순환신경망(RNN)

명지대학교
MYONGJI UNIVERSITY

순환 신경망(RNN)

RNN 개요

순환 신경망(Recurrent Neural Network)이란?

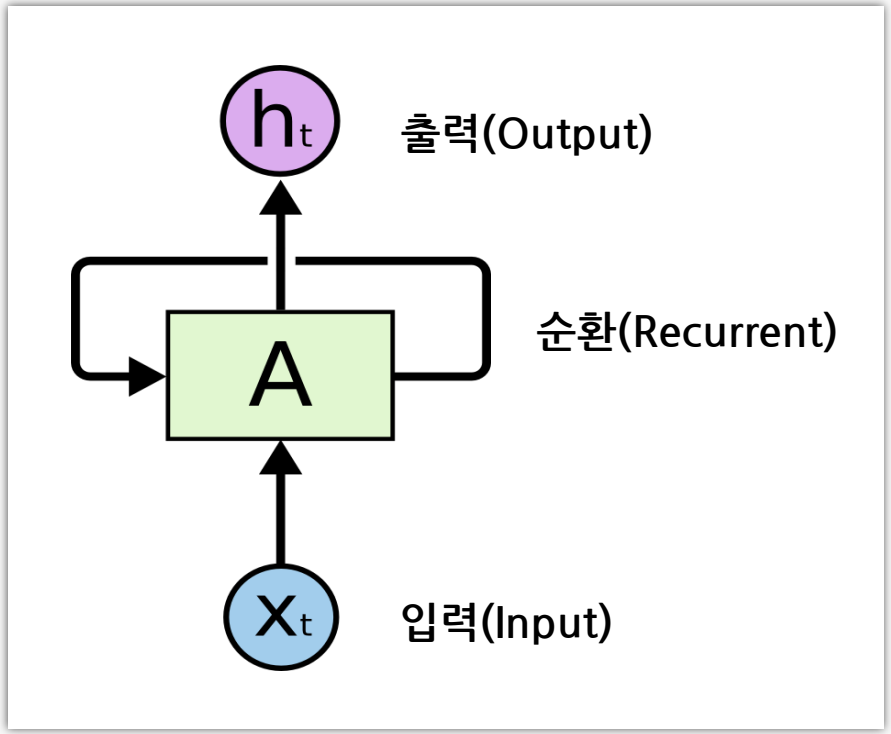
네트워크 내부에 순환(Recurrent) 구조를 포함함으로써
정보의 지속성 확보

음성, 문자 등 시퀀스(Sequence) 데이터나
시계열(Time Series) 데이터 처리에 적합

순환 신경망(RNN)

RNN 구성

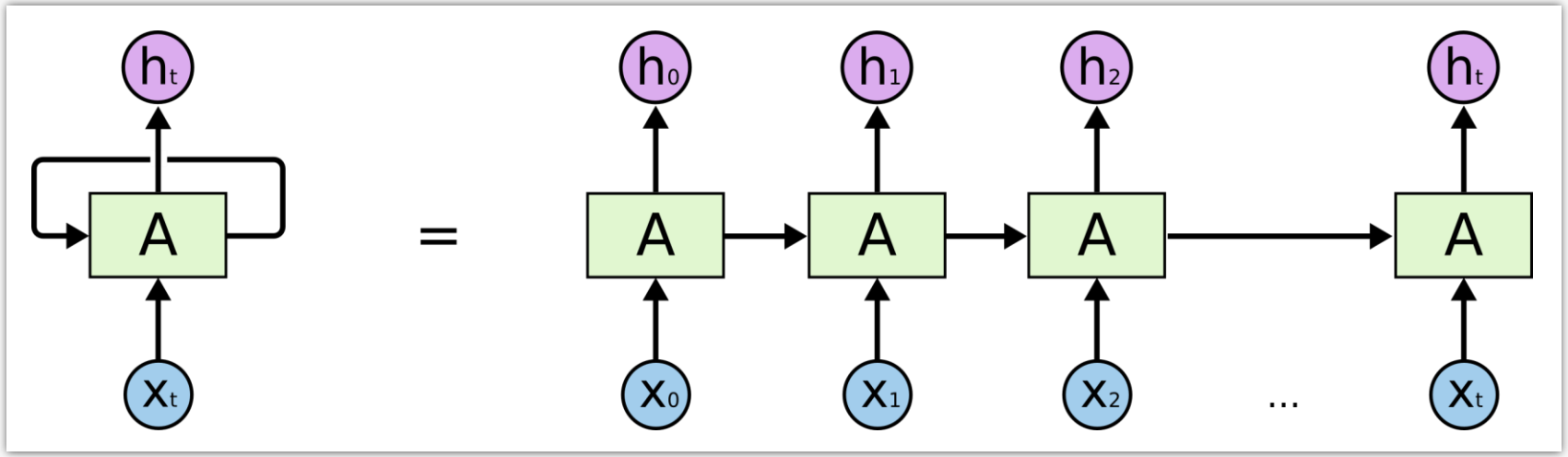
순환 신경망은 입력, RNN Cell의 순환 구조, 출력 등으로 구성



〈출처: http://colah.github.io. 20. 9〉

RNN 입출력

입력	X_0 X_1 X_2 … X_t 시퀀스(Sequence) 데이터나 시계열(Time Series) 데이터를 주로 사용
출력	h_0 h_1 h_2 … h_t 등 은닉 상태(Hidden State)의 출력과 다음 Cell로의 출력 두 가지 경로(Path)가 존재

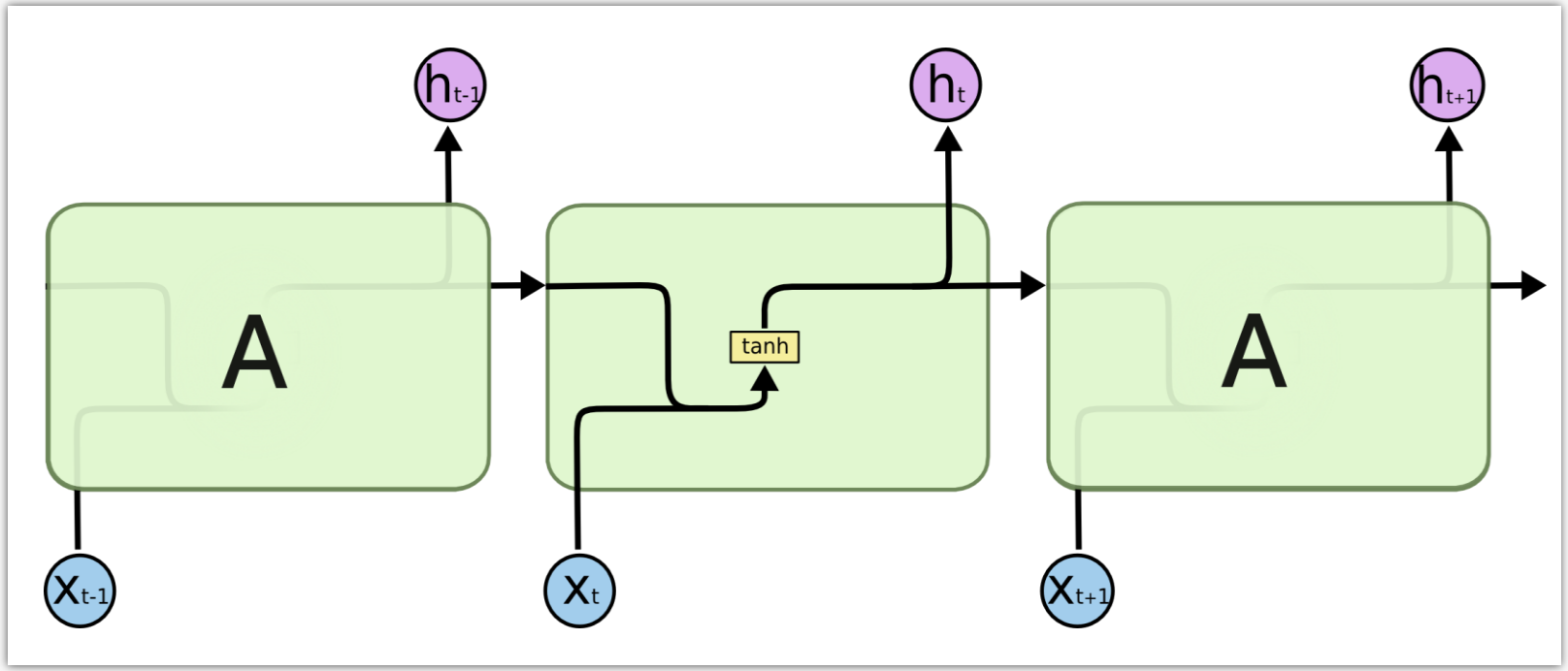


〈출처: http://colah.github.io. 20. 9〉

순환 신경망(RNN)

RNN의 구조

이전 Cell의 출력과 현재 Cell 입력을 tanh 활성화 함수로 처리한 후
현재 Cell의 출력과 다음 Cell의 입력으로 출력

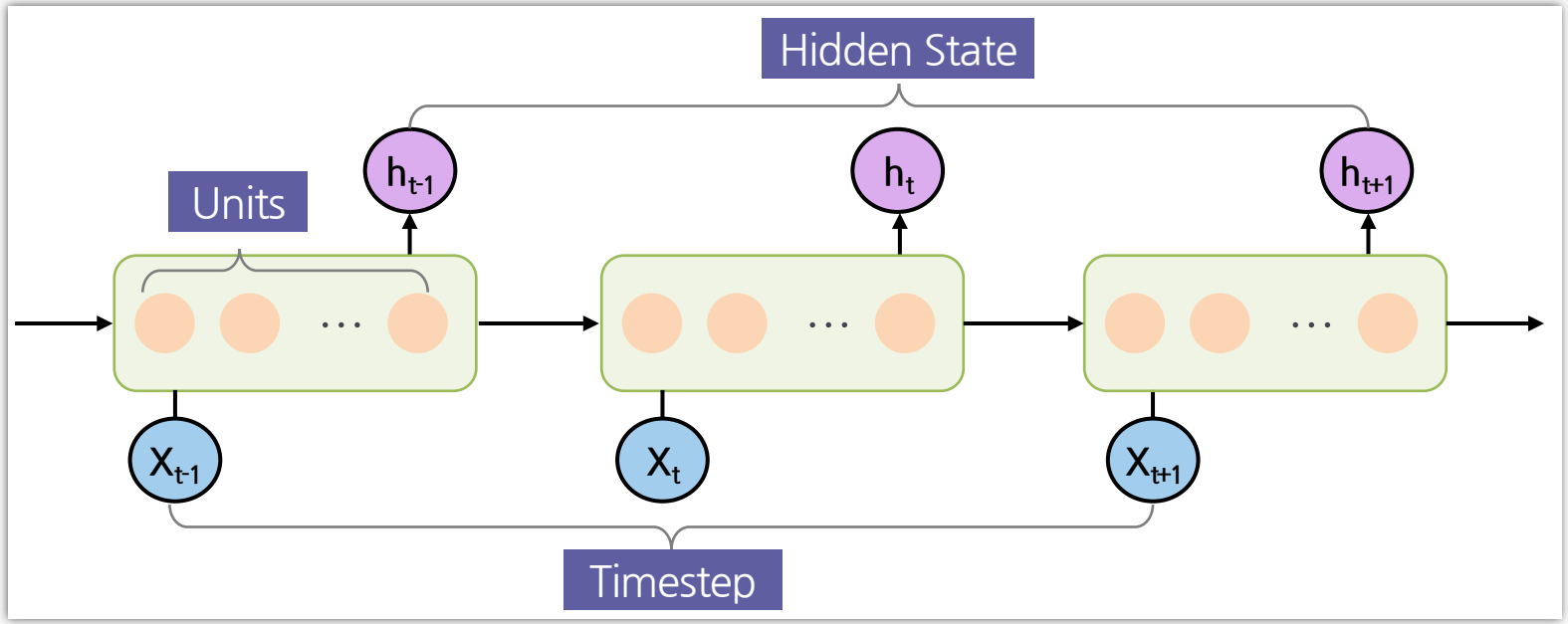


〈출처: http://colah.github.io. 20. 9〉

순환 신경망(RNN)

RNN의 구조

Timestep	RNN 입력데이터로 전달 되는 시퀀스(Sequence)
(Hidden)Units	RNN Cell 내의 Hidden 노드 수
Hidden State	각 Timestep의 입력에 대한 RNN Cell 출력



순환 신경망(RNN)

RNN의 아키텍처 유형별 활용

RNN의 활용 유형에 따라
단일 입력 다중 출력, 다중 입력 단일 출력 및 다중 입력 다중 출력 등 존재

RNN
One to Many
아키텍처

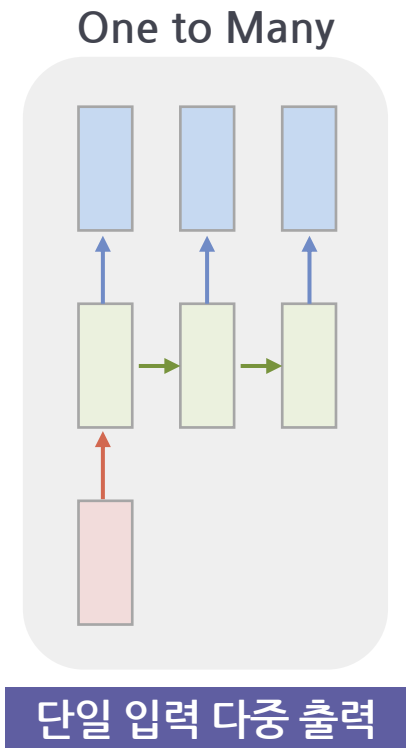
RNN
Many to One
아키텍처

RNN
Many to Many
아키텍처

RNN의 아키텍처 유형별 활용

RNN
One to Many
아키텍처

- 단일 입력에 대해 시퀀스(Sequence) 출력
- 하나의 영상 이미지로부터 캡션(Caption) 데이터 생성

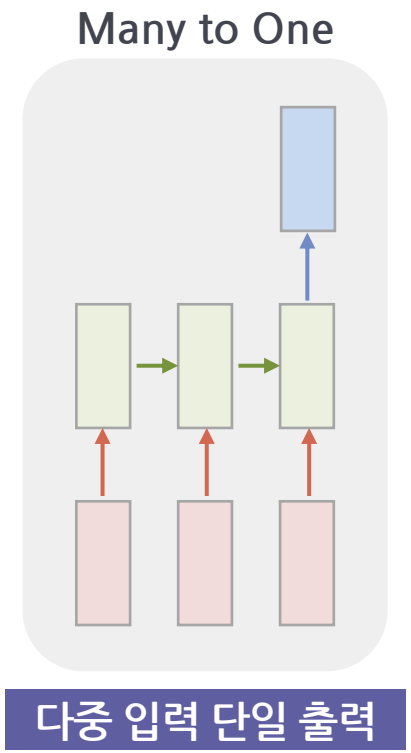


〈출처: http://cs231n.stanford.edu. 20. 9〉

RNN의 아키텍처 유형별 활용

RNN
Many to One
아키텍처

- 시퀀스(Sequence) 입력에 대해 단일 출력
- 영화 리뷰 등에 대한 감성 분석(Sentiment Analysis)

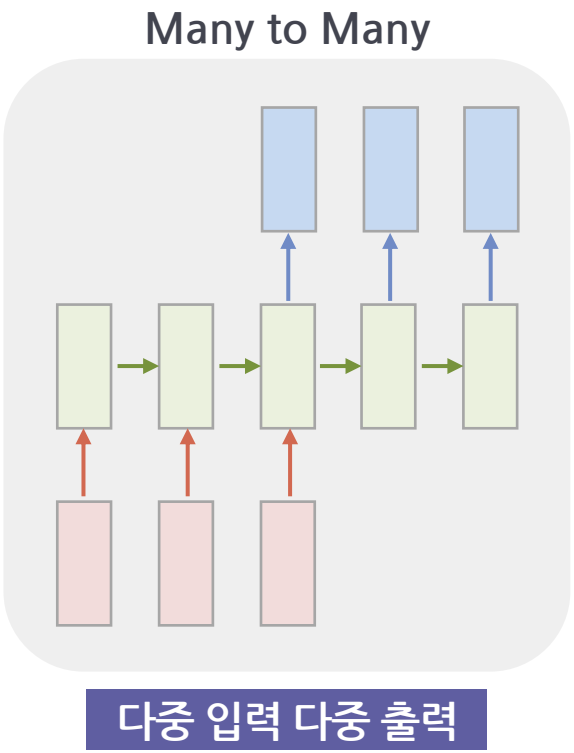


〈출처: http://cs231n.stanford.edu. 20. 9〉

RNN의 아키텍처 유형별 활용

RNN
Many to Many
아키텍처

- 시퀀스(Sequence) 입력에 대해 시퀀스(Sequence) 출력
- 인공지능 번역기술(Machine Translation) 등에 활용



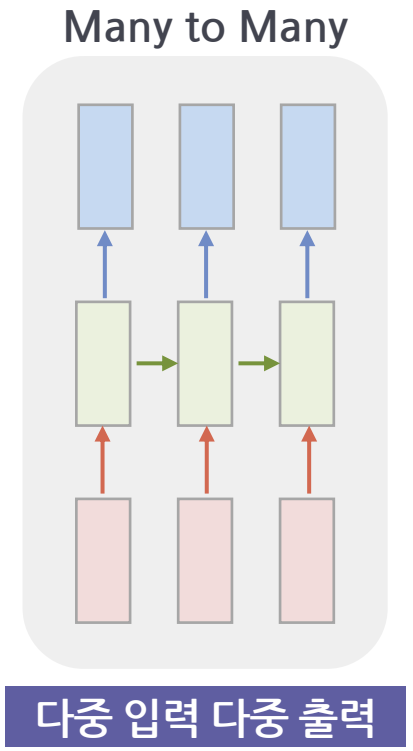
〈출처: http://cs231n.stanford.edu. 20. 9〉

순환 신경망(RNN)

RNN의 아키텍처 유형별 활용

RNN
Many to Many
아키텍처

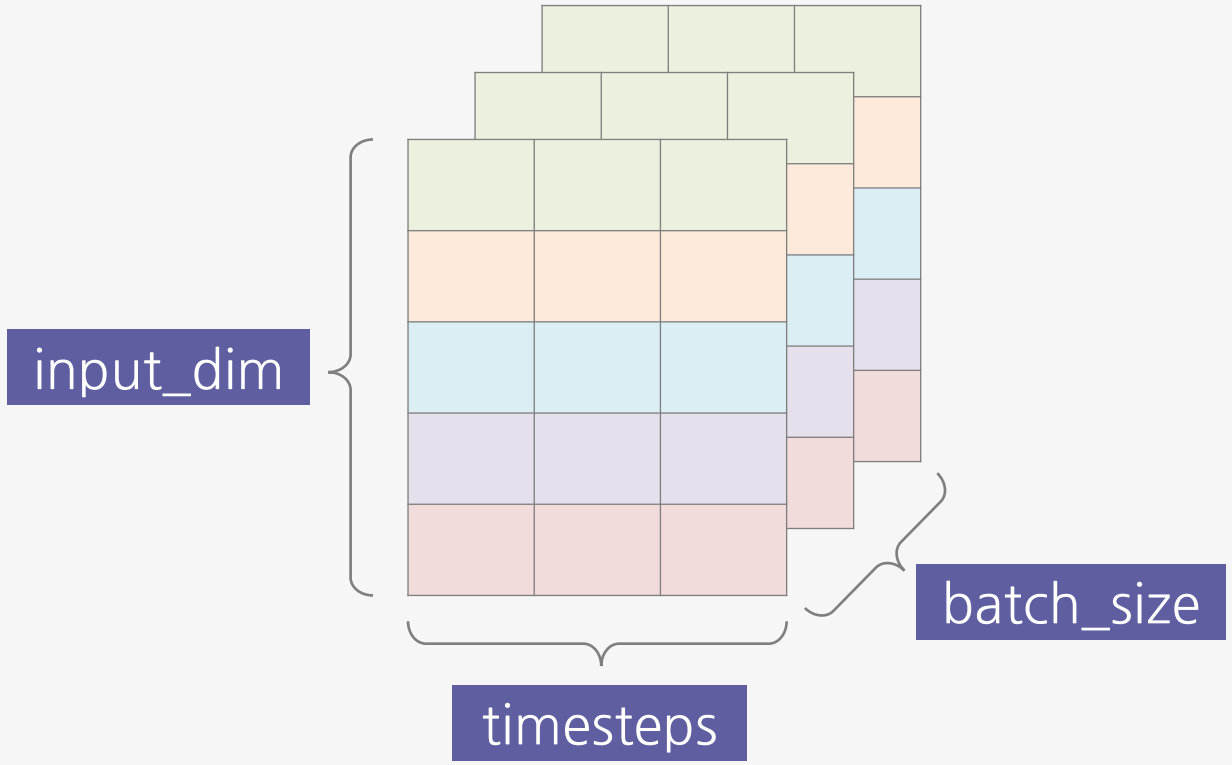
- 시퀀스(Sequence) 입력에 대해 시퀀스(Sequence) 출력
- 비디오 영상에 자동 Caption 생성 등에 활용



〈출처: http://cs231n.stanford.edu. 20. 9〉

RNN의 적용 방법

RNN 입력 데이터는 batch size, timesteps, input dimension 등으로 구성



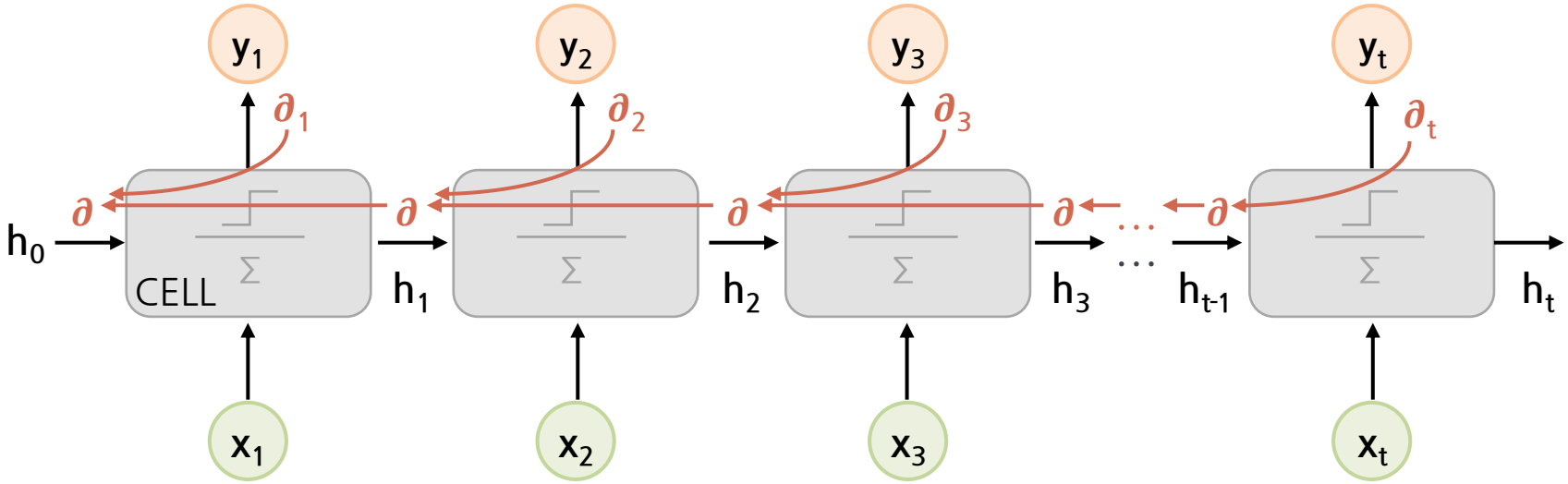
RNN의 개선 모델

기본(Vanilla) RNN의 문제점

RNN의 역전파 BPTT와 Gradient Vashing

- RNN에서의 역전파 방법인 BPTT(Back Propagation Through Time)는 Timstep의 마지막 단계에서 시작 단계까지 역전파 진행

역전파 진행 과정에서 Gradient Vanishing 발생 가능



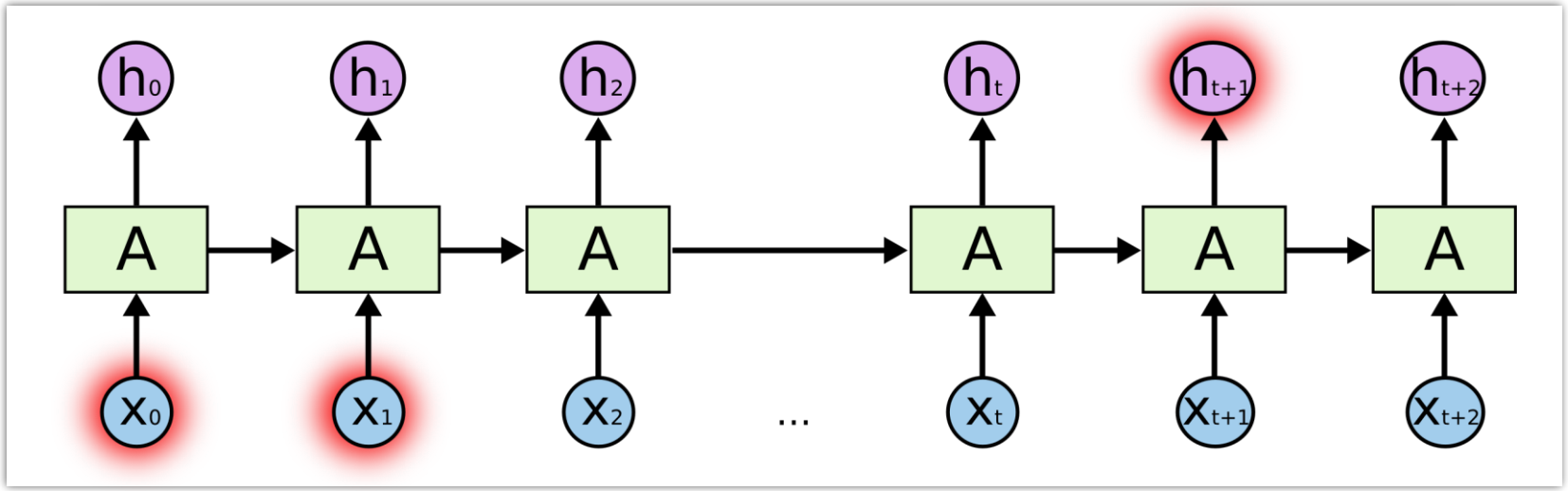
<출처: https://excelsior-cjh.tistory.com/185. 20. 9>

RNN의 개선 모델

기본(Vanilla) RNN의 문제점

장기 의존성 문제(The Problem of Long-Term Dependencies)

- 관련 정보와 그 정보가 필요한 곳의 거리가 먼 경우, 장기 Memory에 대한 학습능력 저하
- Time step의 크기가 큰 경우, Vanilla RNN은 성능 저하



<출처: http://colah.github.io. 20. 9>

RNN의 개선 모델

기본(Vanilla) RNN의 문제점

RNN의 역전파 BPTT와
Gradient Vashing

장기 의존성 문제
(The Problem of Long-Term
Dependencies)

Vanilla RNN의 문제점 개선을 위한 LSTM, GRU 등의 등장

RNN의 개선 모델

LSTM 개요

LSTM(Long Short Term Memory)이란?

Vanilla RNN의 Gradient Vashing 문제와 장기 의존성 문제 해결을 위해 LSTM 등장

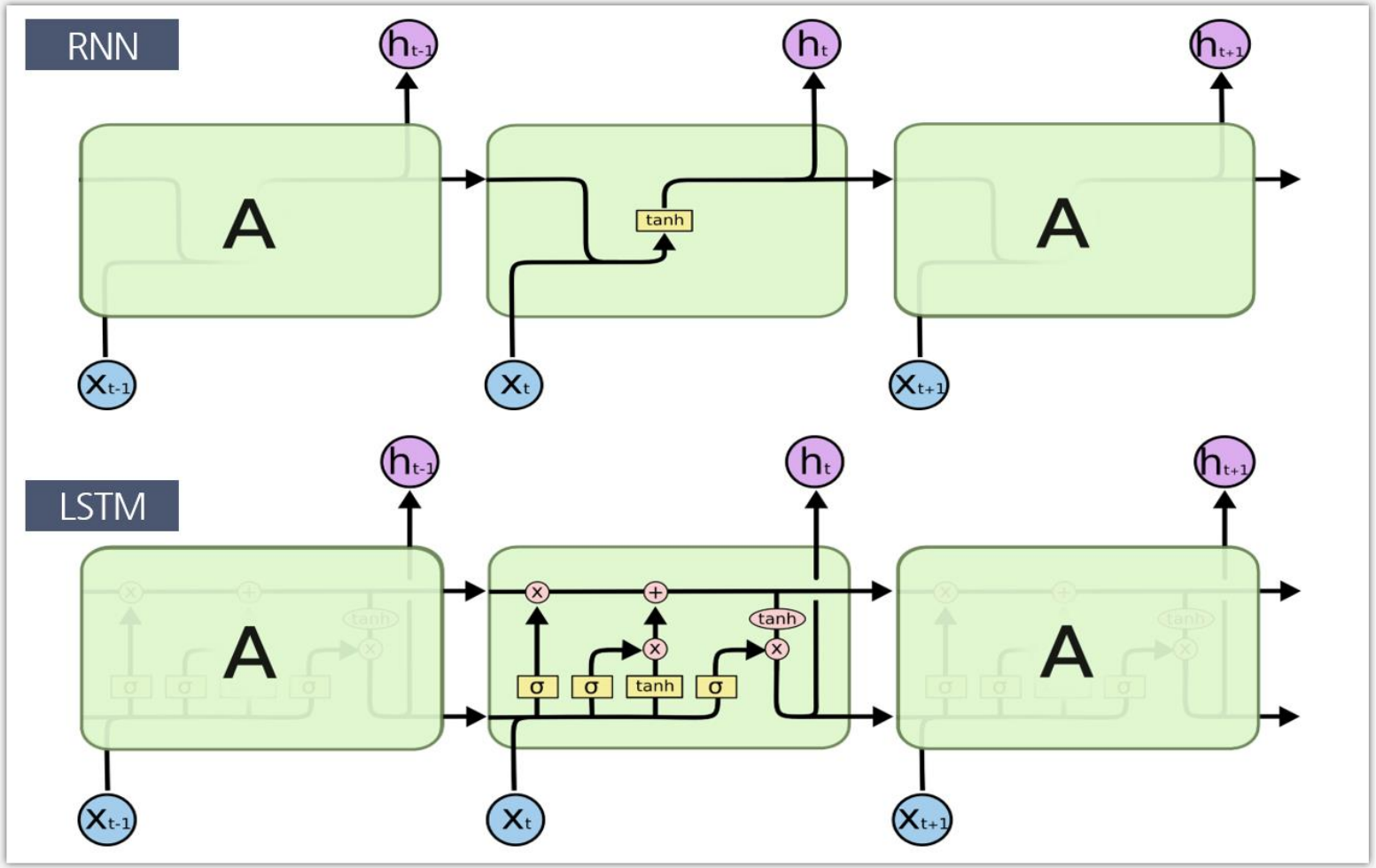
LSTM은 **인간** 뇌의 장단기 기억 구조와 유사

기억의 유지와 망각, 기억의 선택과 집중

LSTM은 Vanilla RNN에 비해 Cell 내부가 **복잡한 구조**

RNN의 개선 모델

RNN과 LSTM



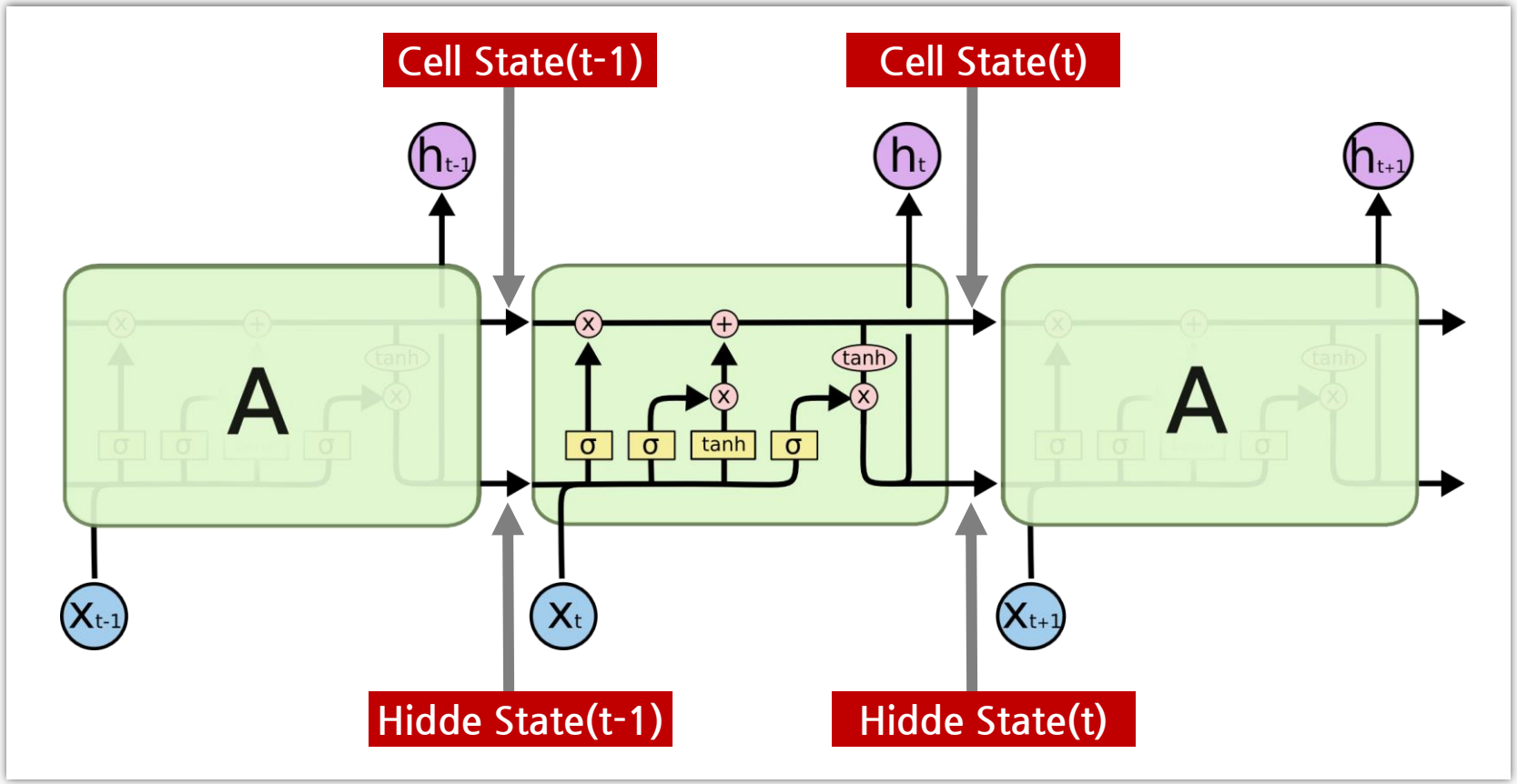
〈출처: http://colah.github.io. 20. 9〉

LSTM의 구조

입력 데이터	현재 Timestep의 입력, 이전 Cell로부터 전달된 Cell State와 Hidden State
Cell 구성	Input Gate, Forget Gate, Output Gate로 구성
출력 데이터	현재 Cell의 Hidden State 다음 Cell의 Cell State, Hidden State로 전달

RNN의 개선 모델

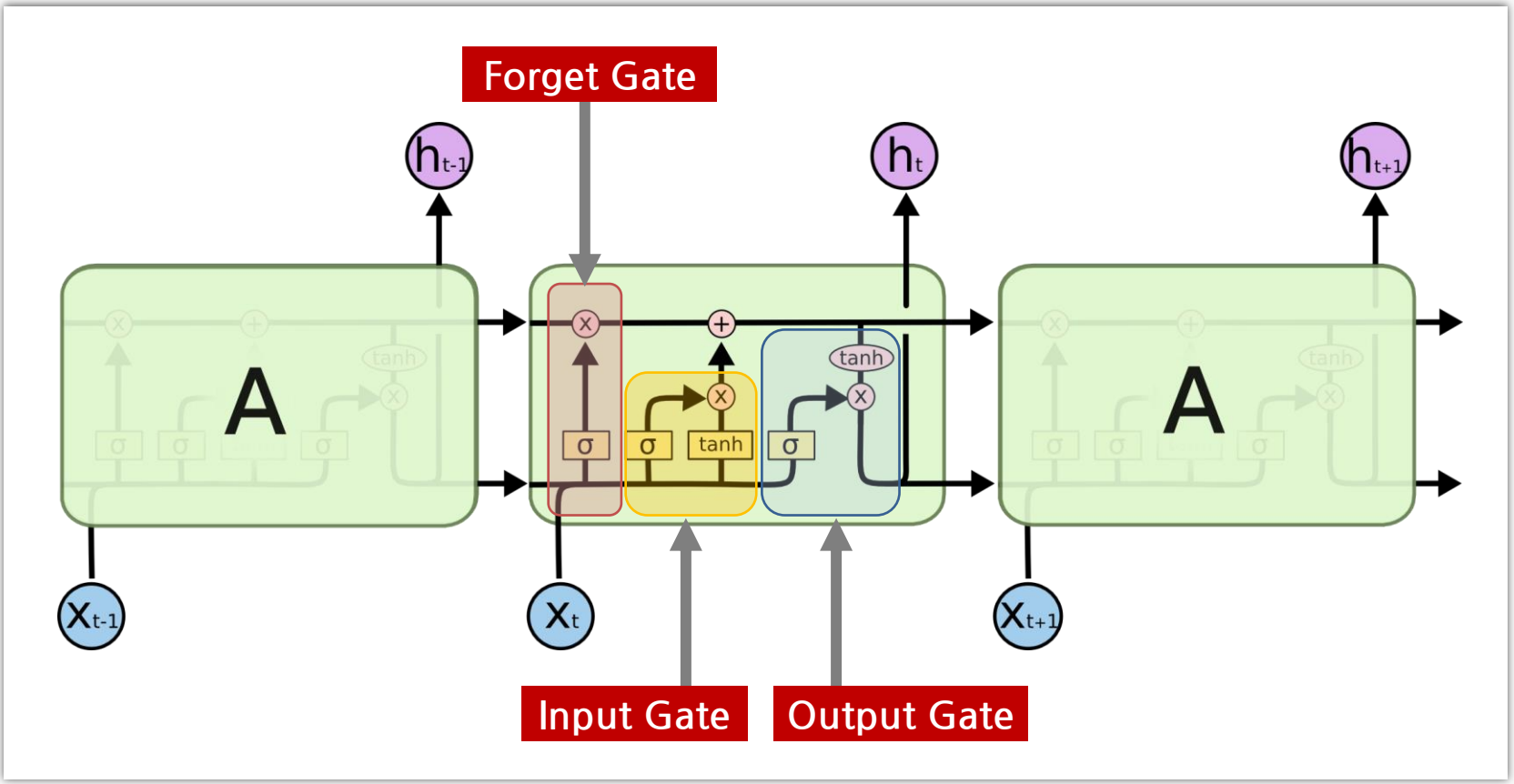
LSTM의 구조



〈출처: http://colah.github.io. 20. 9〉

RNN의 개선 모델

LSTM의 구조



〈출처: http://colah.github.io. 20. 9〉

RNN의 개선 모델

LSTM 메커니즘

이전 Cell에서 전달된 Cell State는
큰 변화없이 다음 Cell의 Cell State로 전달

» 장기 의존성 문제 해결을 위한 메커니즘

이전 Cell에서 전달된 Hidden State는
Input, Forget, Output 3개의 게이트를
통해 다음 Cell에 전달할 데이터 결정

» 선택과 집중 메커니즘

GRU 개요

GRU(Gated Recurrent Unit)란?

한국의 Cho, et al.(2014)에 제안

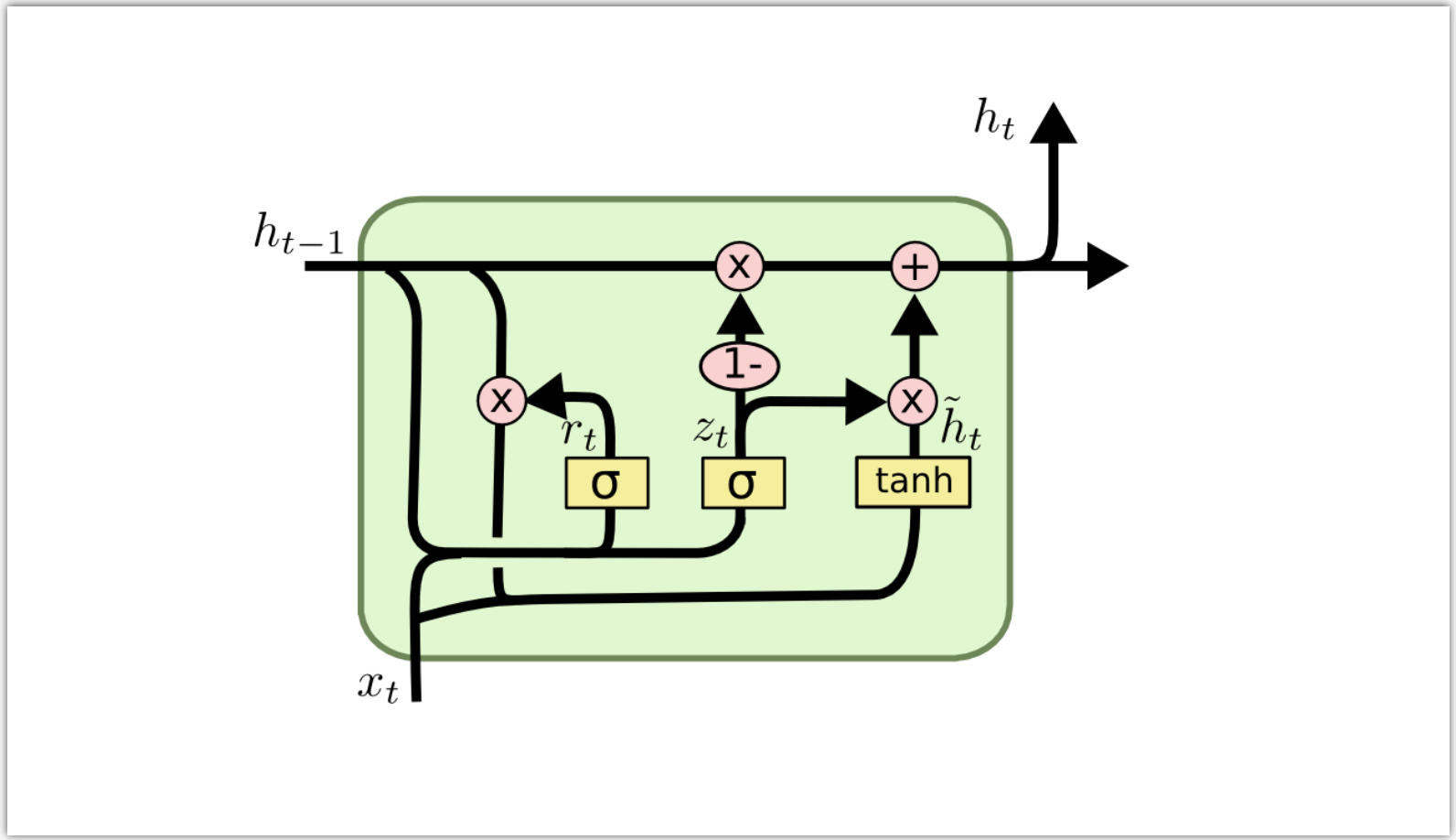
GRU는 LSTM의 복잡한 구조를 **단순화**하여
Reset Gate와 Update Gate 2개의 게이트로 구성

LSTM의 Forget Gate와
Input Gate를
Update Gate로 통합

LSTM에서 Forget Gate역할이
Reset Gate와
Update Gate로 양분

RNN의 개선 모델

GRU 구조

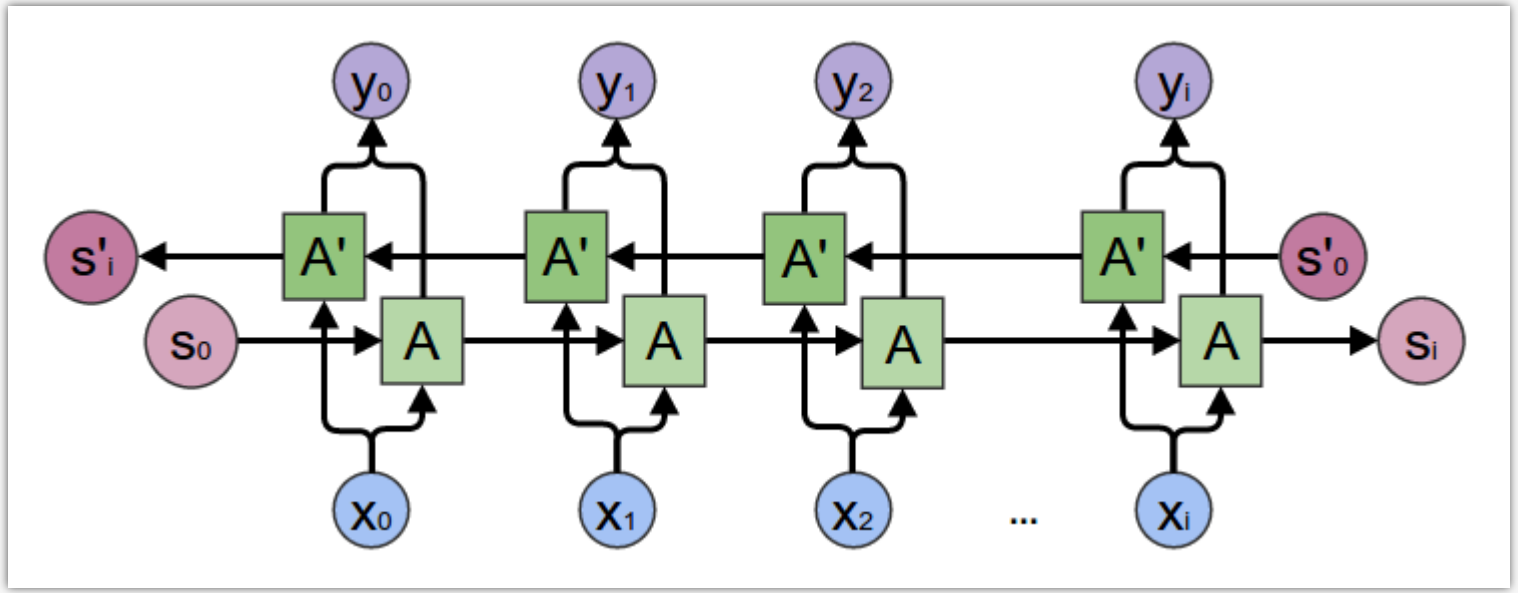


〈출처: http://colah.github.io. 20. 9〉

Bidirectional LSTM 개요

Bidirectional LSTM이란?

- Forward 방향으로의 영향(t → t+1) 뿐만 아니라 Backward(t → t-1) 방향의 양방향 영향도 함께 고려
- 과거와 현재, 미래의 모든 문맥(Context) 고려



순환 신경망 실습

순환 신경망을 이용한 IMDB 감성분석 모델

```
class IMDBDataModule(pl.LightningDataModule):
    def __init__(self, batch_size=32, max_length=128):
        super().__init__()
        self.batch_size = batch_size
        se
        se                                                    uncased')

    def pr
        se

    def setup(self, stage=None):
        self.train_dataset = self.dataset['train']
        self.test_dataset = self.dataset['test']
        val_size = int(0.2 * len(self.test_dataset))
        self.val_dataset, self.test_dataset = torch.utils.data.random_split(
            self.test_dataset, [val_size, len(self.test_dataset) - val_size]
        )
```

2주차_RNN.ipynb

(참고) 활성화 함수

시그모이드(Sigmoid)

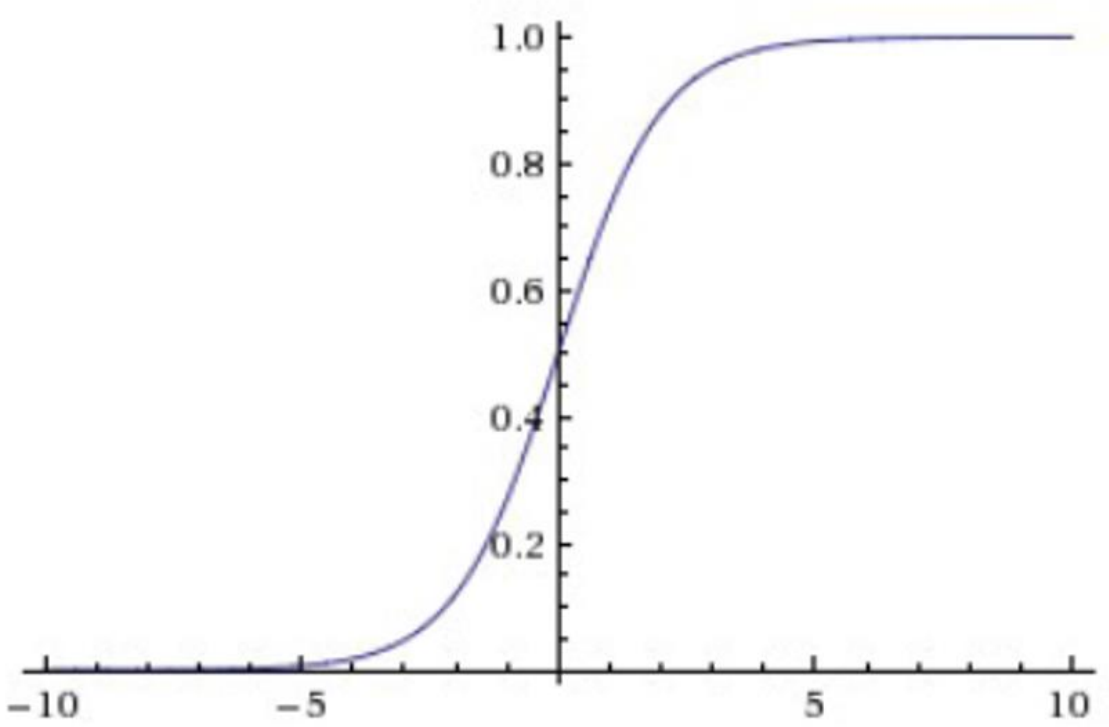
시그모이드(Sigmoid) 활성화 함수

어떠한 값이 입력되더라도
0~1 사이의 값만을 출력하는 활성화 함수

소실(Gradient Vanishing) 문제 발생 가능

(참고) 활성화 함수

- 시그모이드(Sigmoid)



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

(참고) 활성화 함수

시그모이드(Sigmoid)

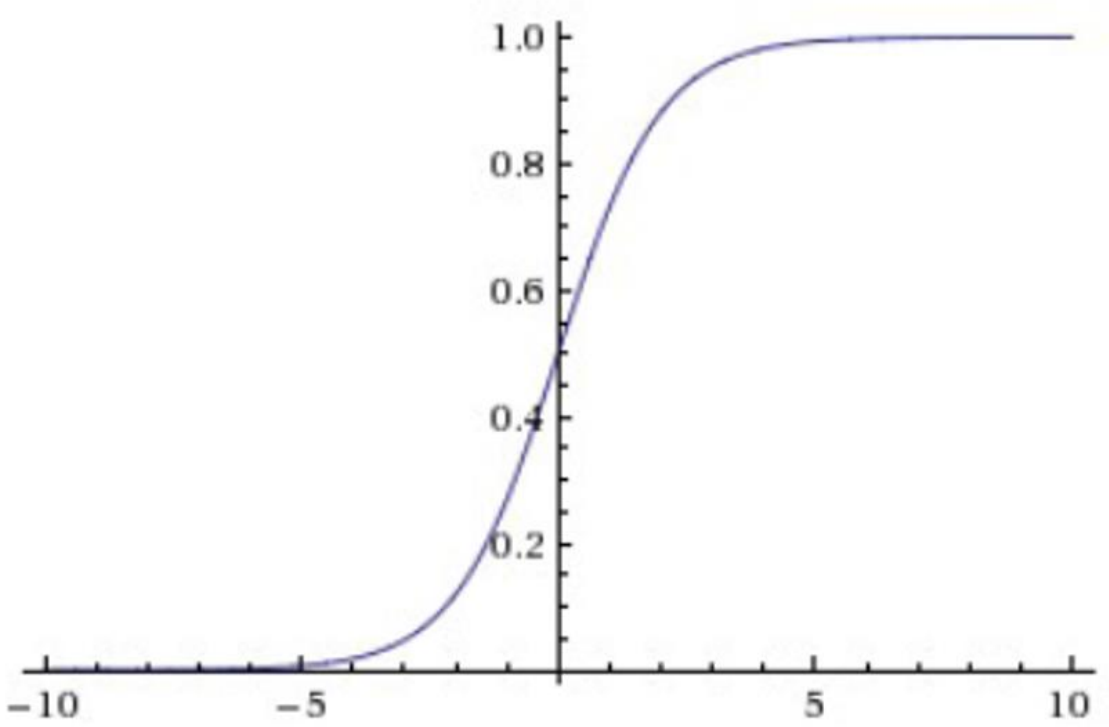
시그모이드(Sigmoid) 활성화 함수

어떠한 값이 입력되더라도
0~1 사이의 값만을 출력하는 활성화 함수

소실(Gradient Vanishing) 문제 발생 가능

(참고) 활성화 함수

- 시그모이드(Sigmoid)



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

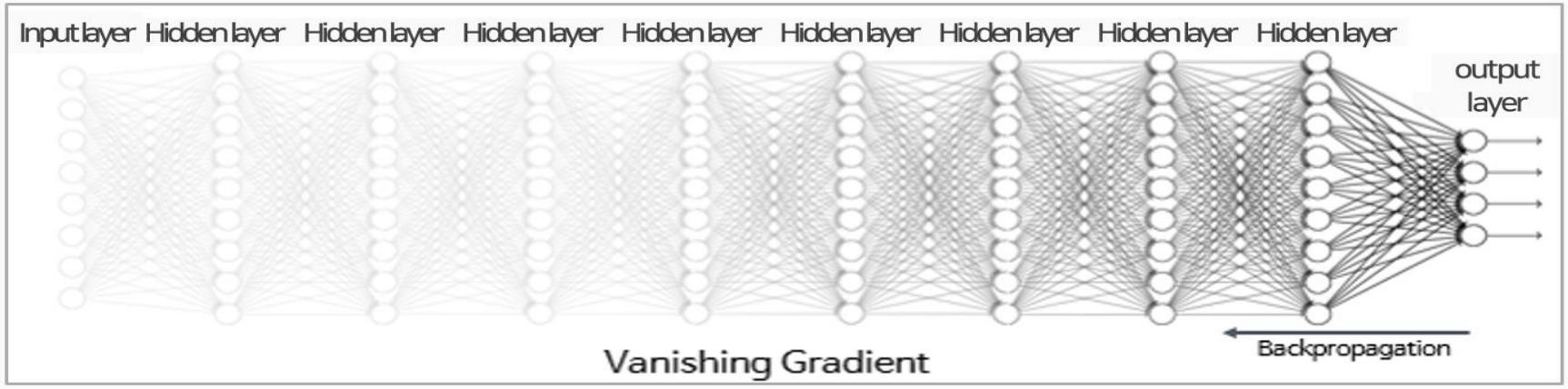
(참고) 활성화 함수

Gradient Vashing

그래디언트 소실(Gradient Vashing)

여러 단계의 Layer를 거치게 되면
더 이상 학습이 진행되지 않고 정체되는 현상

- Backpropagation을 수행하기 위해 여러 단계의 Layer를 거치게 됨
- 소수점 이하의 작은 값들의 곱셈 연산과정에서 Gradient가
  점점 작아져 0에 가깝게 수렴하게 되어 더 이상 학습이 진행되지 않음

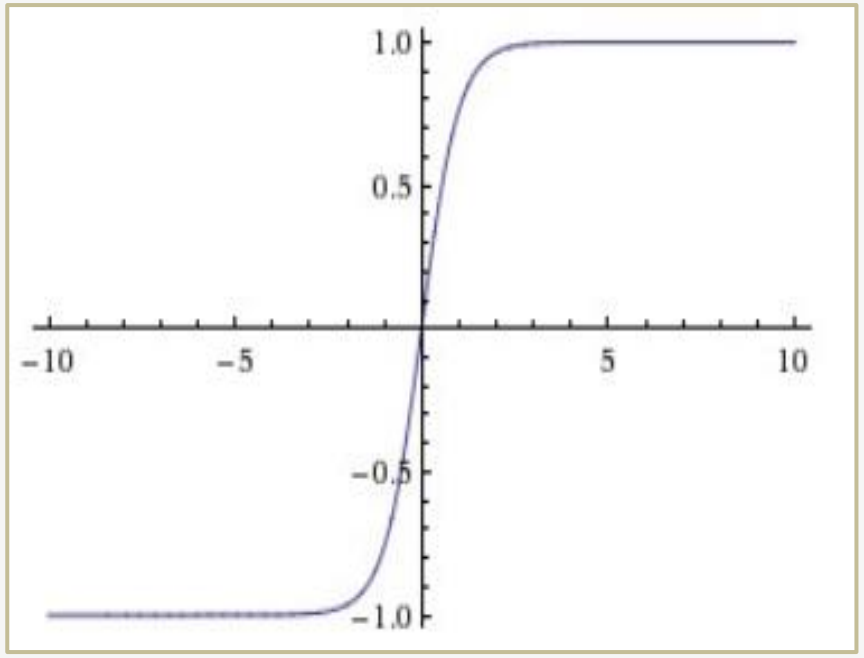


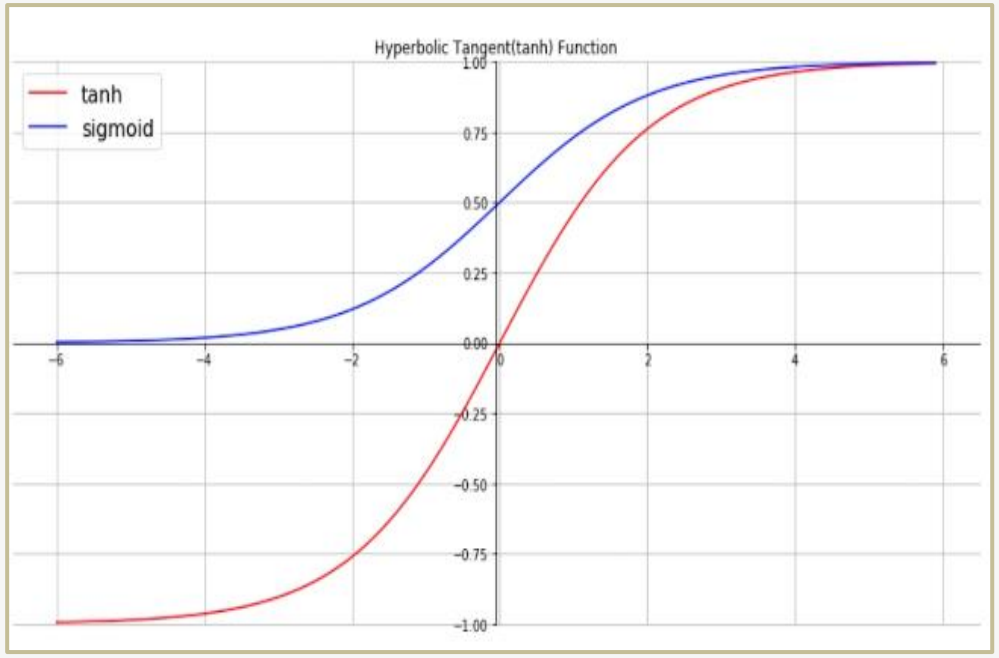
(참고) 활성화 함수

하이퍼 볼릭 탄젠트(tanh)

하이퍼볼릭 탄젠트(tanh) 활성화 함수

어떠한 값이 입력되더라도
-1~1 사이의 값만을 출력하는 활성화 함수





(참고) 활성화 함수

ReLU(Rectified Linear Unit)

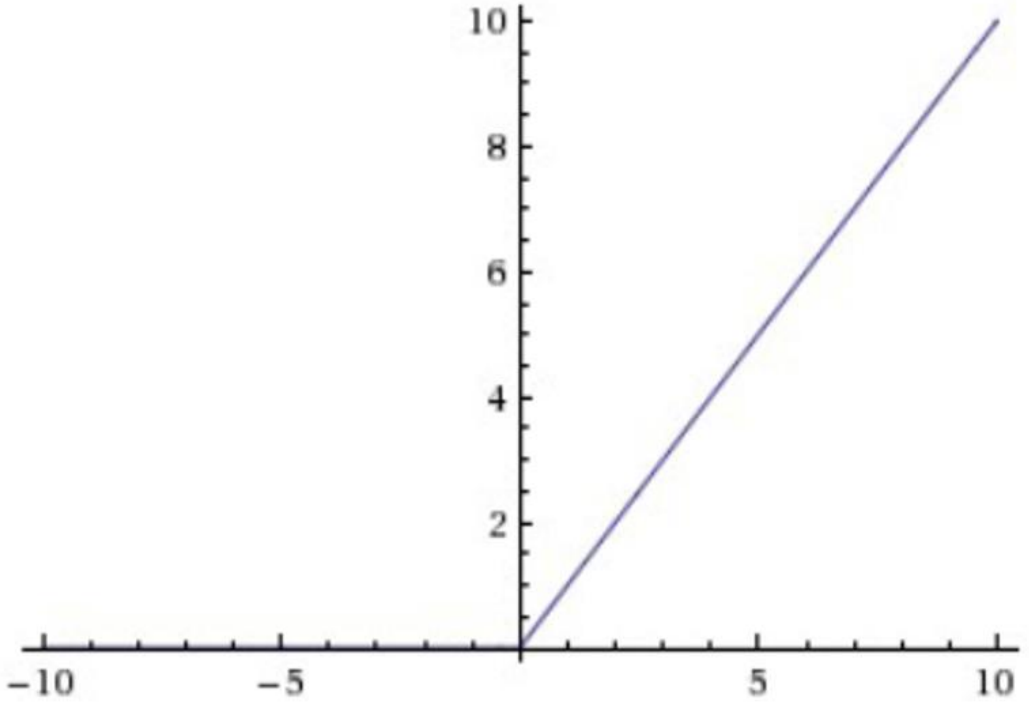
ReLU(Rectified Linear Unit) 활성화 함수

양수가 입력되면 양수 그대로를 출력하고,
음수가 입력되면 0으로 출력하는 활성화 함수

- 음수가 입력되면 0을 출력함
  → 한번 음수가 입력되면 해당 노드는 더 이상 학습이 진행되지 않음

(참고) 활성화 함수

- ReLU(Rectified Linear Unit)



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

(참고) 활성화 함수

- Leaky ReLU(Rectified Linear Unit)

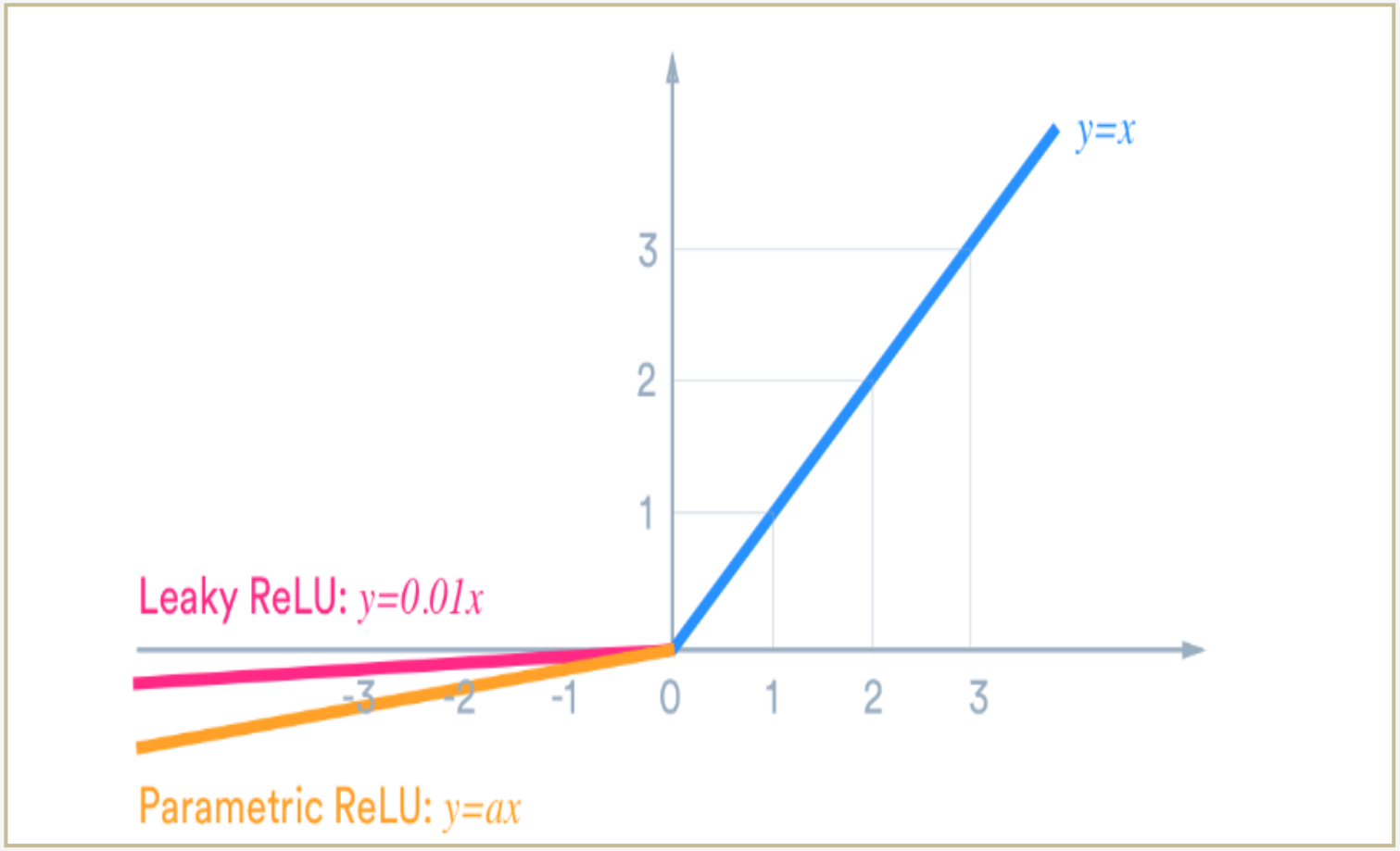
Leaky ReLU(Rectified Linear Unit) 활성화 함수

양수가 입력되면 양수 그대로를 출력하고,
음수가 입력되면 1보다 작은 값을 곱한 음수 값으로 출력하는 활성화 함수

음수 값이 입력되면 0을 출력하여 학습이 진행되지 않는
ReLU의 문제점을 개선하기 위한 활성화 함수

(참고) 활성화 함수

- Leaky ReLU(Rectified Linear Unit)



(참고) 활성화 함수

- ELU(Exponential ReLU)

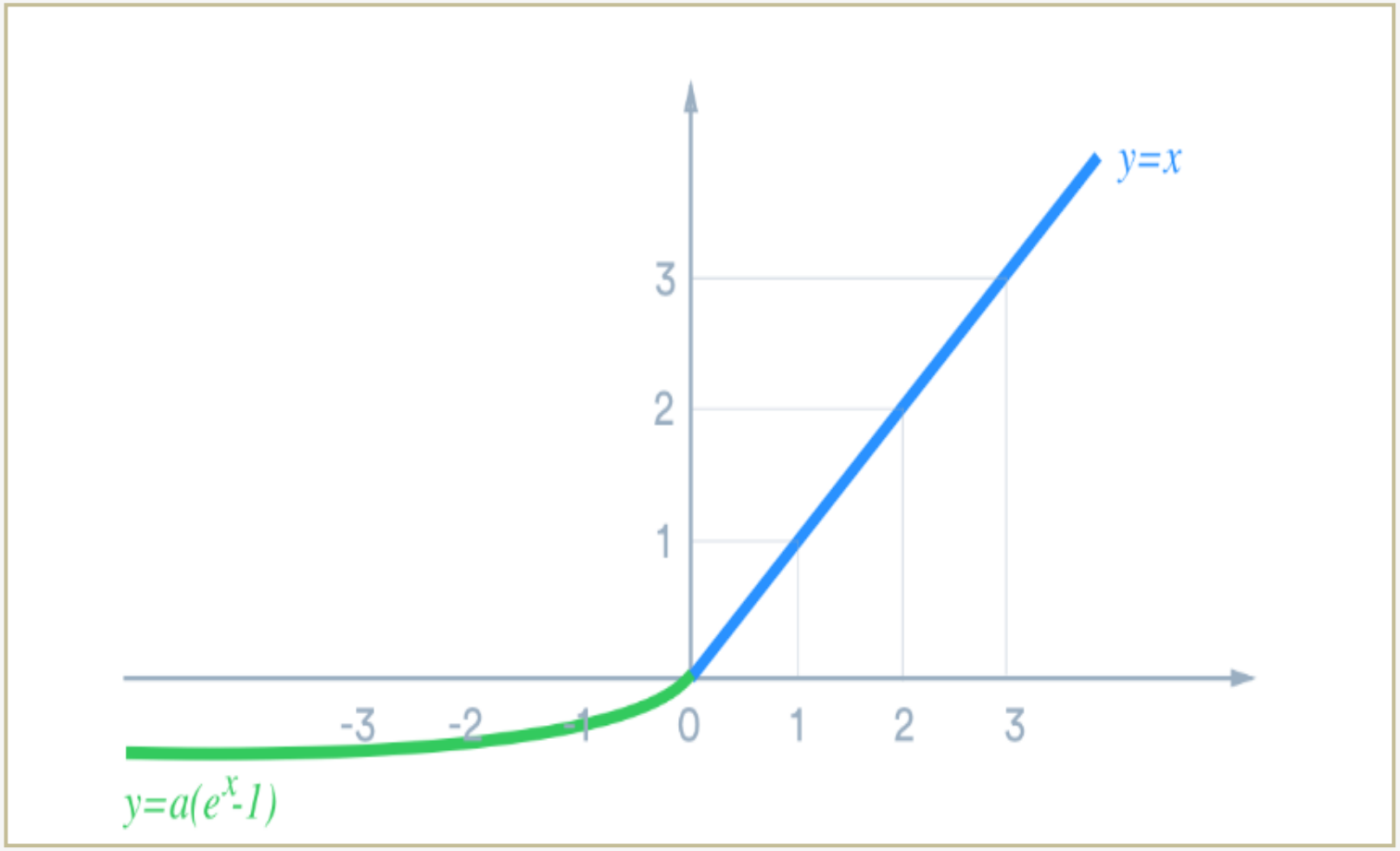
ELU(Exponential ReLU) 활성화 함수

양수가 입력되면 양수 그대로를 출력하고,
음수가 입력되면 Exponential 적용한 음수 값으로 출력하는 활성화 함수

Exponential 계산 비용 필요

(참고) 활성화 함수

- ELU(Exponential ReLU)



(참고) 활성화 함수

- GeLU(Gaussian Error Linear Unit)

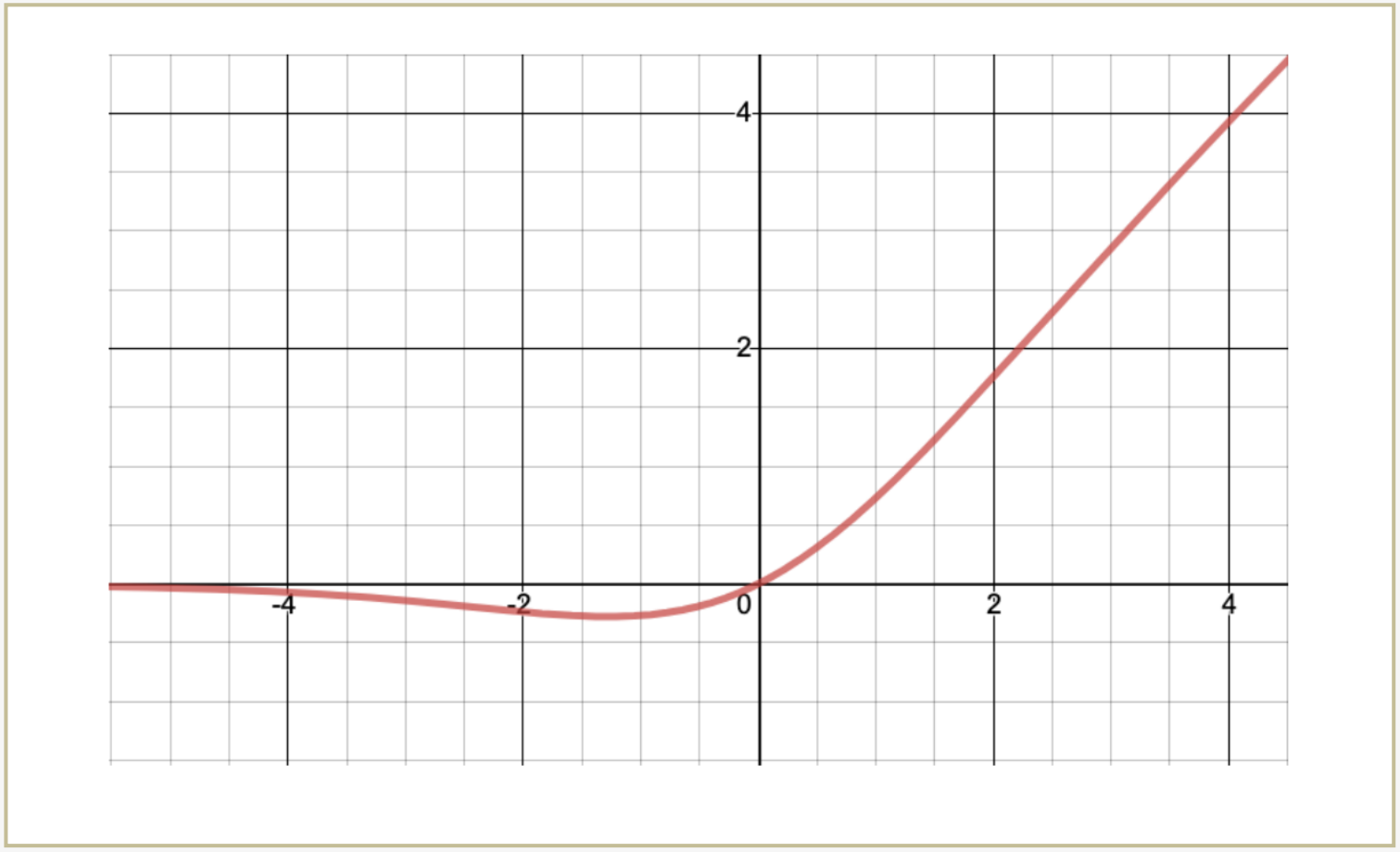
GeLU(Gaussian Error Linear Unit) 활성화 함수

양수가 입력되면 **양수 그대로**를 출력하고,
음수가 입력되면 **0에 가까운 부드러운 곡선 형태로 출력**하는 활성화 함수

ReLU보다 부드러운 출력값을 가지므로
입력 값의 정보를 더 잘 보존하고 학습이 안정적이어서
BERT 같은 최신 NLP 모델에서 많이 사용

(참고) 활성화 함수

- GeLU(Gaussian Error Linear Unit)



(참고) Pytorch Lightning

Pytorch Lightning 개요

PyTorch Lightning

PyTorch에 대한 High Level 인터페이스를 제공하는
오픈소스 Python 라이브러리

PyTorch와 PyTorch Lightning의 관계

- TensorFlow와 Keras의 관계와 유사

(참고) Pytorch Lightning

Pytorch Lightning 개요

PyTorch Lightning

PyTorch Deep Framework의 Training, Testing을 위해
복잡한 코드를 간결한 코드만으로도 사용 가능하도록 구현

(참고) Pytorch Lightning

Pytorch Lightning 개요

PyTorch Lightning

PyTorch

```
num_epochs = 1
for epoch in range(num_epochs):

  # TRAINING LOOP
  for train_batch in mnist_train:
    x, y = train_batch

    logits = pytorch_model(x)
    loss = cross_entropy_loss(logits, y)
    print('train loss: ', loss.item())

    loss.backward() … …
```

PyTorch Lightning

```
model = LightningMNISTClassifier()
trainer = pl.Trainer()

trainer.fit(model)
```

(참고) Pytorch Lightning

- Pytorch Lightning 활용

LightningModule 클래스 상속

- LightningModule 클래스를 상속하여 사용자 정의 클래스 구현

```
import pytorch_lightning as pl

class RNNModel(pl.LightningModule):
```

(참고) Pytorch Lightning

- Pytorch Lightning 활용

init 함수 재정의(Override)

- __init__( ) 함수를 이용하여 생성자 구현 및 주요 멤버 변수 초기화 수행

```
def __init__(self, embedding, lstm_input_size=300, lstm_hidden_size=100, output_size=3):
    super().__init__()
    self.embedding = embedding
    self.lstm = nn.LSTM(lstm_input_size, lstm_hidden_size)
    self.lin = nn.Linear(lstm_hidden_size, output_size)
    self.loss_function = nn.CrossEntropyLoss()

    self.train_accuracy = pl.metrics.Accuracy()
    self.val_accuracy = pl.metrics.Accuracy()
```

(참고) Pytorch Lightning

- Pytorch Lightning 활용

forward 함수 재정의(Override)

- forward( ) 함수를 이용하여 뉴럴 네트워크 구현

```
def forward(self, X: torch.Tensor):
    x = self.embedding[X].to(self.device).permute(1, 0, 2)
    x, _ = self.lstm(x)
    x = F.elu(x.permute(1, 0, 2))
    x = self.lin(x)
    x = x.sum(dim=1)
    return x
```

(참고) Pytorch Lightning

- Pytorch Lightning 활용

training, validation, test별 함수 재정의(Override)

- 학습 단계별 step 함수를 이용하여 학습 및 평가 수행

```
def training_step(self, batch, batch_idx):
    x, y = batch.text[0].T, batch.label
    y_hat = self(x)
    loss = self.loss_function(y_hat, y)
    train_acc = self.val_accuracy(y_hat, y)
    self.log('train_acc', train_acc, prog_bar=True)
    return dict(loss=loss)
    )
```

- Pytorch Lightning 활용

training, validation, test별 함수 재정의(Override)

- 학습 단계별 데이터 로더 함수를 이용하여 데이터 로딩

```
def train_dataloader(self):
    return train_iter

def val_dataloader(self):
    return test_iter
```

- Pytorch Lightning 활용

기타 옵션 함수 재정의(Override)

- optimizer 및 learning rate 설정 등 기타 옵션 함수를 이용하여 처리

```
def configure_optimizers(self):
    return Adam(self.parameters(), lr=0.01)
```

- Pytorch Lightning 활용

trainer 클래스를 이용한 학습 수행

trainer 객체 생성 및 fit( ) 함수를 이용한 학습 수행

```
trainer = pl.Trainer(
    gpus=1,
    max_epochs=3
)
trainer.fit(model)
```

- Pytorch Lightning 활용

trainer 클래스를 이용한 학습 수행

trainer 객체 생성 및 fit( ) 함수를 이용한 학습 수행

```
trainer = pl.Trainer(
    gpus=1,
    max_epochs=3
)
trainer.fit(model)
```