4. 순환신경망 모형

Hosik Choi

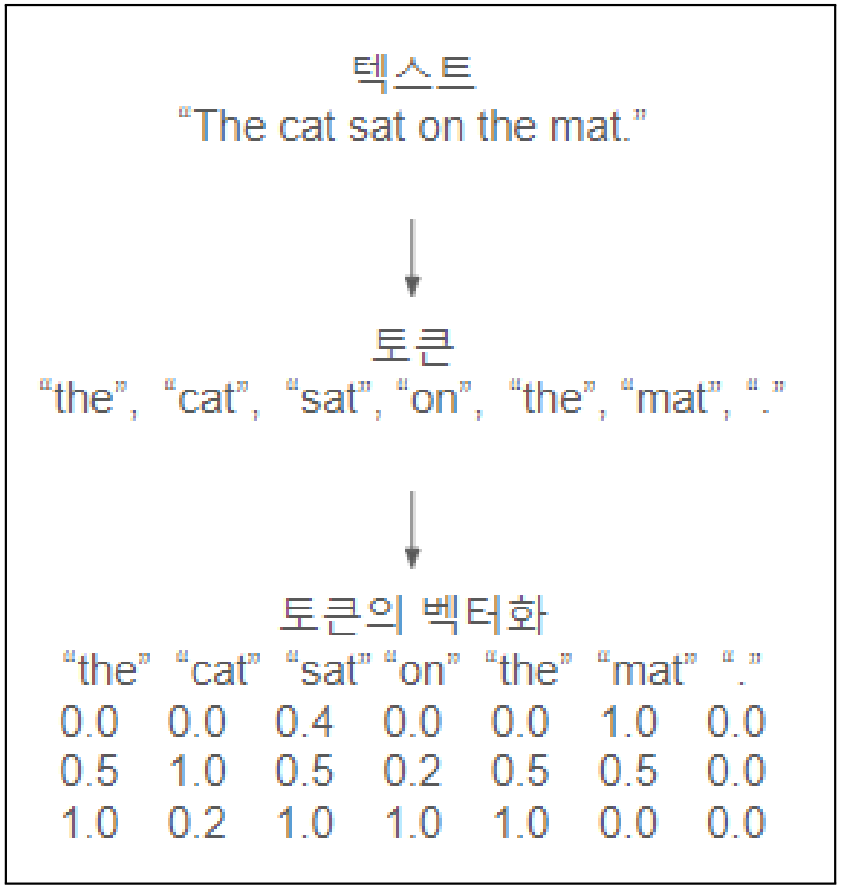
Dept. of Applied Statistics, Kyonggi Univ.

June, 2019

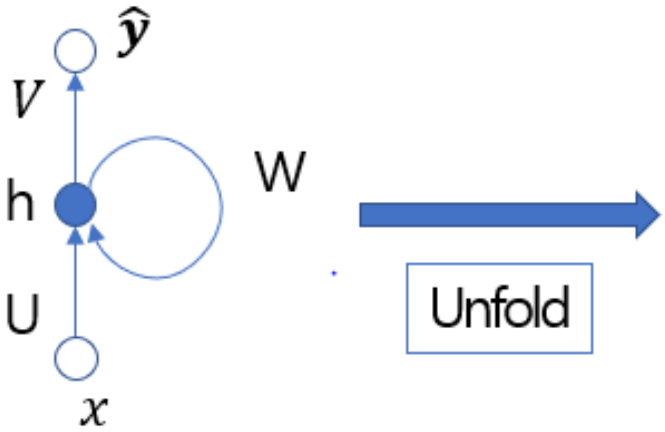
Outline

Recurrent Neural Network

- 신경망에 '시간'의 개념 도입
- 순차적으로 들어오는 정보를 처리하는 neural network
- RNN은 히든 노드가 방향을 가진 edge로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류
- 동일한 task를 입력값 sequence $\mathbf{x} = (x_1, \cdots, x_t, \cdots, x_T)$의 모든 요소마다 적용하고, 출력결과 $\hat{y}_t$에서는 이전 시간의 계산 h_{t-1}에 영향을 받음
  - 예를 들어, 문장에서 다음에 나올 단어를 추측하고 싶은 경우 이전에 나온 단어의 정보가 필요.
  - 기사주제 또는 책저자 식별과 같은 문서분류 또는 시계열분류
  - 번역: sequence to sequence learning
  - 트윗, 영화감상평의 정서를 긍정 또는 부정으로 분류


텍스트
"The cat sat on the mat."
토큰
"the", "cat", "sat", "on", "the", "mat", "."
토큰의 벡터화
"the" "cat" "sat" "on" "the" "mat" "."
0.0 0.0 0.4 0.0 0.0 1.0 0.0
0.5 1.0 0.5 0.2 0.5 0.5 0.0
1.0 0.2 1.0 1.0 1.0 0.0 0.0

RNN 모형



Why RNN?

Non-linear Auto-Regression with eXogneous inputs(NARX) model:

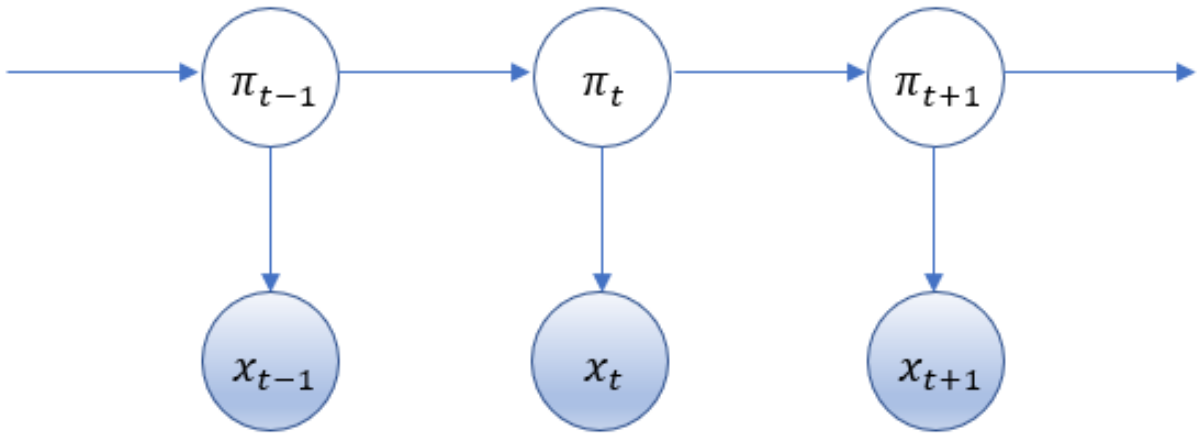
$$\hat{y}_{t+1} = F(y_t, y_{t-1}, y_{t-2}, \cdots, y_{t-d_y}, x_t, x_{t-1}, \cdots, x_{t-d_x})$$

여기서 F는 비선형 함수

- 시계열 예측에 주로 활용되는 모형으로 "Delay(d_y, d_x)"는 현재시점으로부터 과거 시점까지 입력값으로 반영할지를 결정함
- 고정크기 "Delay(d_y, d_x)"는 flexible 모형 구축에 어려움을 수반함
- 일반적으로 AR기반 모형은 dependency가 지수적으로 감소

- Hidden Markov models (HMMs)
- input: $\mathbf{x}_{1:T} \longrightarrow$ hidden: $\boldsymbol{\pi}_{1:T} \longrightarrow$ output: $\mathbf{y}_{1:L}$
- 은닉변수 π_t는 직전 시점의 π_{t-1}에만 의존.

$$P(\pi_t|\pi_{t-1}, \pi_{t-2}, \cdots, \pi_1) = P(\pi_t|\pi_{t-1})$$



- Long-range dependencies의 자료에서는 state의 개수가 너무 커지게 되어 계산량의 측면에서 비실용적임
- RNN의 경우 HMMs의 계산 문제를 피하고 long-range dependency를 포착하는 모형을 만들 수 있음

예. 장기기억

- Example of Long range dependency
  - Question: Marry went to the room, and Jane went to the room too. Jane said 'hi' to ( ).
  - Answer: 'Marry'
- The answer is the first word while the question locates at the 17th position.

RNN 모형

- $\mathbf{x}_t$: 시간 t에서의 입력값
- h_t: 시간 t에서의 중간층, 네트웍의 메모리
- $h_t = f(U\mathbf{x}_t + Wh_{t-1} + b_h)$
  - 여기서, 비선형 함수 f는 tanh, ReLU, $h_0 = 0$으로 초기화
- $\hat{y}_t = \mathrm{softmax}(Vh_t + b_y)$으로 시간 t에서의 출력값

RNN의 특징

- 각 층마다 모수의 값들이 일반적 deep Neural Network와는 달리 모든 시간 스텝에 대해 모수(앞의 그림의 U, V, W)를 공유함
- 이를 통해서 학습해야 하는 모수의 수가 줄여듦
- 해당 시점 이전의 정보를 모두 다 저장하는 중간층 $\mathbf{h}_t$는 네트워크 메모리라 할 수 있음
- RNN은 시간에 따라 펼쳐진 구조에서 역전파 방법을 이용하여 훈련시킴. 이와 같은 방법을 back-propagation through time(BPTT)이라 함.

- (vanilla) RNN은 경사소멸문제(vanishing gradient problem)를 겪음
- 경사소멸문제: 실제 구현에서는 너무 먼 과거에 일어난 일들을 잘 기억하지 못함
  - 시간 t에서 이전 시간대의 정보를 기억을 유지할 수 있어야 하나, 장기적 의존성 학습에 문제점 / 불가능
  - 계속해서 전달해야 하는 정보와 필요하지 않아 삭제해야할 정보를 잘 구분하지 못함
- Vanishing gradient 문제 해결 방법
  - W 행렬의 초기값 설정 또는 정칙화(regularization)
  - 활성함수로 ReLU 사용
  - LSTM or GRU 모형 사용

실습1: embedding layer I

- IMDB 리뷰데이터를 통해 긍정/부정 이진분류 문제에 대하여

  단어를 저차원의 실수벡터로 임베딩(embedding layer, 차원축소)하여 분류에 적용하여 보자

IMDB 데이터 자료 준비

```
> library(keras)
> max_features <- 10000 # 문서당 1000 단어로 제한
> maxlen <- 20 # 한 문서당 20개 단어로 감상평 추출

# 자료
> imdb <- dataset_imdb(num_words = max_features)
> c(c(x_train, y_train), c(x_test, y_test)) %<-% imdb

# 2D 정수값을 가지는 행렬 텐서, shape: (표본, maxlen)
> x_train <-  pad_sequences(x_train, maxlen = maxlen)
> x_test <-  pad_sequences(x_test, maxlen = maxlen)
> head(x_train)
```

망 구축, 적합

각 감상평에서 20개의 단어만 봄

```
> model <- keras_model_sequential() %>%
    # 임베딩 층: 10000 -> 8
    # shape: (표본, maxlen, 8)
    layer_embedding(input_dim = 10000, output_dim = 8,
      input_length = maxlen) %>%
    # 임베딩 3D 텐서를 2D 텐서 (표본, maxlen * 8)로 만듦
    layer_flatten() %>%
    layer_dense(units=1, activation="sigmoid") #최상층 분류로 시그모이드 추가

> model %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("acc"))

> history <- model %>% fit( x_train, y_train,
    epochs = 10, batch_size = 32,  validation_split = 0.2
)
```

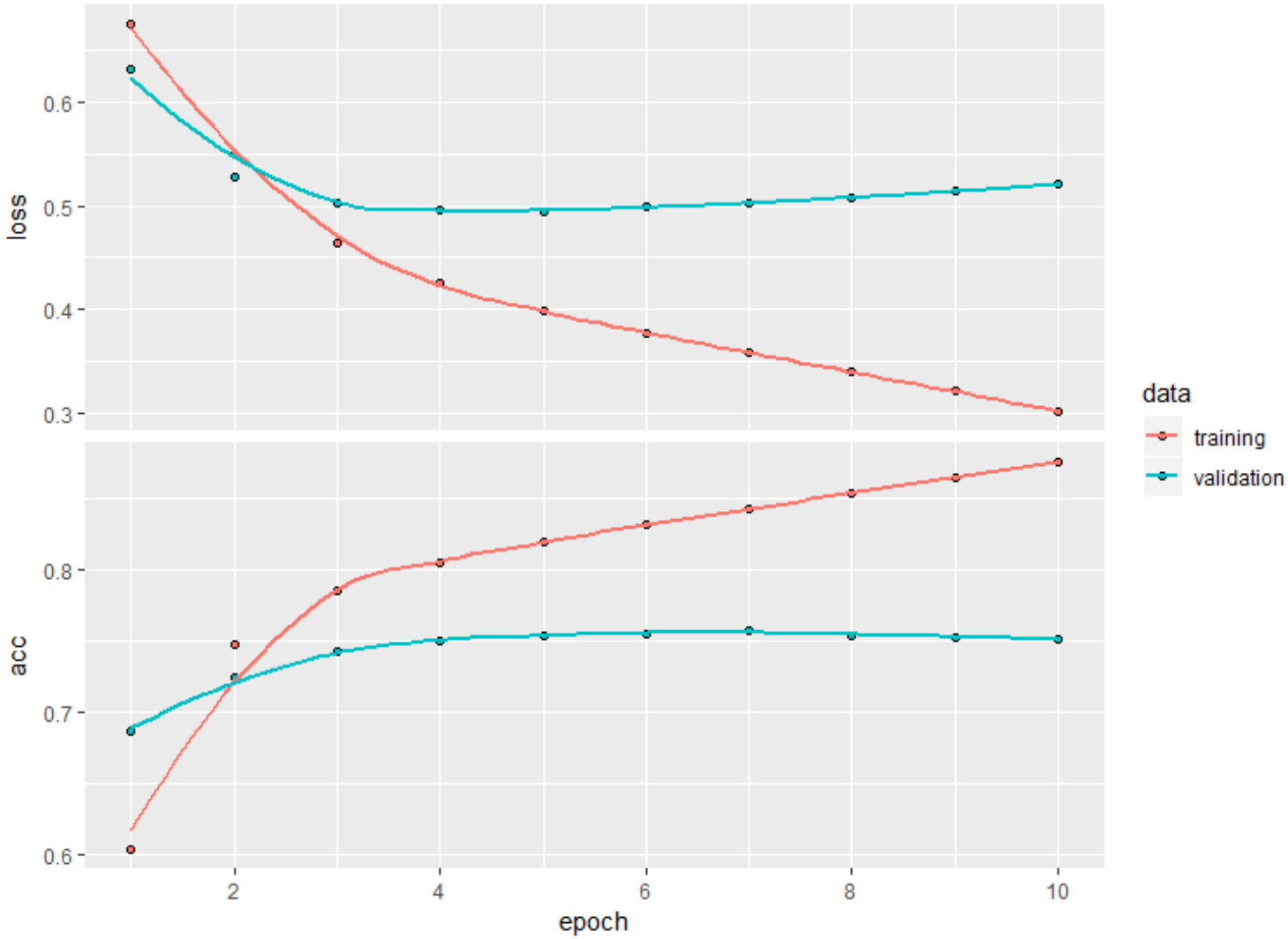


Figure: Embedding layer 활용 결과

실습2: RNN I

- IMDB 리뷰데이터를 통해 긍정/부정 이진분류 문제에 대하여 임베딩한후 RNN으로 학습하고 이를 활용하여 분류하여 본다.

자료준비

```
> library(keras)
> max_features <- 10000 ; maxlen <- 500
> batch_size <- 32

> imdb <- dataset_imdb(num_words = max_features)
> c(c(input_train, y_train), c(input_test, y_test)) %<-% imdb
> cat(length(input_train), "train sequences\n") #25000 train sequences
> cat(length(input_test), "test sequences\n")  #25000 test sequences
> cat("Pad sequences (samples x time)\n")

> input_train <- pad_sequences(input_train, maxlen = maxlen) # 25000 500
> input_test <- pad_sequences(input_test, maxlen = maxlen) # 25000 500
> cat("input_train shape:", dim(input_train), "\n")
> cat("input_test shape:", dim(input_test), "\n")
```

망 구성

```
> model <- keras_model_sequential() %>%
    layer_embedding(input_dim = max_features, output_dim = 32) %>%
    layer_simple_rnn(units = 32) %>%
    layer_dense(units = 1, activation = "sigmoid")
> summary(model)
```

```
_____________________________________________________________________________________________
Layer (type)                                 Output Shape                              Param #
=============================================================================================
embedding_1 (Embedding)                      (None, None, 32)                          320000
_____________________________________________________________________________________________
simple_rnn_1 (SimpleRNN)                     (None, 32)                                2080
_____________________________________________________________________________________________
dense (Dense)                                (None, 1)                                 33
=============================================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
```

망 적합, 평가

```
> model %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("acc"))

> history <- model %>% fit(
    input_train, y_train,
    epochs = 10,
    batch_size = 128,
    validation_split = 0.2)

> plot(history)
```

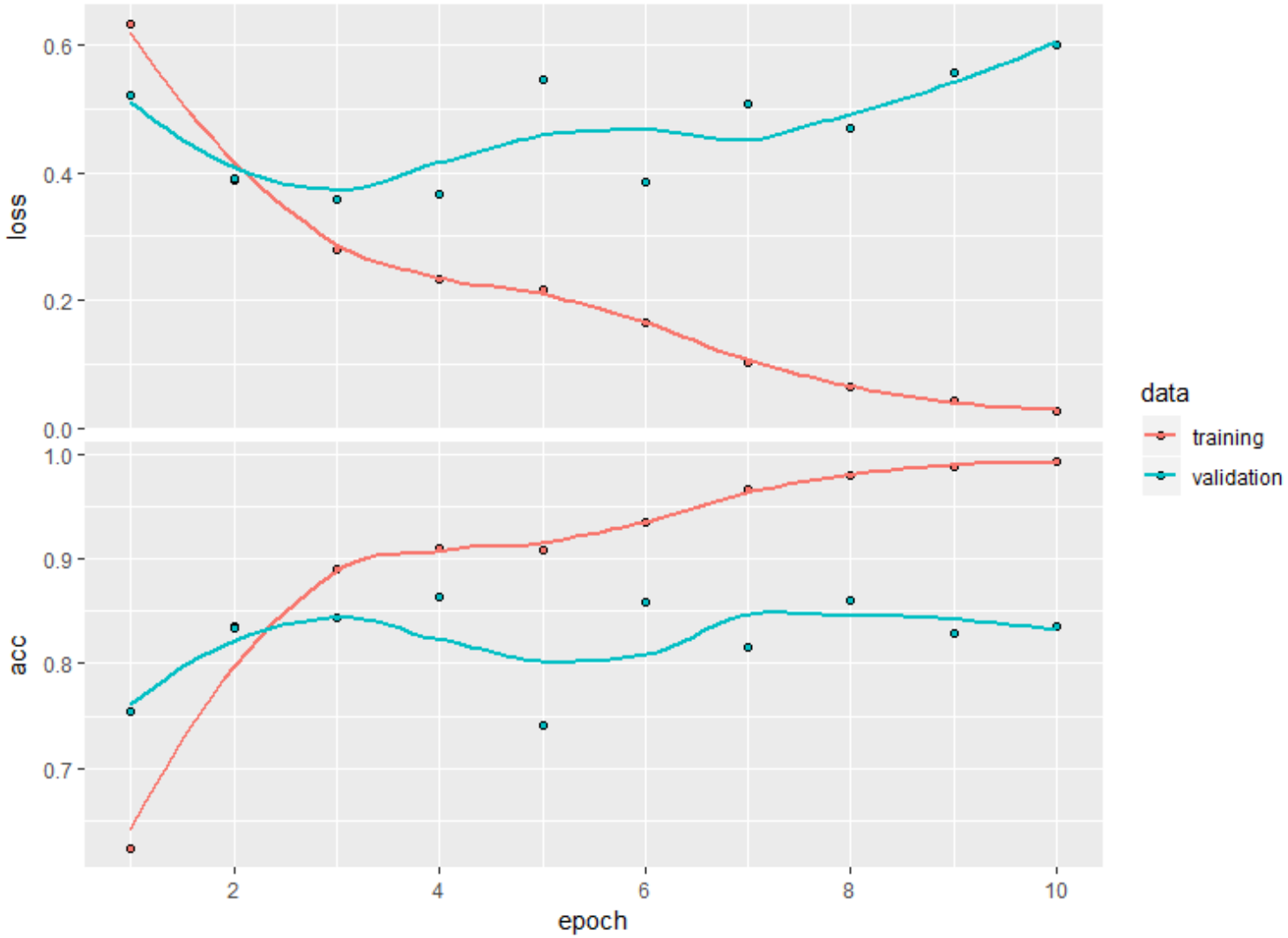
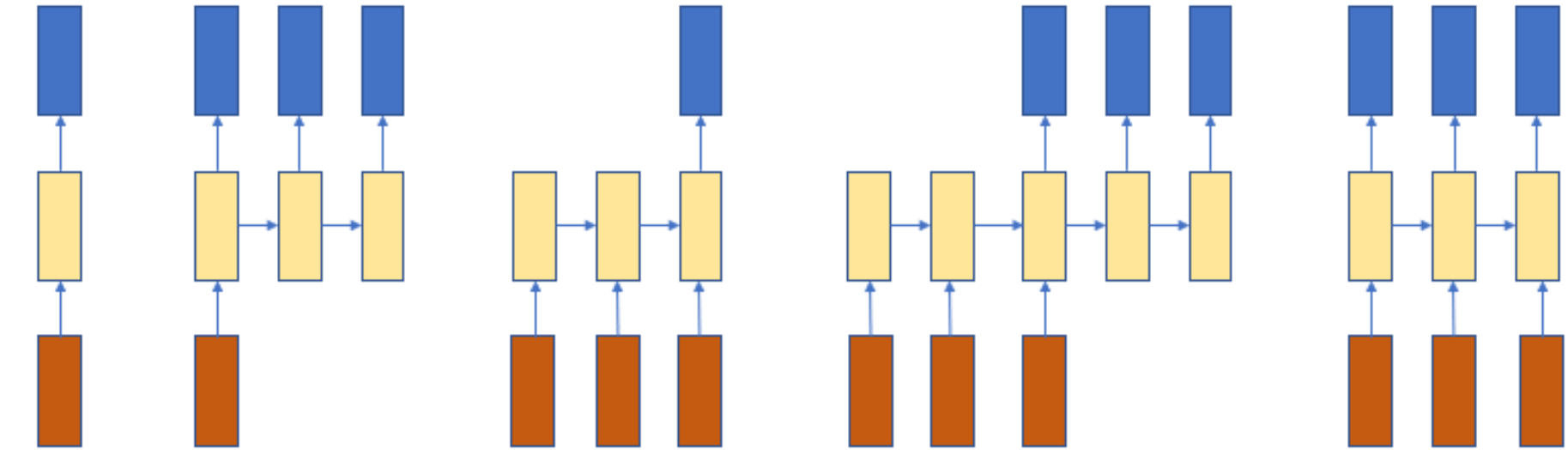


Figure: RNN 결과

RNN의 응용



One-to-one
기본 NN

One to many
예: 이미지 캡션
이미지 -> 문장

many to one
예: 감정 분류
문장 -> 호/불호

many to many
예: 기계번역
문장 -> 문장

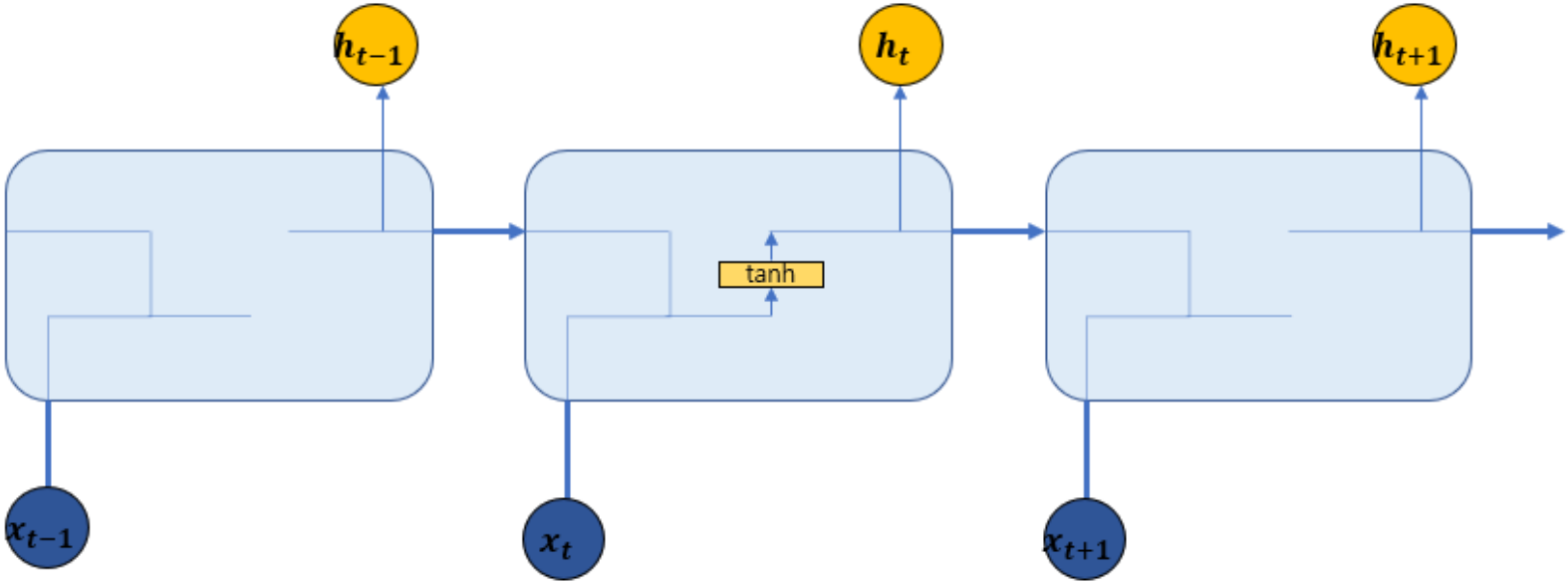
many to many
예: 비디오 캡션
비디오 -> 문장

LSTM 모형

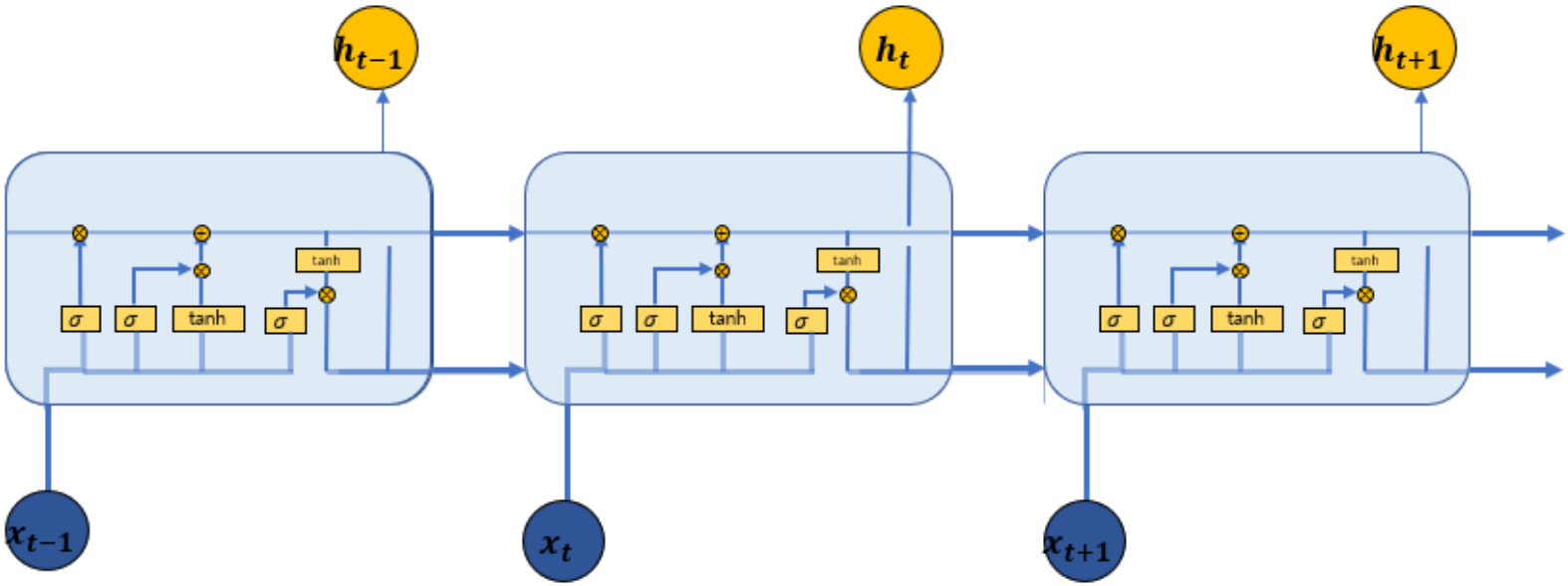
Long Short Term Memory(Hochreiter & Schmidhuber, 1997)은 셀 하나는 서로 상호작용하는 4개의 입력계층(1개의 기억 셀 + 3개의 게이트)으로 구성됨

- Vanishing gradient 문제를 해결하기 위한 대표적인 방법 중 하나
- Memory cell이라는 새로운 노드를 추가하여 과거의 정보가 현재에도 영향을 잘 미칠 수 있도록 함
- 셀 상태의 정보를 제어하기 위한 3가지 게이트 도입

vanilla RNN

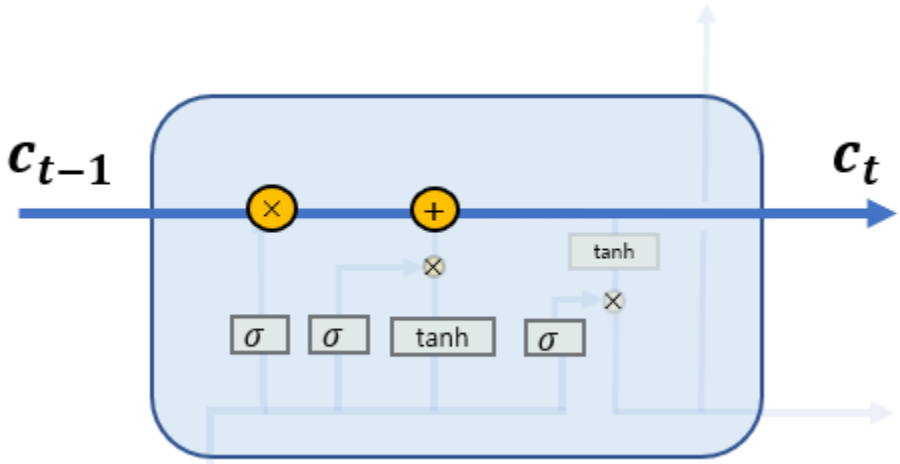


LSTM



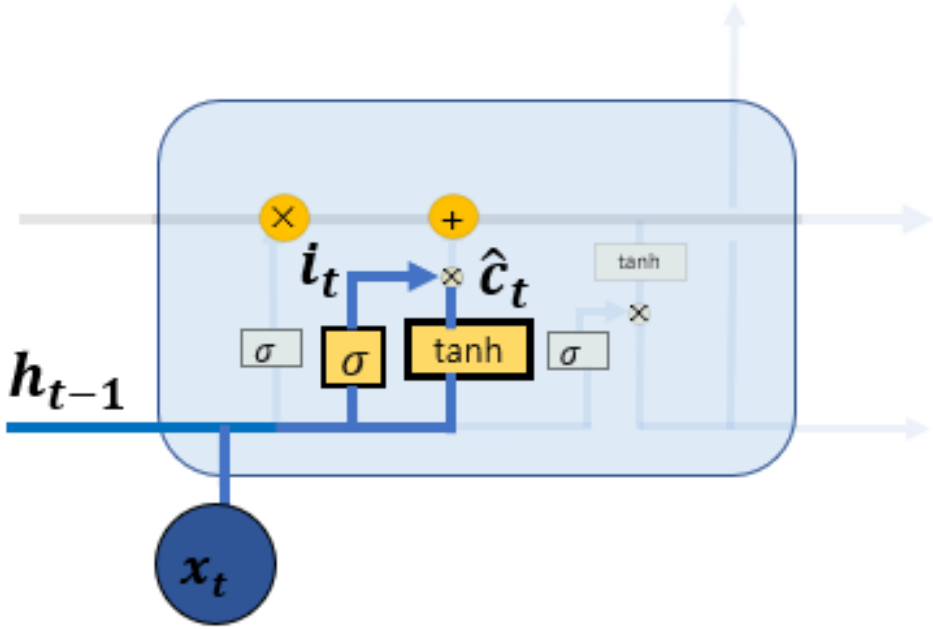
Gate & Cell

- 기억 셀(memory cell): c_t라 표기.
  과거의 정보를 저장
- $c_t = c_{t-1} + \text{function}(\mathbf{x}_t, \mathbf{h}_{t-1})$
- 게이트(gate): 특정 정보의 보존 정도
  $\in [0, 1]$
  - 0: 모든 정보가 지워짐
  - 1: 모든 정보를 보존
  - 은닉상태는 상위계층의 입력값으로
    전달됨



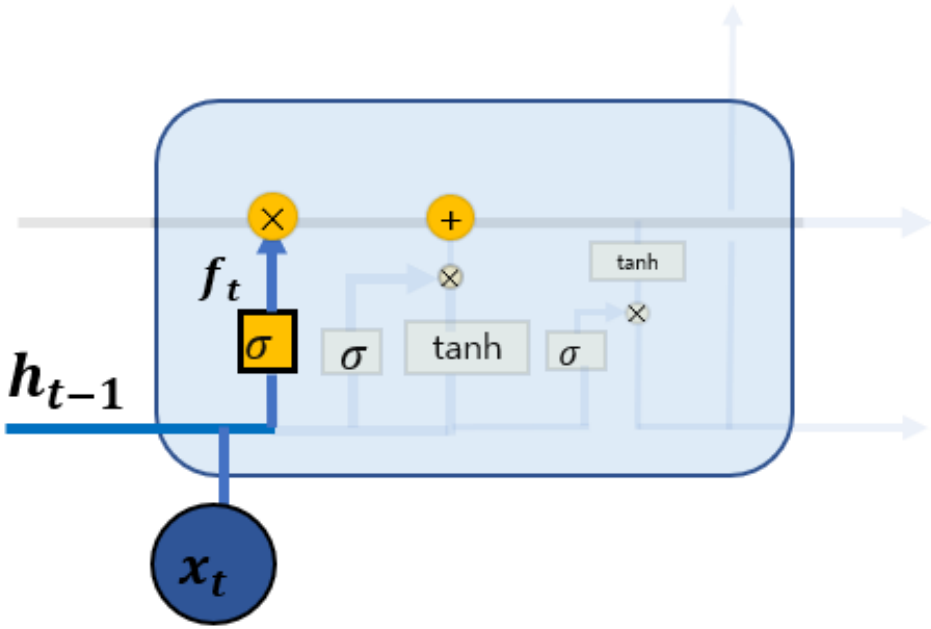
입력 게이트(input gate)

- 어떤 정보를 셀 상태에 저장할지 결정 (i_t)
- $i_t = \sigma(S_i \mathbf{x}_t + W_i h_{t-1} + b_i)$
- $\tilde{c}_t$는 셀 상태에 더해질 수 있는 새로운 후보값 벡터 생성
- $\tilde{c}_t = \tanh(W_c \mathbf{x}_t + W_c h_{t-1} + b_c)$



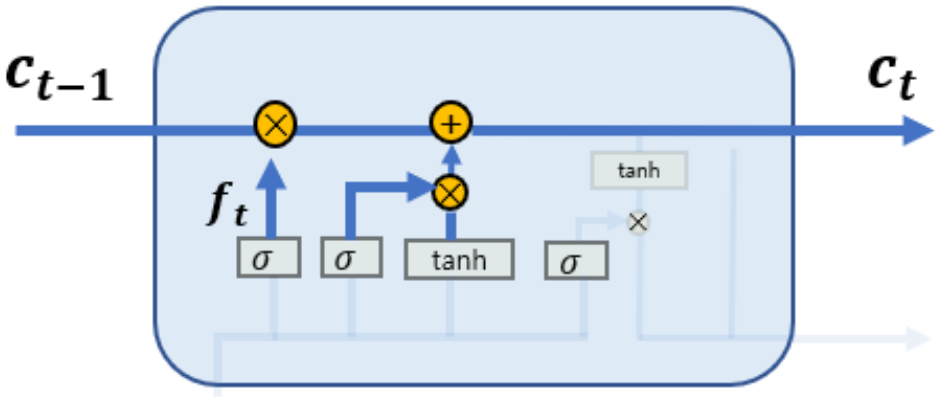
망각 게이트(forget gate) (Gers & Schmidhuber, 2000)

- $t-1$시점의 어떤 정보를 버릴지를 결정함
  - 입력데이터: $\mathbf{x}_t$, 은닉상태: h_{t-1}
- $f_t = \sigma(S_f \mathbf{x}_t + W_f h_{t-1} + b_f)$



셀 상태 업데이트

- $c_t = i_t * \hat{c}_t + f_t c_{t-1}, c_0 = 0$
- 입력 게이트에서 계산한 값과 이전 시점의 셀 상태의 값을 더하여 계산
- 이전 시점의 셀 상태에 f_t로 이전 정보를 유지비율 결정함

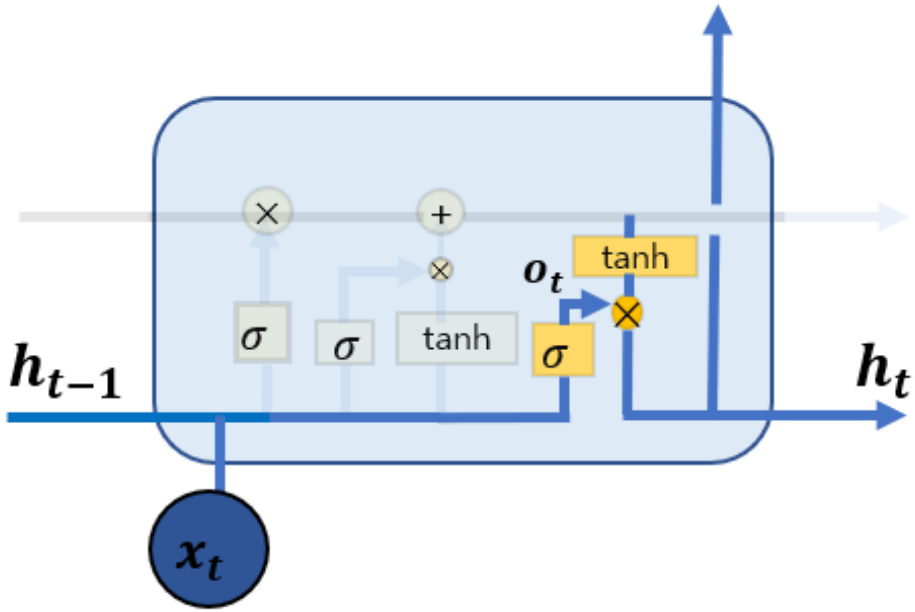


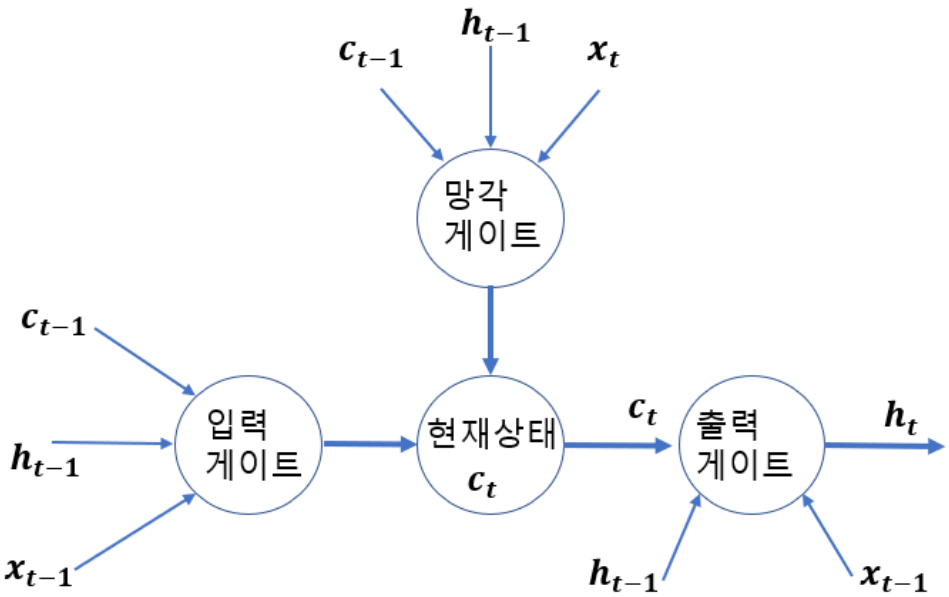
$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t$$

출력 게이트(output gate)

- o_t의 값으로 셀 상태에서 어떤 부분을 출력할지 결정
- 업데이트한 셀 상태의 값에 tanh를 변환한 뒤, o_t를 곱함

$$
\begin{aligned}
o_t &= \sigma(S_o \mathbf{x}_t + W_o h_{t-1} + b_o), \\
h_t &= o_t * tanh(c_t)
\end{aligned}
$$




c_{t-1}
h_{t-1}
x_t
망각
게이트
c_{t-1}
h_{t-1}
x_{t-1}
입력
게이트
현재상태
c_t
c_t
출력
게이트
h_t
h_{t-1}
x_{t-1}

실습: LSTM I

- IMDB 리뷰데이터를 통해 긍정/부정 이진분류 문제에 대하여 임베딩한후 RNN의 LSTM 망으로 학습하고 이를 활용하여 분류하여 본다.

LSTM 망 구성

```
> model <- keras_model_sequential() %>%
    layer_embedding(input_dim = max_features, output_dim = 32) %>%
    layer_lstm(units = 32) %>%
    layer_dense(units = 1, activation = "sigmoid")
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
lstm (LSTM)                  (None, 32)                8320
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
```

망 적합, 평가

```
> model %>% compile(
    optimizer = "rmsprop",
    loss = "binary_crossentropy",
    metrics = c("acc"))

> history <- model %>% fit(
    input_train, y_train,
    epochs = 10,
    batch_size = 128,
    validation_split = 0.2)

> plot(history)
```

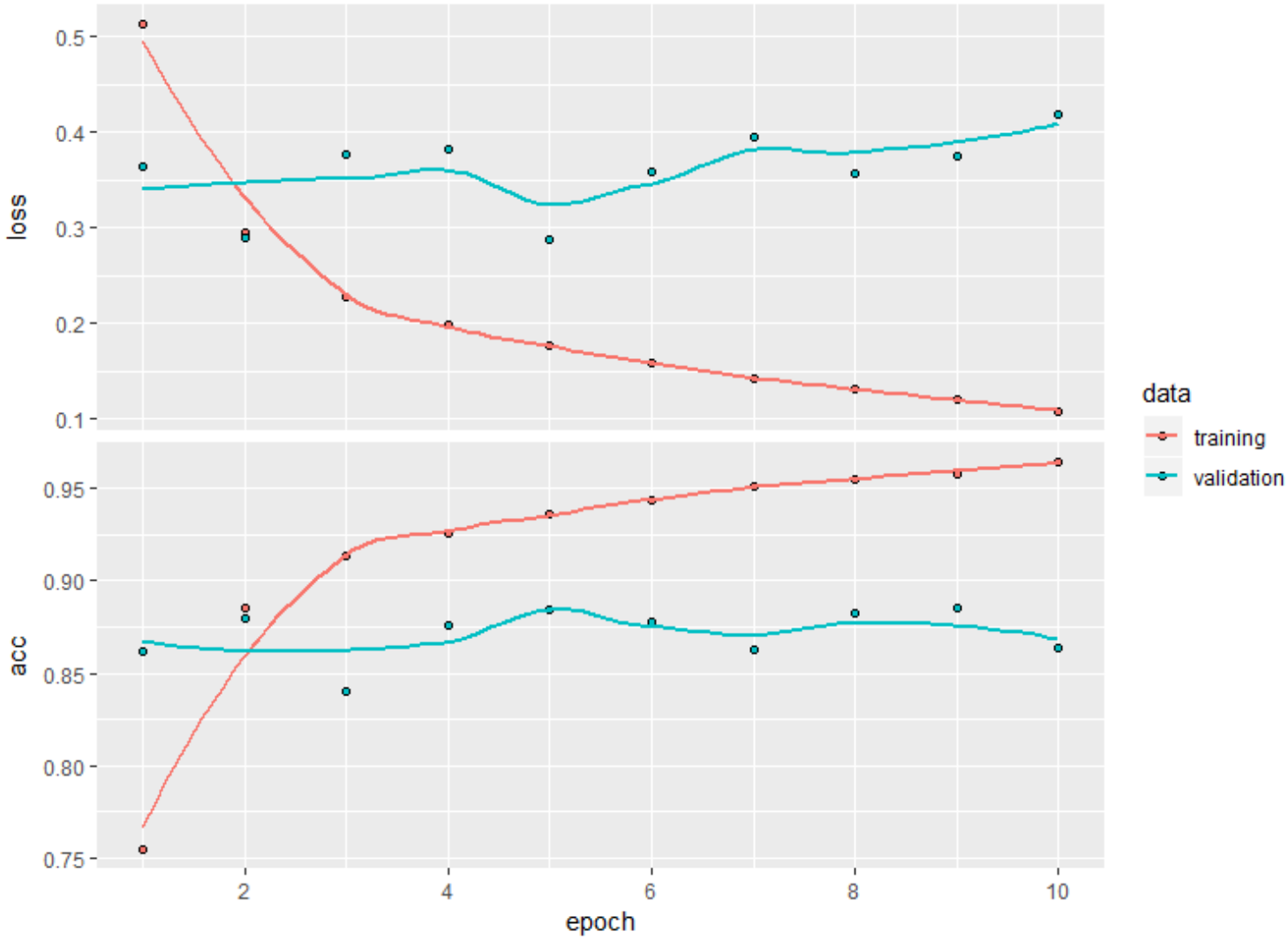


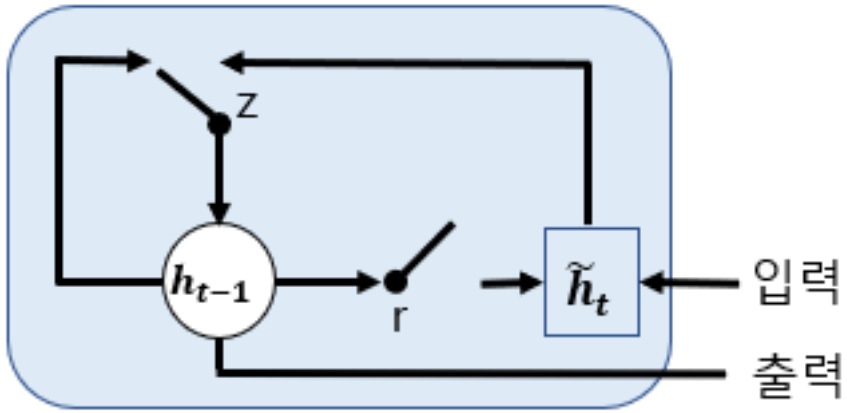
Figure: LSTM 결과

GRU 모형 I

GRU: LSTM의 변형으로 LSTM 보다는 단순한 구조를 가짐

- reset gate r과 update gate z로 총 두 가지 게이트로 구성
  - Reset gate: 새로운 입력을 이전 메모리와 어떻게 합칠지를 결정
  - Update gate: 이전 메모리를 얼마만큼 기억할지 결정

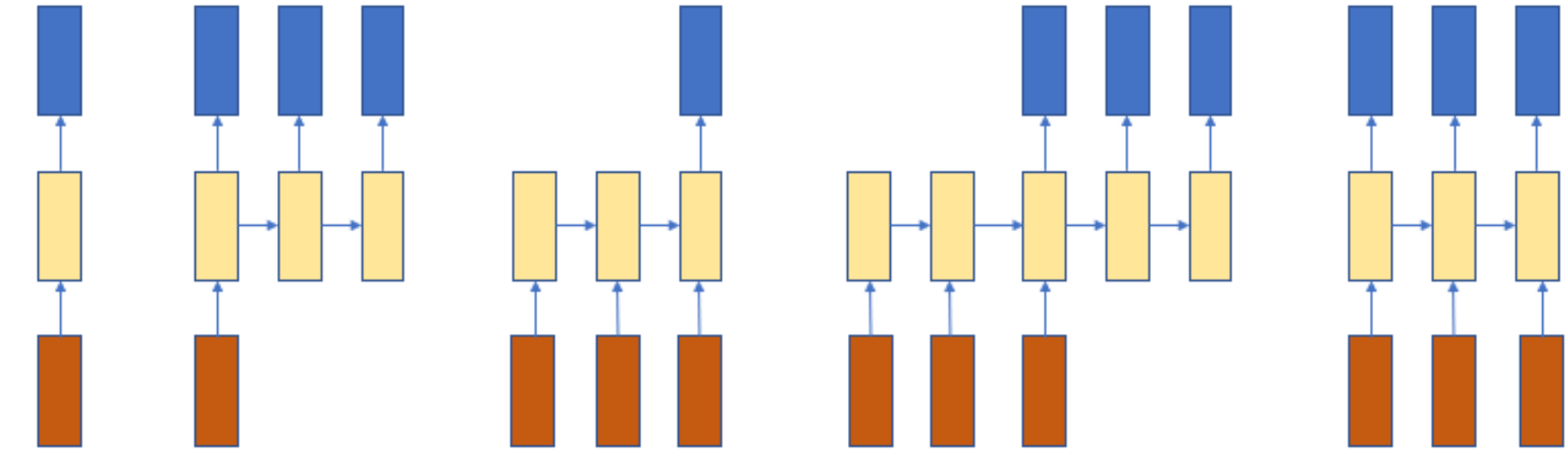
$$
\begin{aligned}
z_t &= \sigma(W_z \mathbf{x}_t + U_z h_{t-1}) \\
r_t &= \sigma(W_r \mathbf{x}_t + U_r h_{t-1}) \\
\tilde{h}_t &= \tanh(W \mathbf{x}_t + U_h (h_{t-1} \circ r) \\
h_t &= (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1}
\end{aligned}
$$



정리: LSTM vs GRU

- LSTM과 GRU 둘 다 RNN에 게이팅 메커니즘을 도입하여 긴 시퀀스를 잘 기억하도록 함.
- GRU의 경우 게이트가 2개이고, LSTM은 3개.
  - GRU은 LSTM에 있는 출력 게이트(output gate)가 없음.
- LSTM의 input gate와 forget gate가 GRU의 update gate로 합쳐짐.
- 또한 출력값을 계산 할 때에 GRU은 추가적인 비선형 함수를 적용하지 않음.

RNN의 응용



One-to-one
기본 NN

One to many
예: 이미지 캡션
이미지 -> 문장

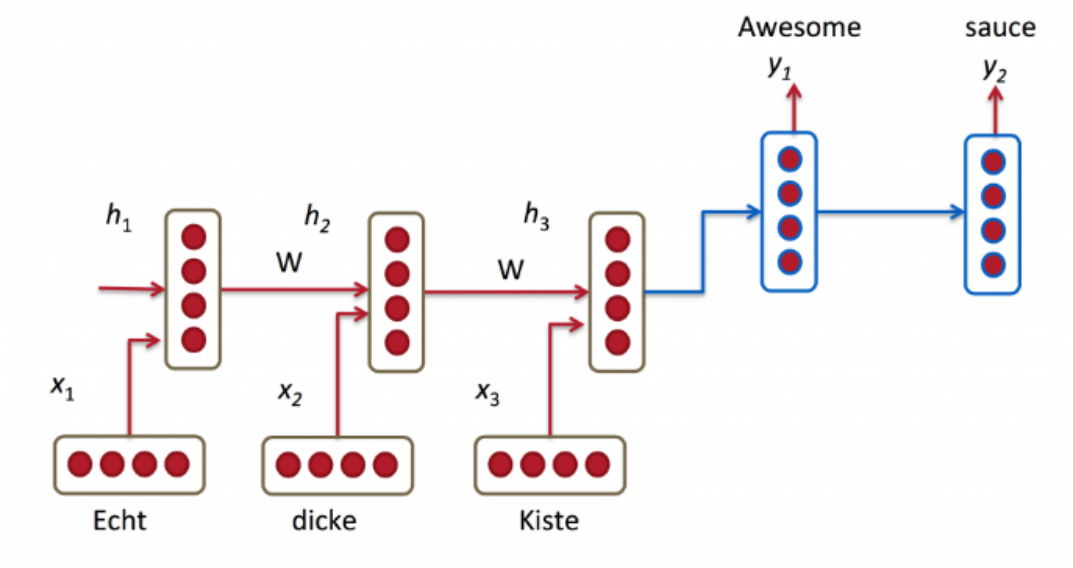
many to one
예: 감정 분류
문장 -> 호/불호

many to many
예: 기계번역
문장 -> 문장

many to many
예: 비디오 캡션
비디오 -> 문장

번역

- 입력: 단어들의 sequence, 출력: 다른 언어의 단어들 sequence
- seq2seq learning
- 어순 문제로 입력 sequence를 끝까지 입력으로 받고, 출력으로 내보내는 것이 통상적



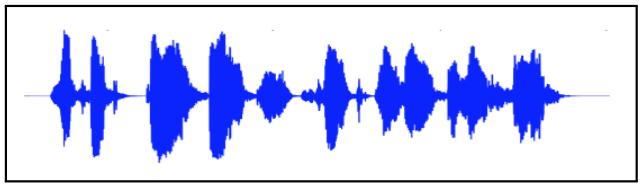
이미지 캡션

- 임의의 이미지를 텍스트로 설명해주는 시스템.
- 입력이 이미지, 출력이 단어의 sequence
- CNN(이미지 feature) + RNN(언어 모델링)
- 기계 번역의 구조에서 Encoder RNN 부분을 CNN으로 대체.

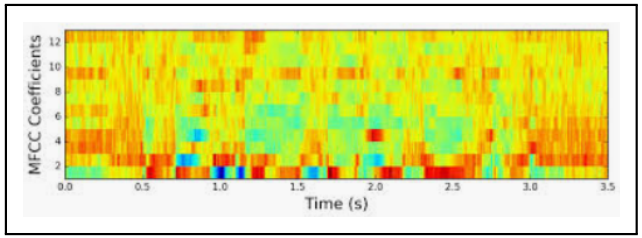
Figure: O. Vinyals et al., 2015

음성인식

- 입력: 음성, 출력: 단어들 sequence(스크립트)
- seq2seq learning
  - acoustic waveform, acoustic signal digitialization

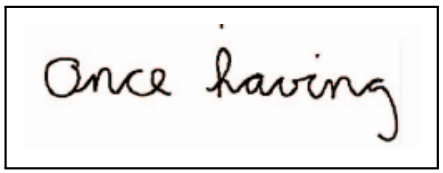


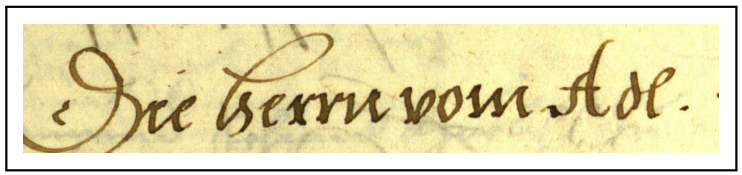
  - MFCC(Mel-Frequency Cepstral Coefficient) representation of audio



  - an input: $\mathbf{x}_i = 13 \times T_i$ matrix
  - an output: $\mathbf{y}_i = (y_1 y_2 \cdots y_{L_i})$ where $y_t \in \mathcal{L}$(label set) or $2026 \times L_i$

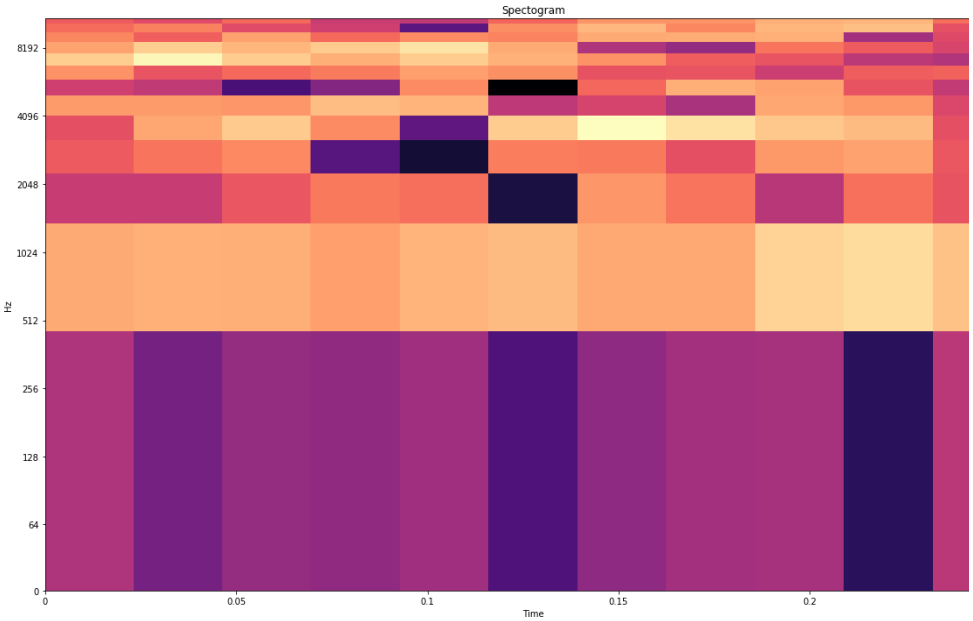
- Speech recognition의 특징
  - input: a sequence of vector(e.g. 13 dimension) with length(T)
  - output: a sequence of class labels(e.g. characters) with length(L)
  - temporal classification
  - loss: edit distance
- Handwritten text recognition

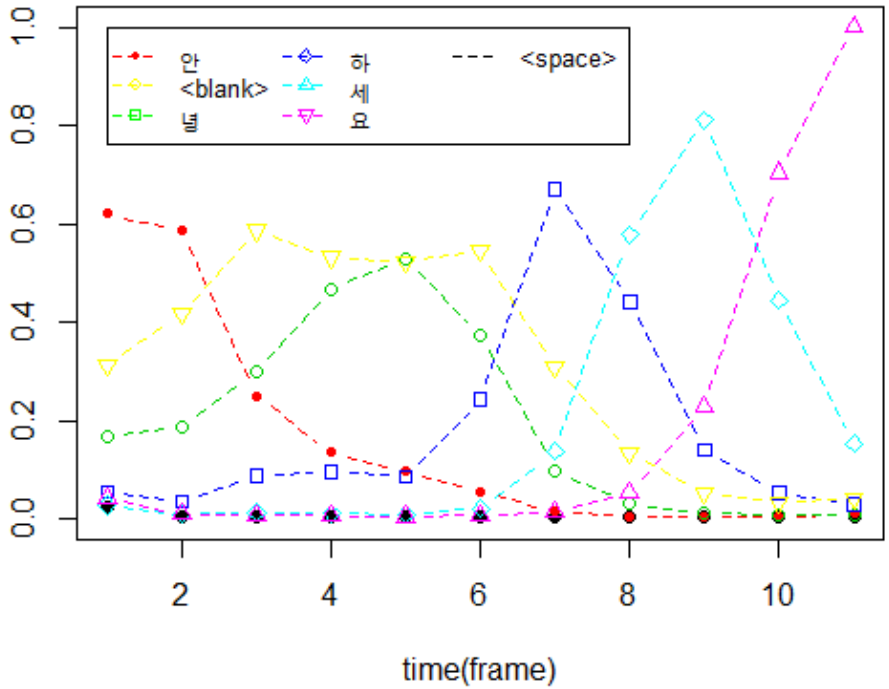


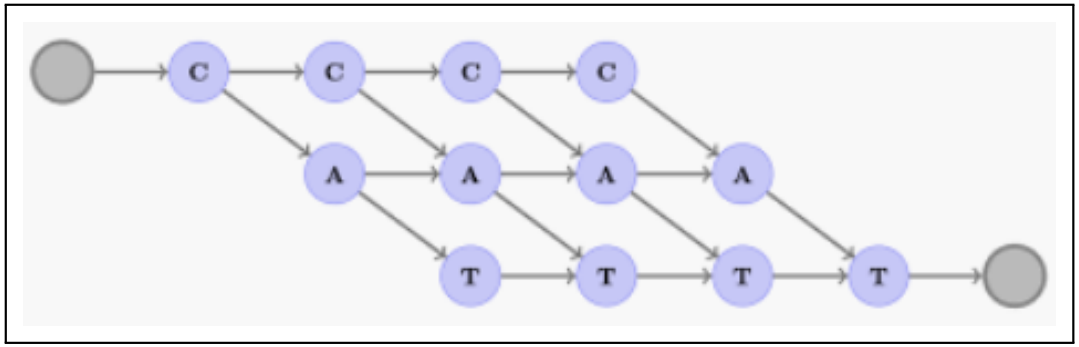


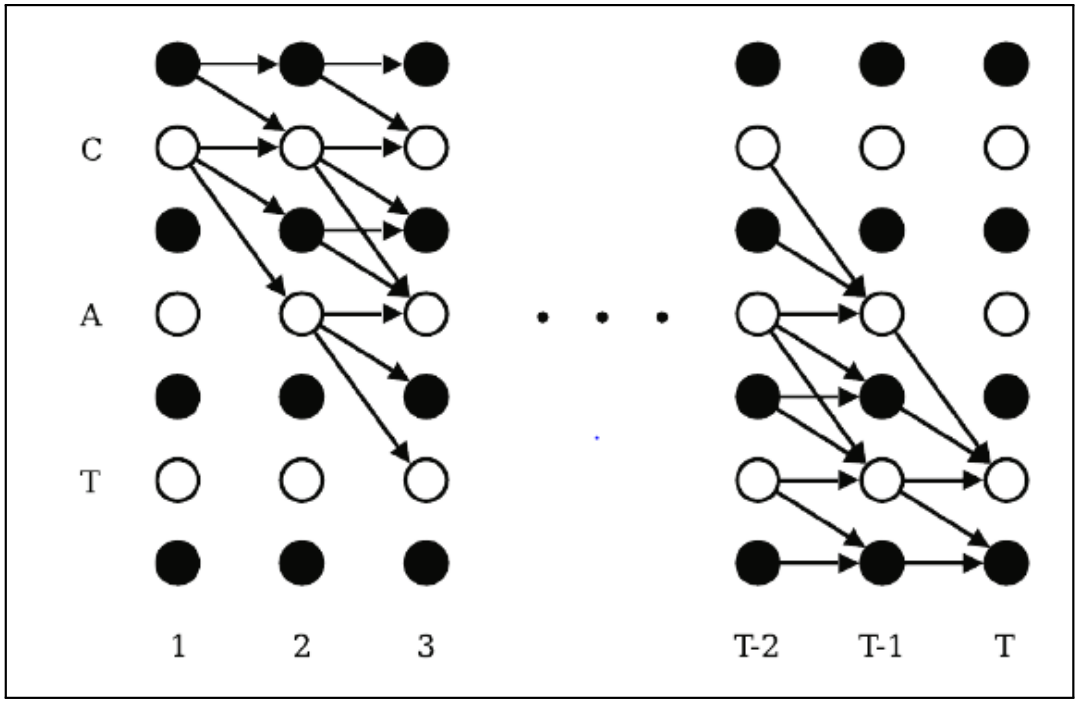
Toy example: '안녕하세요'

- '<space>': 0, '안': 1, '녕': 2, '하': 3, '세': 4, '요': 5, '<blank>': 6





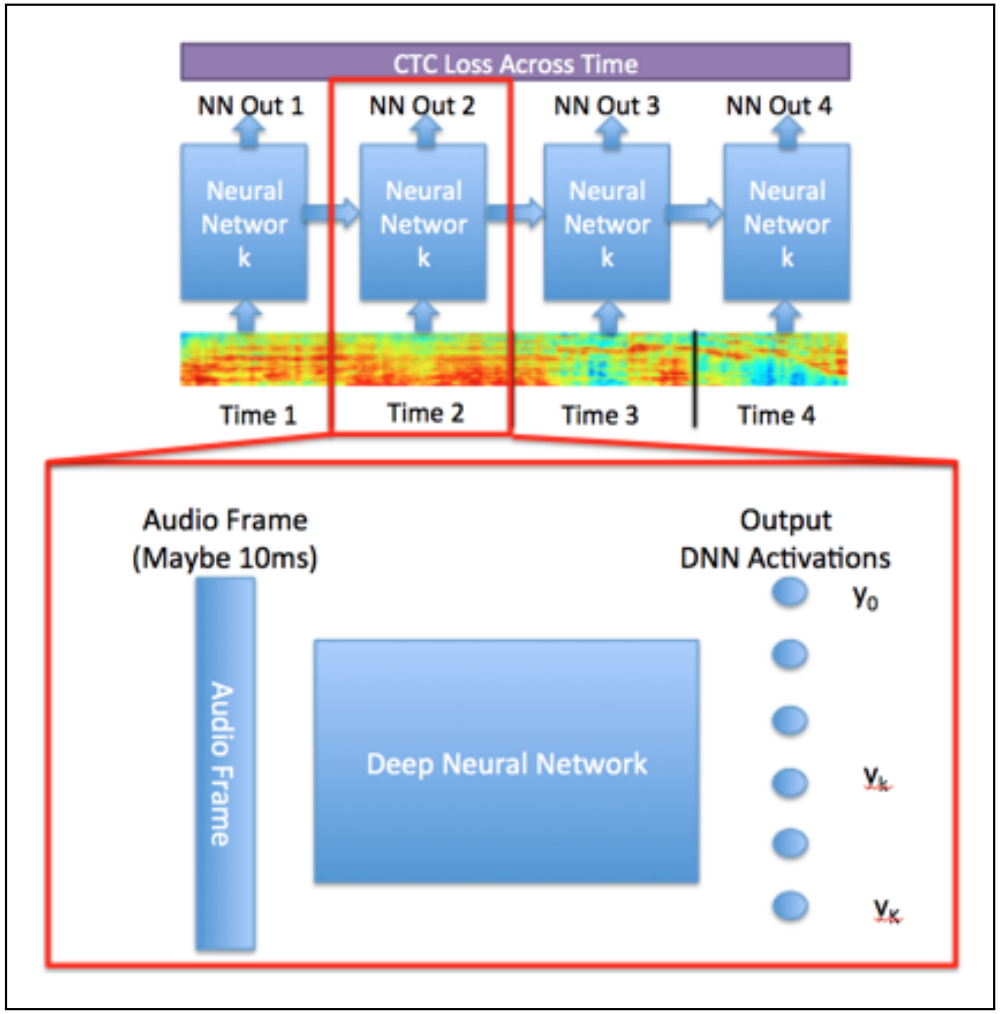




- Dynamic programming, α-β algorithm (Rabiner, 1989)
- integrate possibilities of trelis along possible paths
- tensorflow: tf.nn.ctc_loss, pytorch: nn.CTCLoss
- Google, Deepmind, Baidu, Facebook, Amazon, ...

RNN model

$$\begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_T \end{bmatrix} = \begin{bmatrix} RNN(\mathbf{h}_0, \mathbf{x}_1) \\ RNN(\mathbf{h}_1, \mathbf{x}_2) \\ \vdots \\ RNN(\mathbf{h}_{T-1}, \mathbf{x}_T) \end{bmatrix}$$



그 외의 응용들

- 비디오 자료 분류: 입력: 아미지들의 sequence, 프레임(frame)
- question answering, word2vec etc