로 압축

□ tanh(x) 함수 (시그모이드와 비슷) 를 사용하면 미분값의 범위가 확장



□ 고차원 데이터를 다룰 경우에는 값이 커질 수 있어 다시 경사가 소실될 수 있음

□ 복잡한 데이터일수록 고차원일 경우가 많은데 이를 회피할 수 있는 활성화 함수가 ReLU 함수

Performance improvement

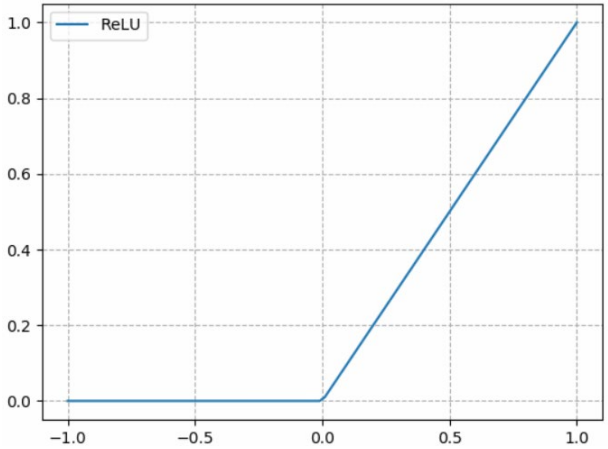
■ Activation Function

(2-2) Tanh 활성함수 & ReLU (Rectified Linear Unit) 활성함수

□ ReLU 함수의 도함수는 x가 아무리 커져도 1을 반환하므로 경사가 소실되지 않음
□ 시그모이드 함수나 쌍곡탄젠트 함수에 비교해 학습 속도가 빠름
□ ReLU와 ReLU의 도함수는 단순한 식으로 표현되기 때문에 빠르게 계산이 가능

$$\varphi(\mathbf{x}) = \begin{cases} \mathbf{x}, \mathbf{x} > \mathbf{0} \\ \mathbf{0}, x \leq \mathbf{0} \end{cases}$$

$$\varphi(x) = \max(\mathbf{0}, \mathbf{x})$$



$$f'(x) = \begin{cases} \mathbf{1}, \mathbf{x} > \mathbf{0} \\ \mathbf{0}, x \leq \mathbf{0} \end{cases}$$

ReLU 함수의 도함수 (미분한 식)

- x (입력값)가 0보다 작을 때 (음수)는 모든 값을 0으로 출력
- x (입력값)가 0보다 클 때 (양수)는 입력값을 그대로 출력
- **x가 0보다 크기만 하면 미분값이 1**이 되기 때문에 여러 은닉층을 거쳐도 맨 처음 층까지 사라지지 않고 남아있음
- 딥러닝의 발전에 기여!!!

Performance improvement

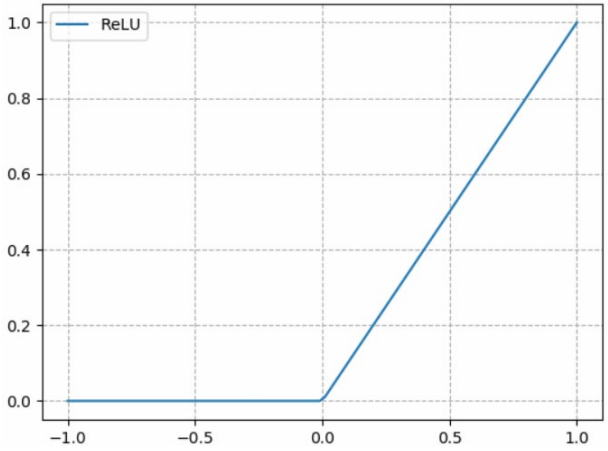
■ Activation Function

(2-2) Tanh 활성함수 & ReLU (Rectified Linear Unit) 활성함수

□ ReLU 함수의 도함수는 x가 아무리 커져도 1을 반환하므로 경사가 소실되지 않음
□ 시그모이드 함수나 쌍곡탄젠트 함수에 비교해 학습 속도가 빠름
□ ReLU와 ReLU의 도함수는 단순한 식으로 표현되기 때문에 빠르게 계산이 가능

$$\varphi(\mathbf{x}) = \begin{cases} \mathbf{x}, \mathbf{x} > \mathbf{0} \\ \mathbf{0}, x \leq \mathbf{0} \end{cases}$$

$$\varphi(x) = \max(\mathbf{0}, \mathbf{x})$$



$$f'(x) = \begin{cases} \mathbf{1}, \mathbf{x} > \mathbf{0} \\ \mathbf{0}, x \leq \mathbf{0} \end{cases}$$

ReLU 함수의 도함수 (미분한 식)

- 단점으로는 x≦0일 때는 함수값도 경사도 0이기 때문에 ReLU를 활성화 함수로 사용한 신경망 모델의 뉴런 중 활성화되지 못한 뉴런은 학습 동안 활성화가 되지 않는 문제가 있음

Performance improvement

■ Activation Function

(3) Softmax 활성화 함수

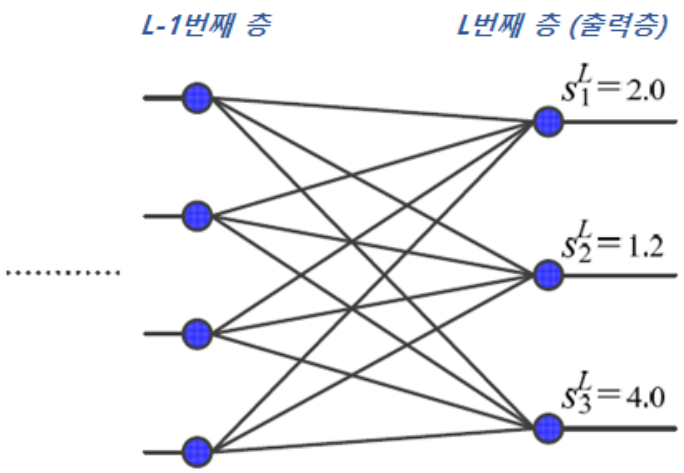
□ max를 모방 (출력 노드의 중간 계산 결과 s_i^L 에서 최댓값은 더욱 활성화하고 작은 값은 억제)

□ 모두 더하면 1이 되어 확률 모방

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$ *n: 출력층의 뉴런수(총 클래스 수), x 는x 번째 클래스*

$$softmax(x) = \left[\frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}}, \frac{e^{x_2}}{e^{x_1}+e^{x_2}+e^{x_3}}, \frac{e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}} \right] = [p_1, p_2, p_3]$$

총 클래스 수가 3개일 때, 각 각 [x번째 일 확률 / 전체확률]



Sigmoid	max	softmax
0.8808	0.0	0.1131
0.7685	0.0	0.0508
0.9820	1.0	0.8360

Thank you

KOREA
UNIVERSITY