Recurrent Neural Networks

Jaegul Choo (주재걸)
Korea University
https://sites.google.com/site/jaegulchoo/

Slides partly made by my student, Cheonbok Park, Yunjey Choi

Contents

1. Recurrent Neural Network

1-1. RNN의 여러가지 형태
1-2. Character-level LM
1-3. Vanishing Gradient Problem

2. LSTM과 GRU

2-1. LSTM이란 무엇인가
2-2. GRU란 무엇인가

1. Recurrent Neural Network

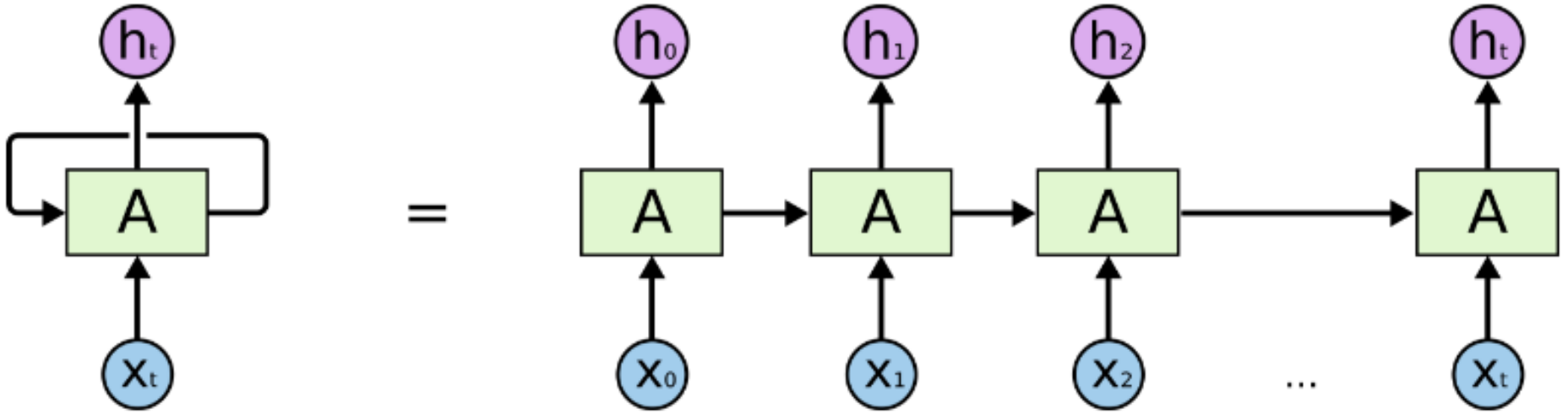
1-1. RNN의 여러가지 형태
1-2. RNN 모델
1-3. Character-level LM

01

RNN의 여러가지 형태

가장 기본적인 구조

리컬시브
CNN은 사실 연결 단계가 없는거. 이미지 하나만 내고, 다음엔 영향 받는게 없음
RNN은 이전의 이미지가 다음의 인풋으로 나오기에 이전의 것이 반영됨. 메모리 셀이 전달이 됩니다.



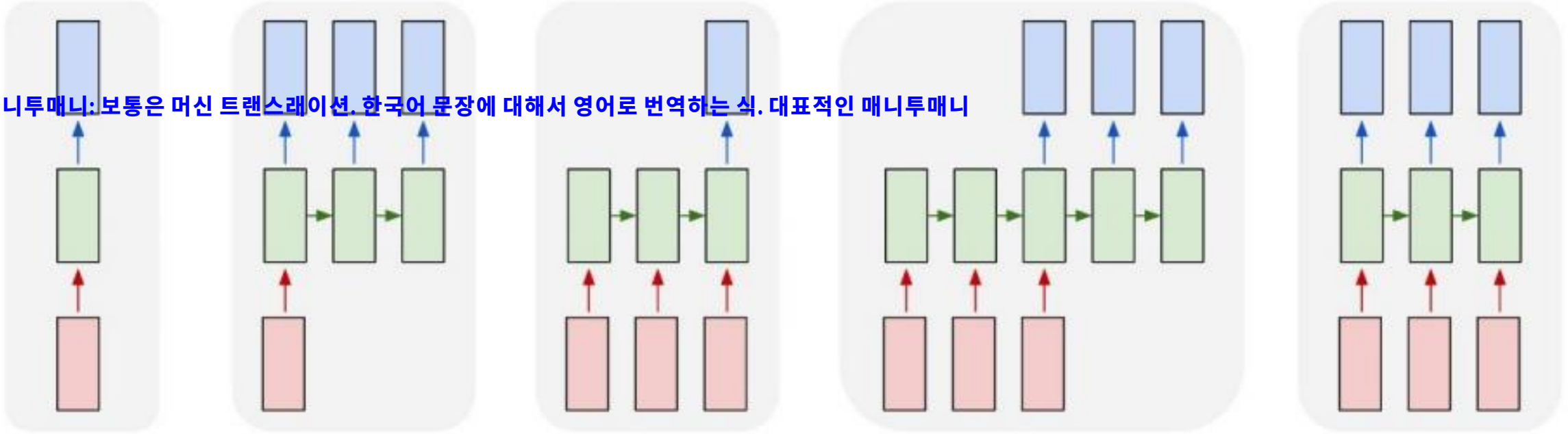
An unrolled recurrent neural network.

RNN의 여러가지 형태

#각각 구조를 알아야해요
- 백본을 알아야 프로젝트시 수여
원투원: 씨앤앤. 클래시피케이션하여 아웃풋 하나 나오는거
원투 매니: 이미지 캡셔닝. 이미지 주고 디스크립션 하게 되는것. 타자가 공을 친다. 피쳐가 공을 던지다.

기본 neural networks 구조

one to one | one to many | many to one | many to many | many to many



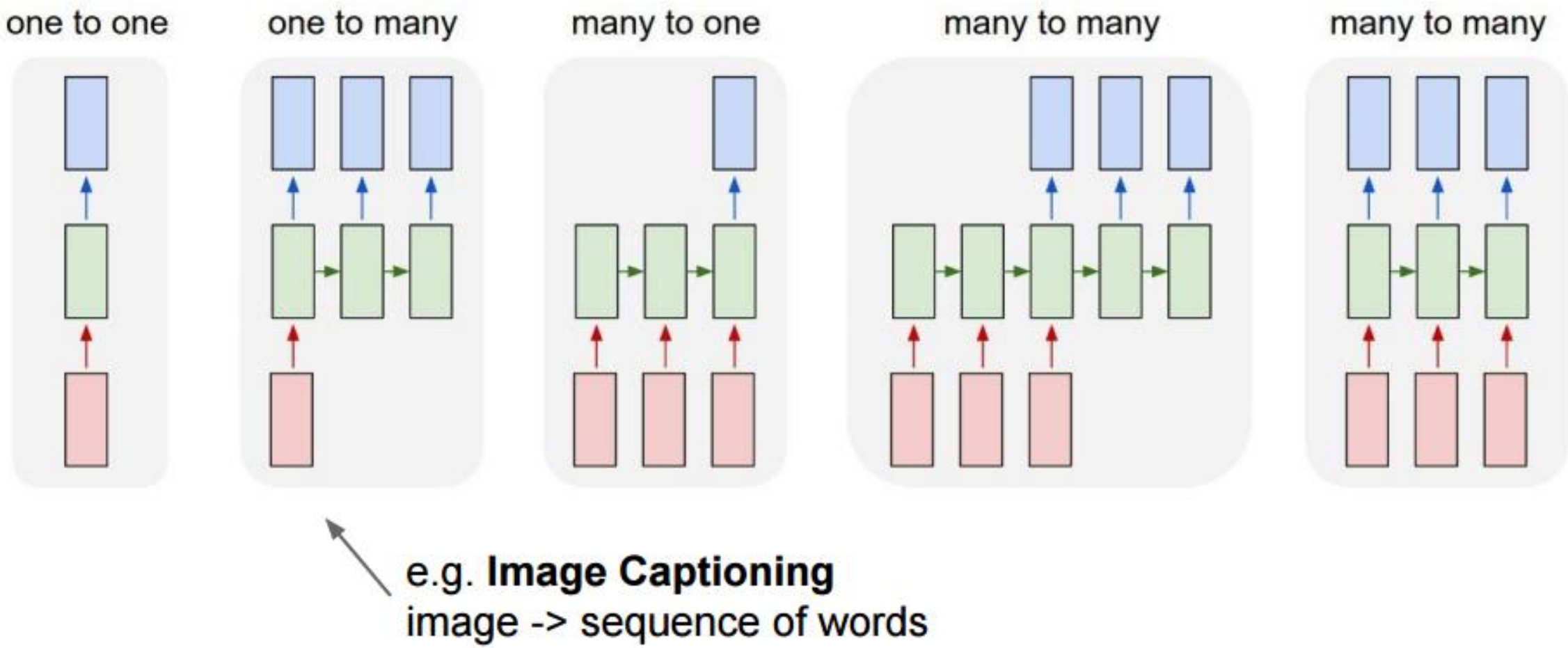
매니투원 : 여러가지가 하나로 가는 것으로, 시퀀스. nlp에서 시퀀스오면 분류하는 문제. 센티맨탈클래시피케이션. 시퀀스오브 월드로

매니투매니: 보통은 머신 트랜스래이션. 한국어 문장에 대해서 영어로 번역하는 식. 대표적인 매니투매니

Vanilla Neural Networks

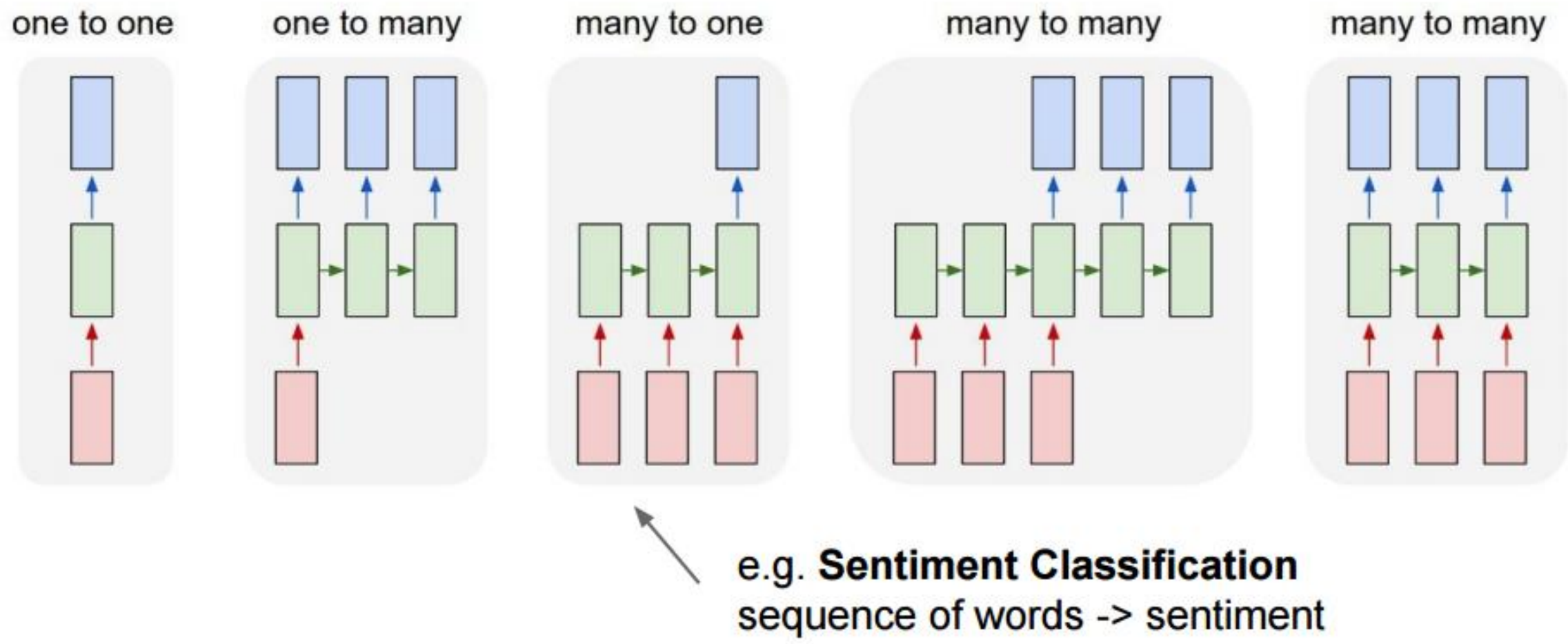
RNN의 여러가지 형태

one-to-many 형태



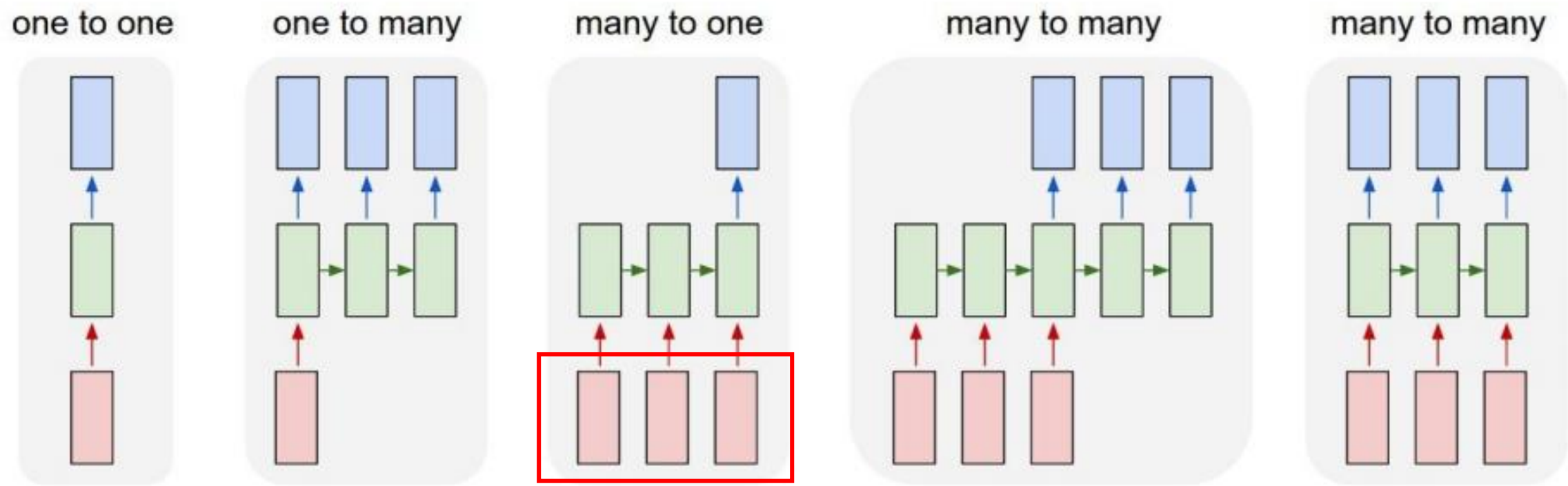
RNN의 여러가지 형태

many-to-one 형태



RNN의 여러가지 형태

many-to-one 형태

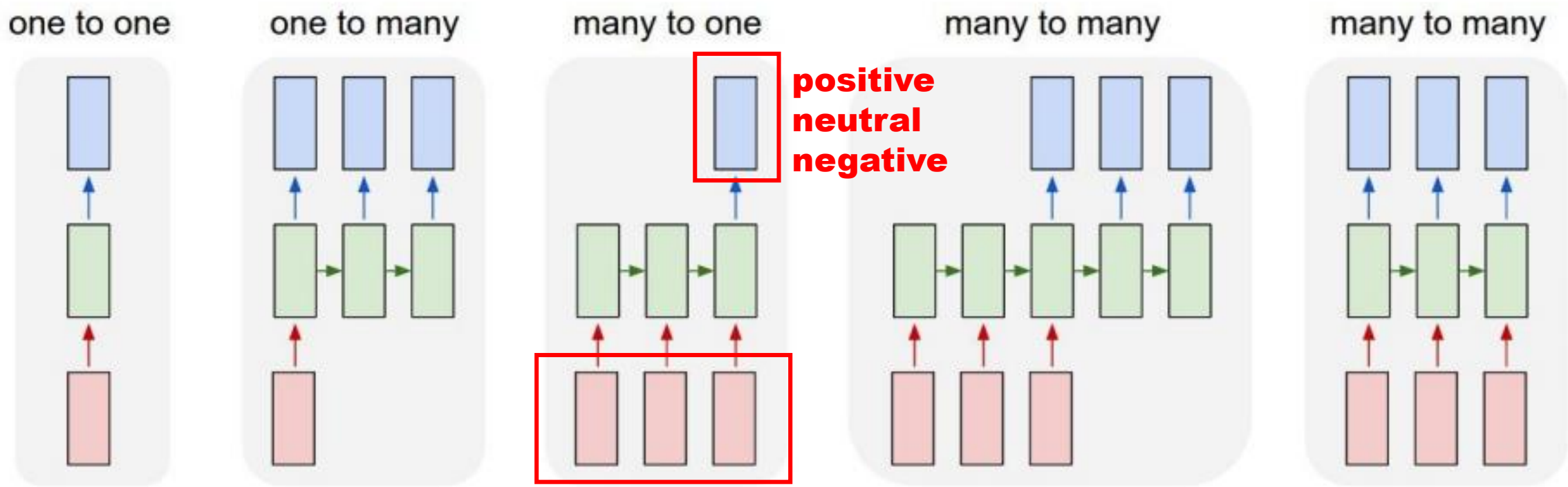


sequence of text

e.g. **Sentiment Classification**
sequence of words -> sentiment

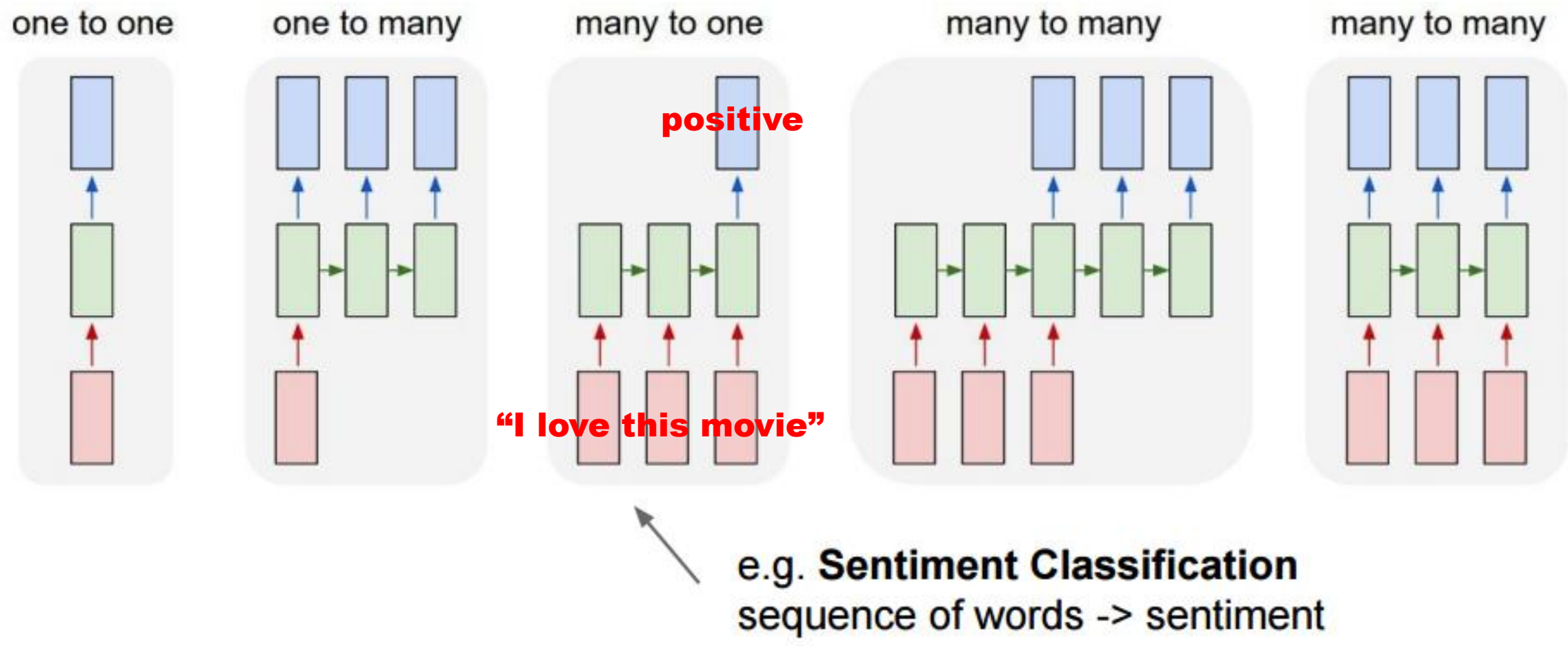
RNN의 여러가지 형태

many-to-one 형태



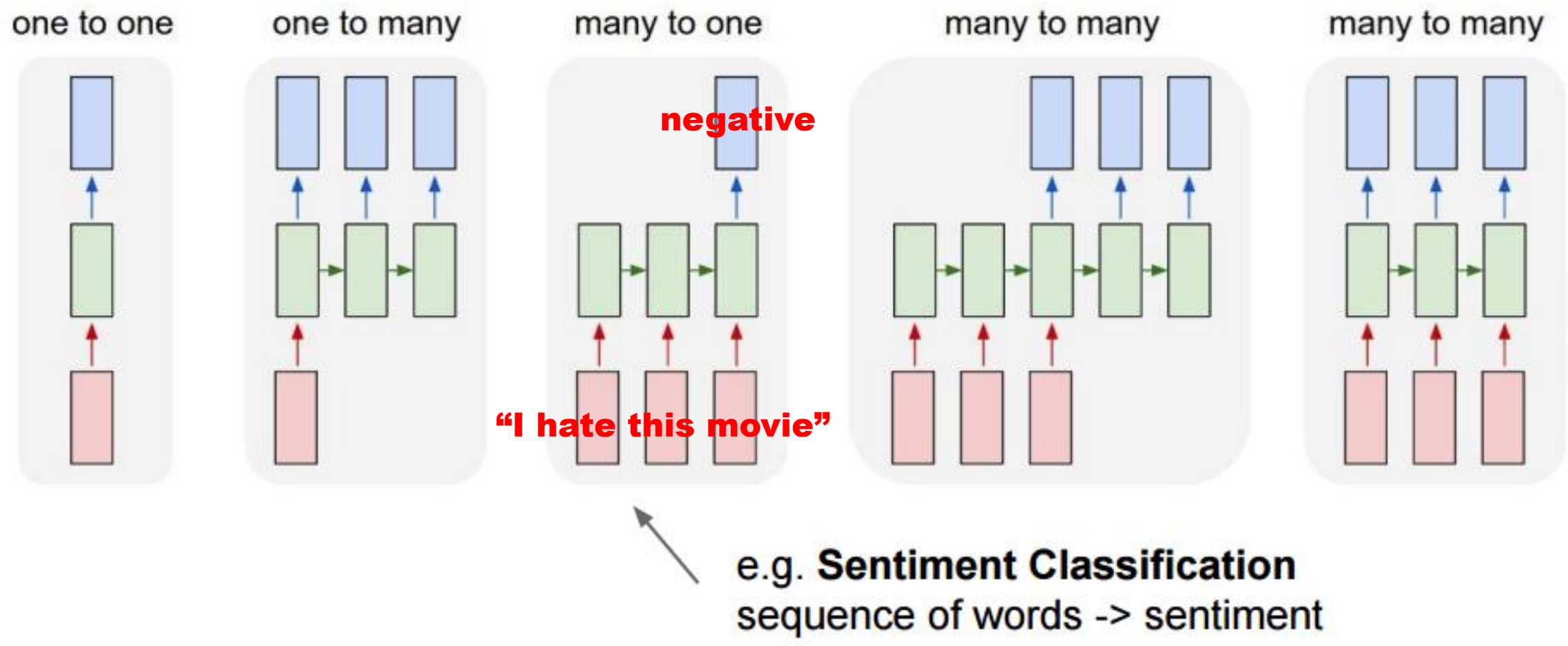
RNN의 여러가지 형태

many-to-one 형태



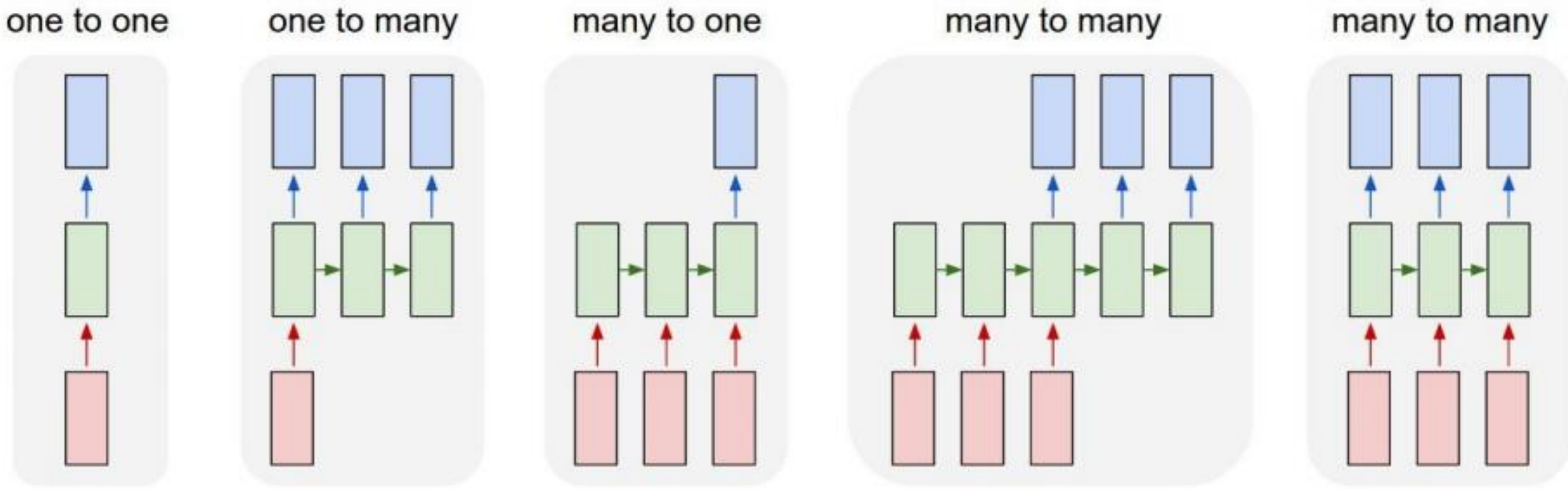
RNN의 여러가지 형태

many-to-one 형태



RNN의 여러가지 형태

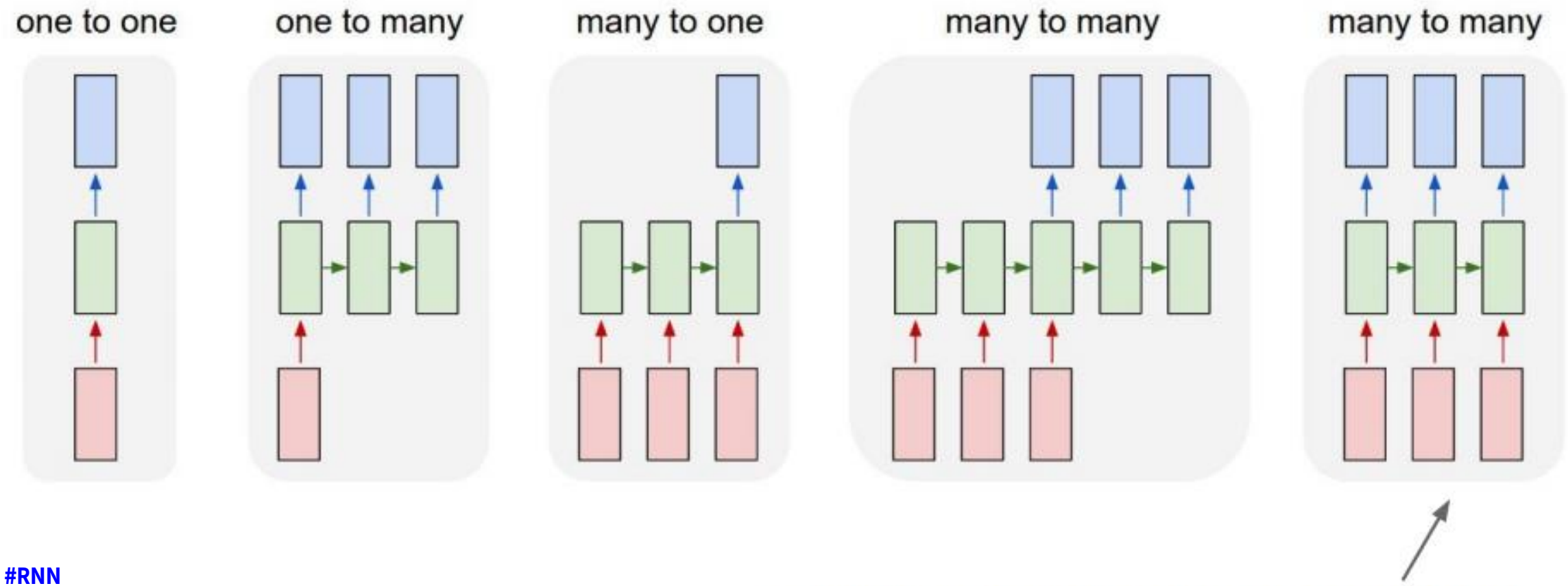
sequence-to-sequence 형태



e.g. **Machine Translation**
seq of words -> seq of words

RNN의 여러가지 형태

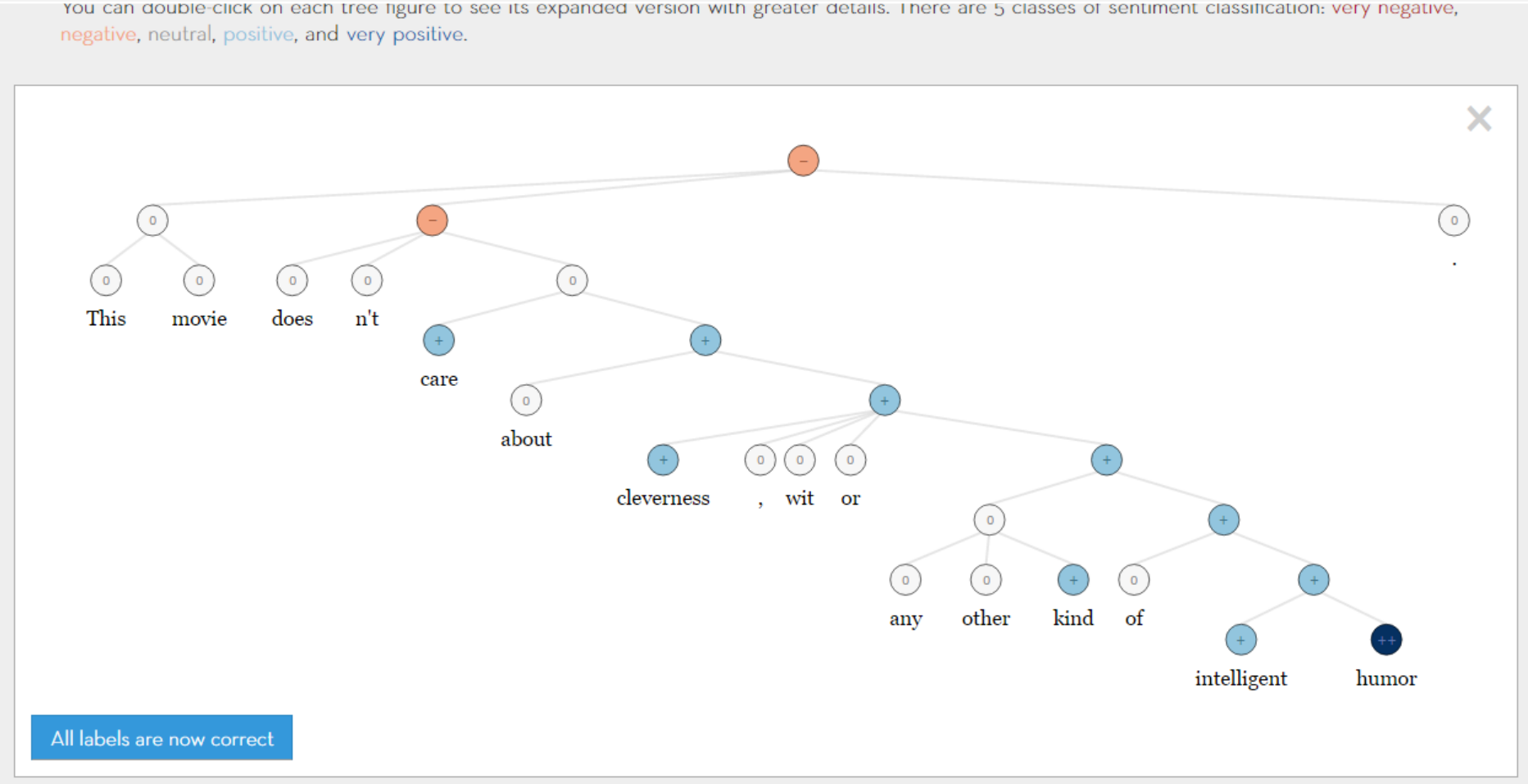
sequence-to-sequence 형태



#RNN
- 펑션은 시퀀스마다 동일하게 공유되고 중요한건 '올드스테이트'
- 이전 스테이트와 현재를 고려하여 현재의 뉴 스테이트를 만드는것. 리컬시브한 연산
- 컴퓨터입장에서 이 느낌은 튜링 머신과 유사함.
- 튜링 컴플릿하다고 해요. 알엔에은 모든 계싼 문제 풀고 컴퓨터 모델링

Sentiment Analysis

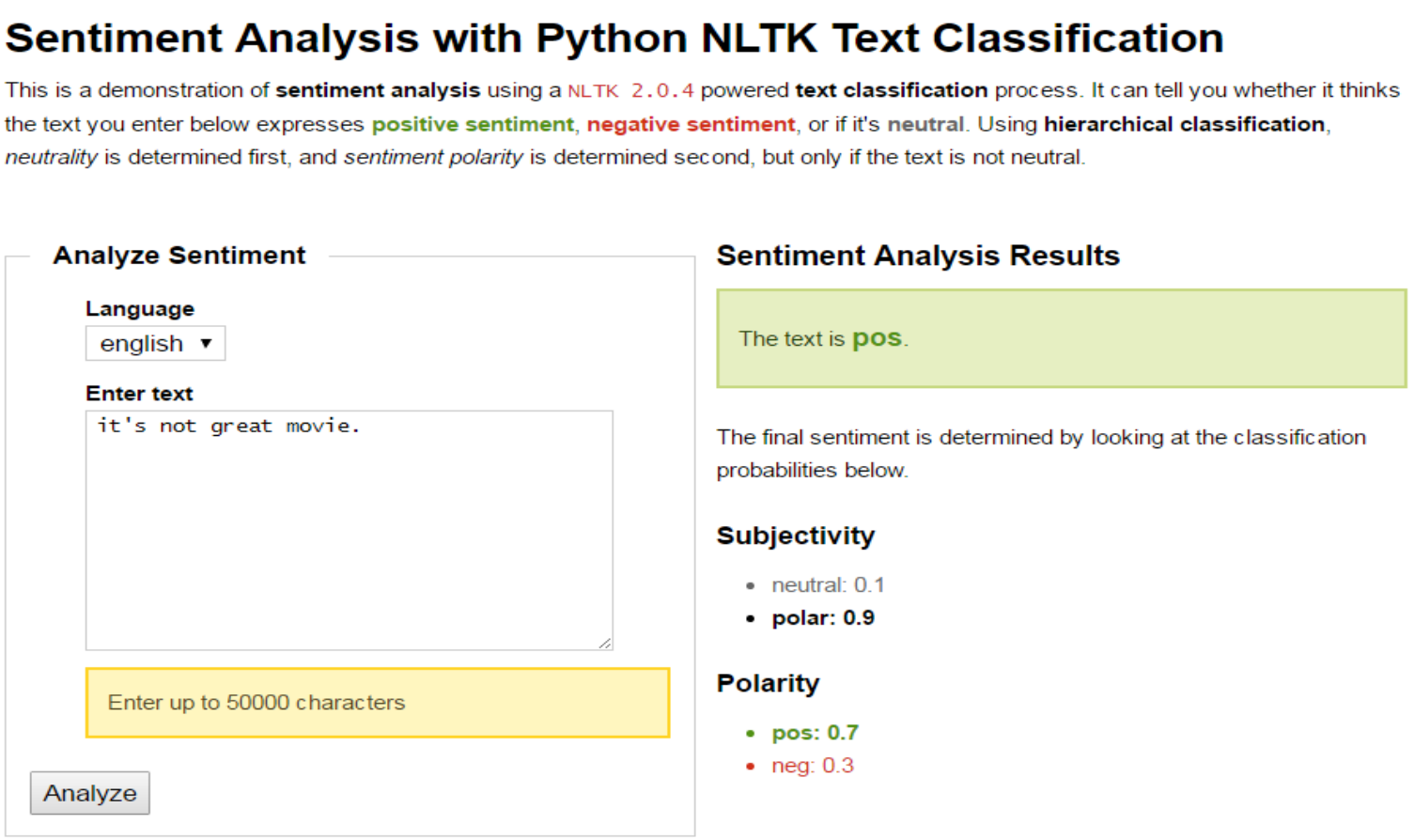
Deep Learning 기술을 사용한 경우



[Sentiment Analysis - The Stanford Natural Language Processing Group]

Sentiment Analysis

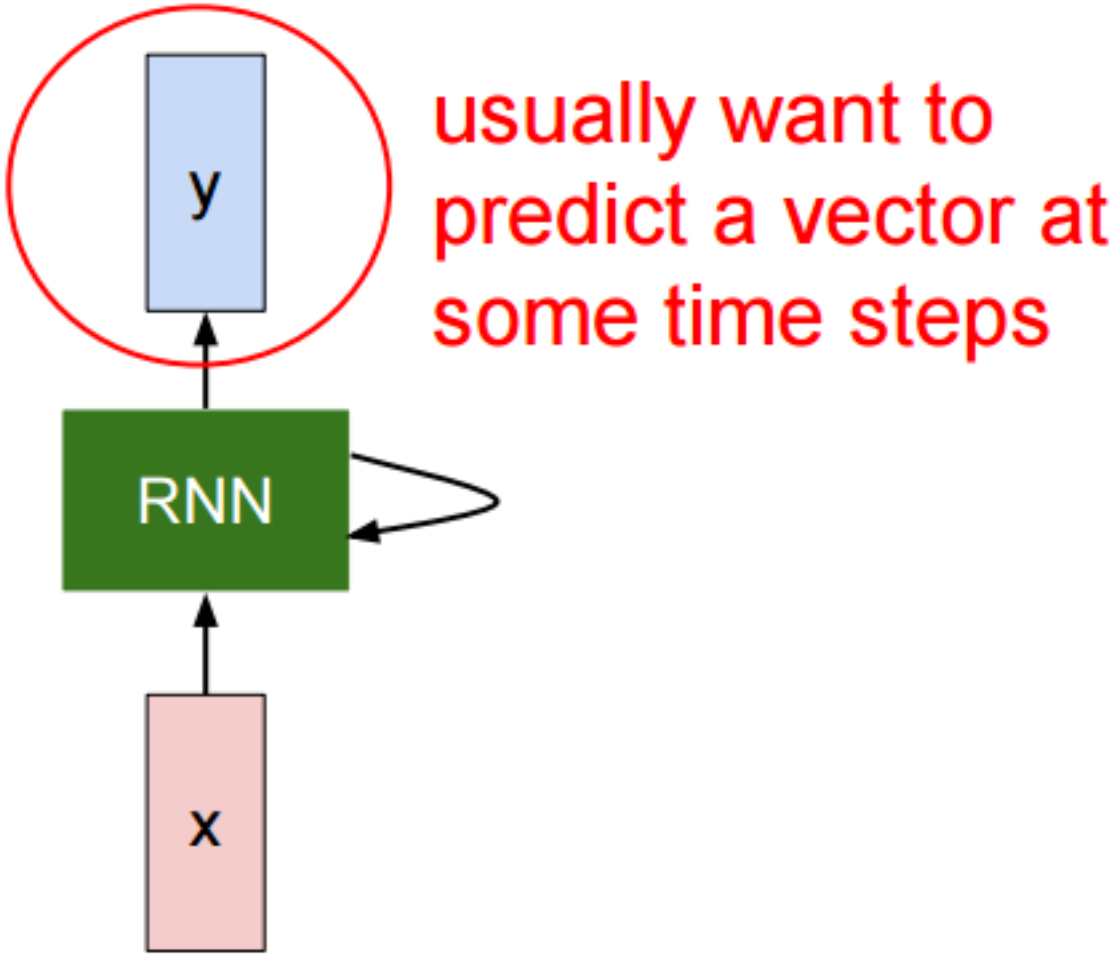
Deep Learning 기술을 사용하지 않은 경우



[Python NLTK Sentiment Analysis with Text Classification Demo]

Recurrent Neural Network

RNN의 탄생 배경



Recurrent Neural Network

RNN의 hidden state값을 어떻게 계산할까?

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

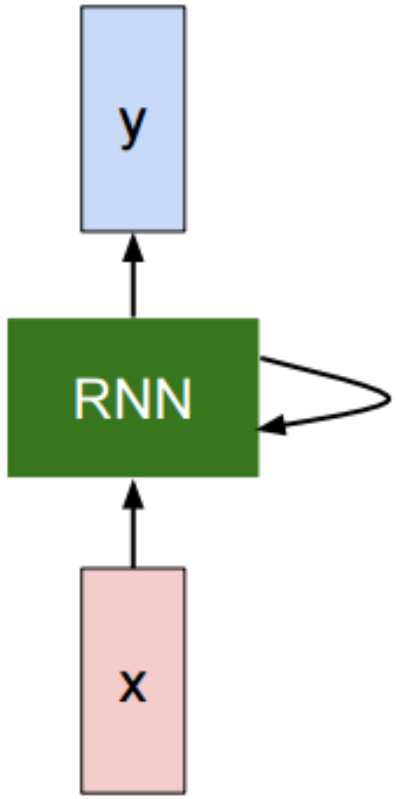
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



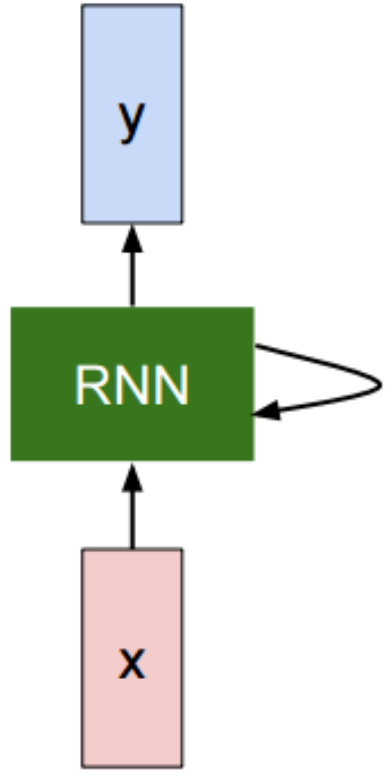
Recurrent Neural Network

RNN의 hidden state값을 어떻게 계산할까?

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

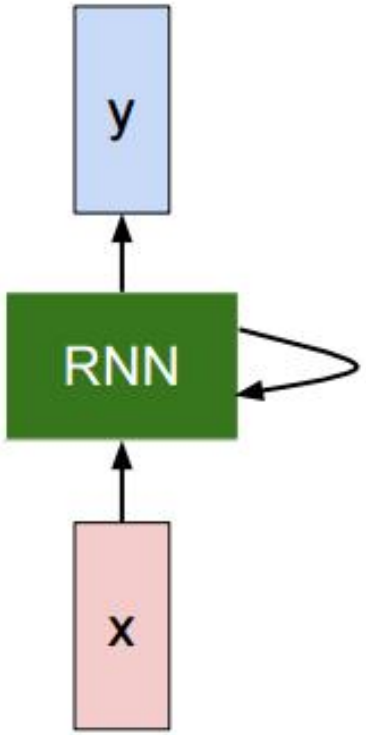
Notice: the same function and the same set of parameters are used at every time step.



Recurrent Neural Network

RNN의 output값을 어떻게 계산할까?

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

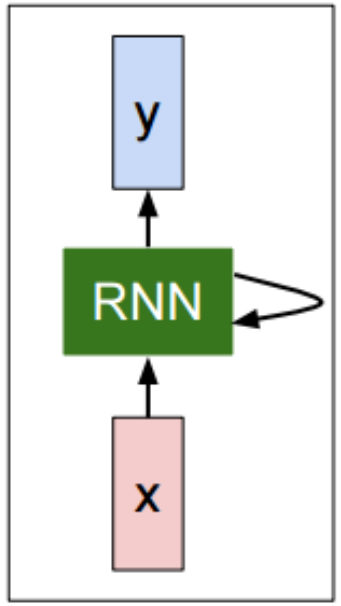
Character-level Language Model

RNN에게 “hello”를 생성할 수 있도록 학습시켜보자

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



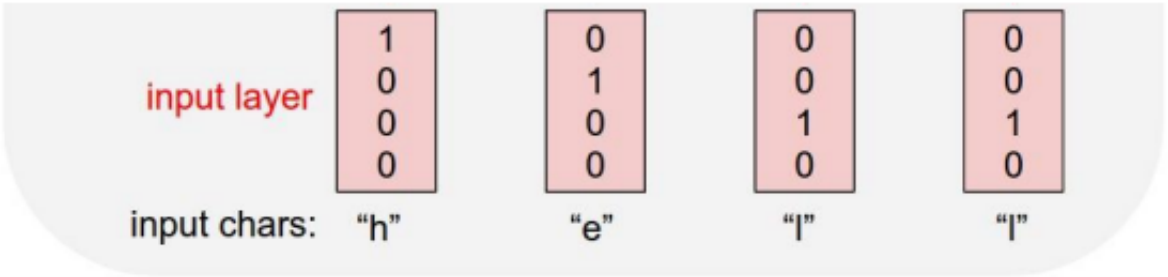
Character-level Language Model

RNN이 “hello”를 생성할 수 있도록 학습시켜보자

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Character-level Language Model

RNN이 “hello”를 생성할 수 있도록 학습시켜보자

Character-level language model example

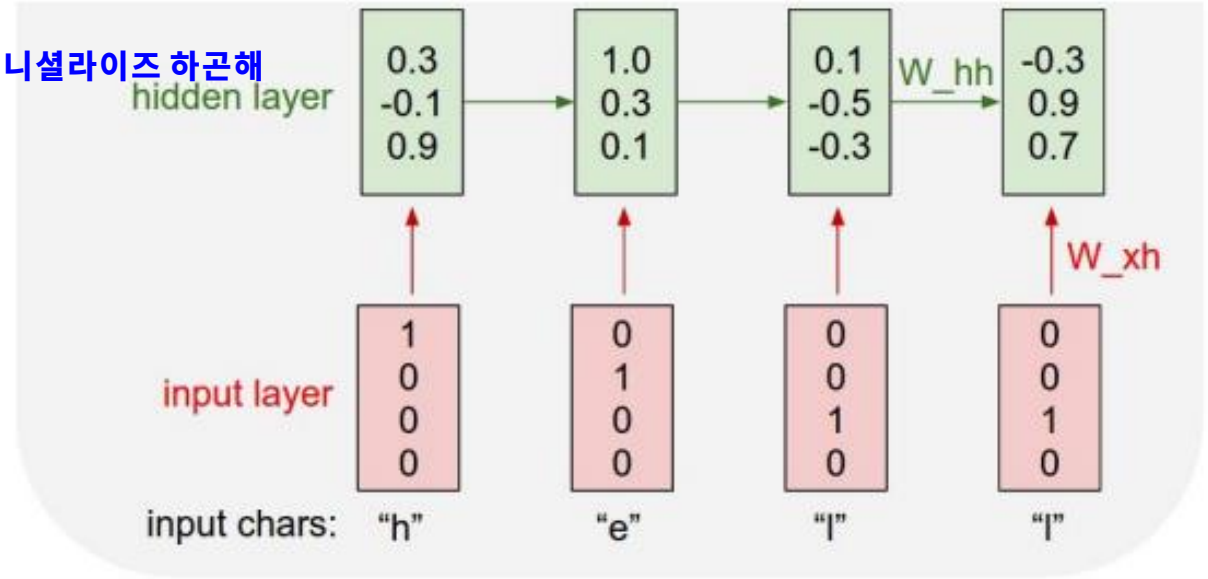
Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

캐릭터 레벨 랭귀지 모델
- 아마 의아할텐데
- 인풋이 원핫으로 보이죠. 1과0만있는듯
- 근데 실제로는 원핫을 임배딩으로 두는걸 하나 추가적으로 두고 그 임배딩에
- 더블류 엑스 에이치를 곱하는 식으로 사용하곤합니다.
- 그리고 이전의 히든 시테이트 고려해서 웨이트 에이치에이치를 탄에이치 펑션 이용해 나타남

에ㅐ이치 원
- 초반에 영으로 이니셜라이즈 하곤해

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Character-level Language Model

RNN이 “hello”를 생성할 수 있도록 학습시켜보자

시퀀스투시퀀스 주의
-트레이닝과 테스트 인풋이 달라요
- 실제로 테스트 한다 생각을 하면 h넣어 e 나온다 하면, 이 넣어서 아이 나오고.
- 이렇게 하면 트레이닝은 애매. 알엔엔이 처음에 랜덤하니까 아무거나 넣겠죠
- 예를 들어서 에이치> 오>엘>아이>에스가 나오는 식이면 총체적 난관.
-

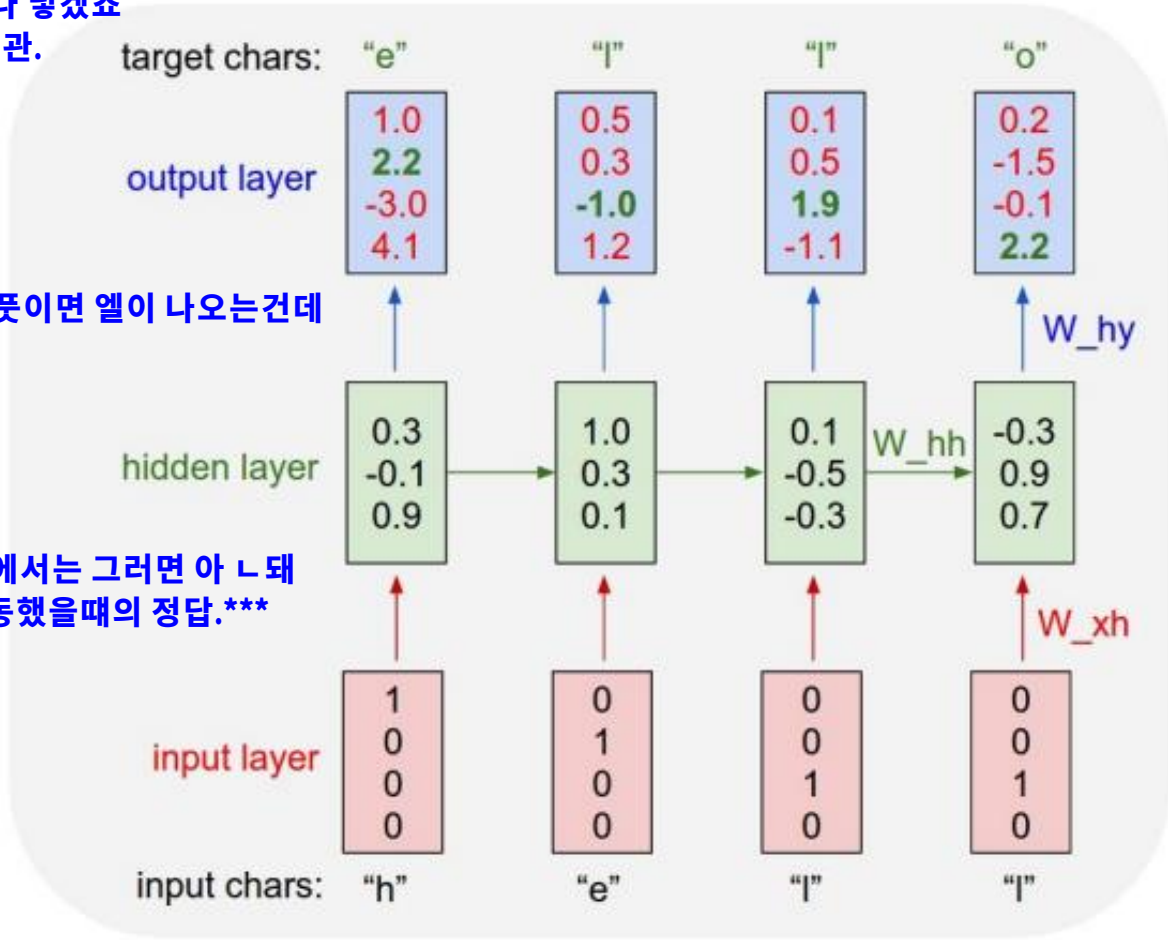
Character-level language model example

로스 펑션이라고 하면 에이치 오가 나왔을때, 엘이 오랑 에이치가 인풋이면 엘이 나오는건데
- 이거 가정이 잘못되었죠

Vocabulary: [h,e,l,o]

- 만약에 테스트 할때는 결과를 다음 인풋으로 넣는단건데, 트레이닝에서는 그러면 아 ㄴ돼
- 트레이닝은 뭐가 나오든 그라운드 (정답)을 넣어줘요. 올바르게 행동했을때의 정답.***

Example training sequence: “hello”



- 문제) 하지만 이런 차이가 되는게

Character-level Language Model

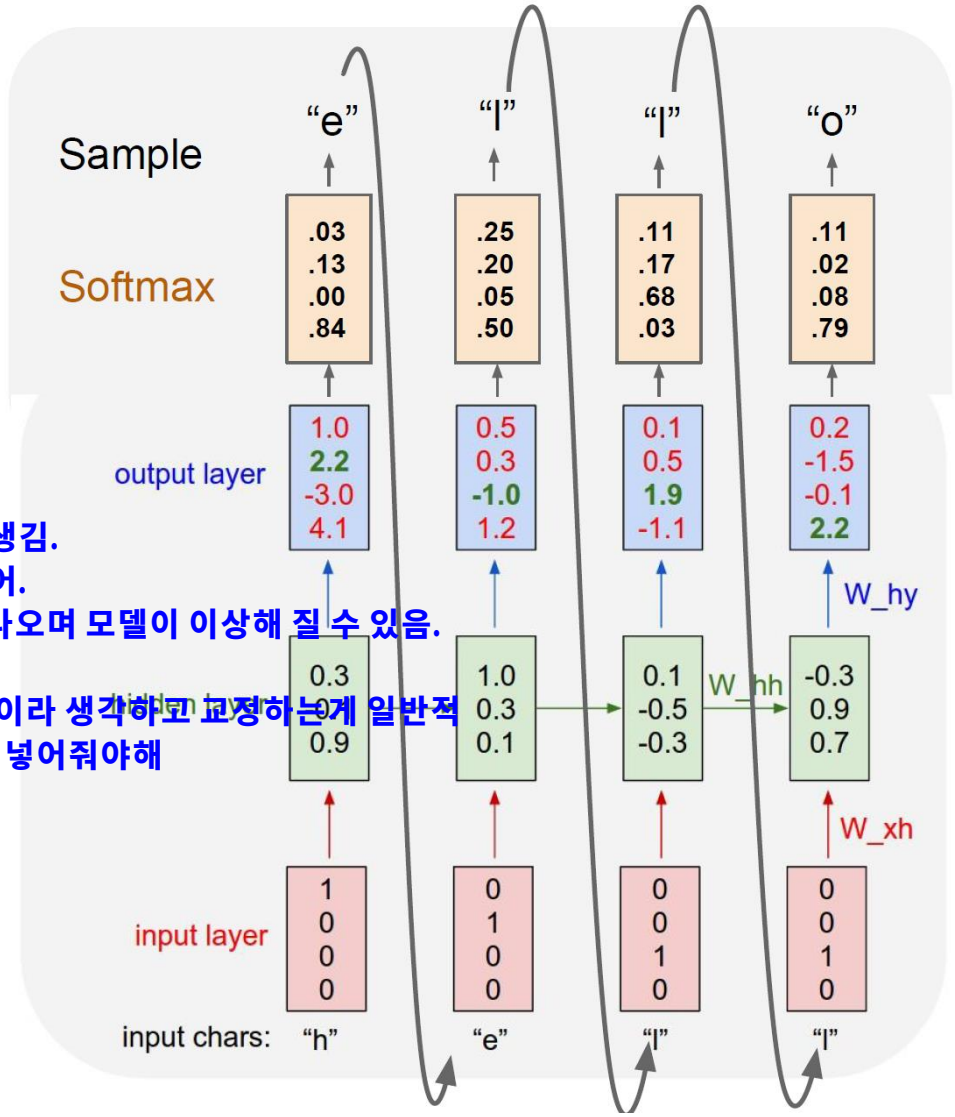
RNN이 “hello”를 생성할 수 있도록 학습시켜보자

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a time,
feed back to model

- 문제) 하지만 이런 차이가 되는게 문제
  - 알엔엔에서는 항상 인풋이 좋은 값이라는 문제가 생김.
  - 모델이 잘못 배워서 순간적으로 실수를 하게 되었어.
  - 갑자기 h e l l 하고 e 가 나와서 깨지면 계속 e e e 나오며 모델이 이상해 질 수 있음.
- 해결 테스트)많이 있음
  - 일단 처음 학습할때는 그랑누드 넣고 하나씩 정답이라 생각하고 교정하는게 일반적
  - 실수말것) 생성된거 할때는 정답 말고 그대로 인풋 넣어줘야해



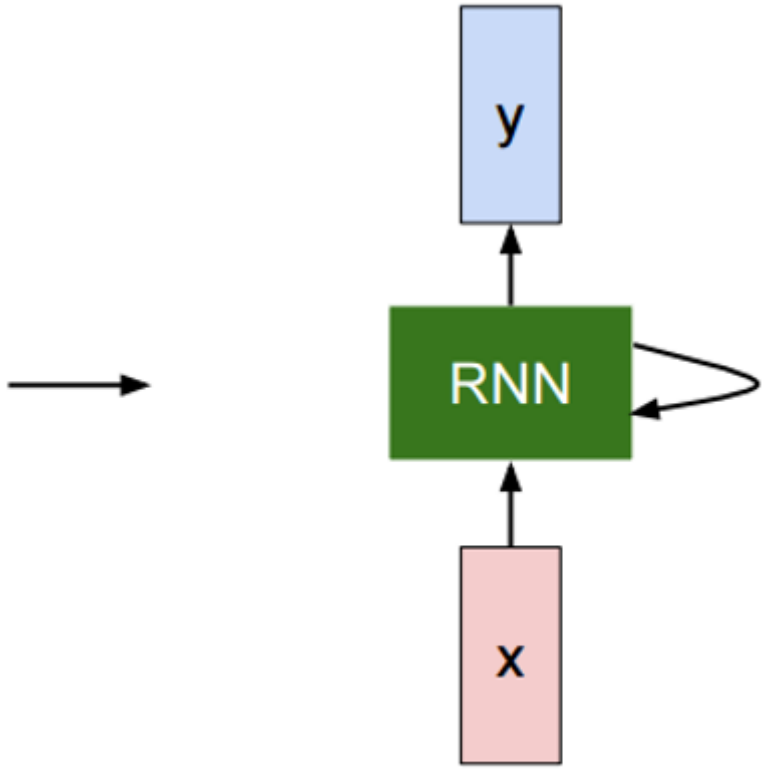
Character-level Language Model

RNN에게 희곡을 학습시키면?

Sonnet 116 - Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.



Character-level Language Model

RNN 학습과정

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Character-level Language Model

RNN 학습결과

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Character-level Language Model

RNN의 논문 생성

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

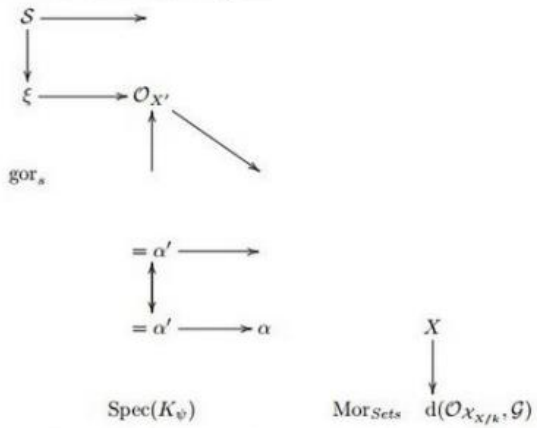
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_t}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{\eta}})$$

is an isomorphism of covering of $\mathcal{O}_{X_t}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

Character-level Language Model

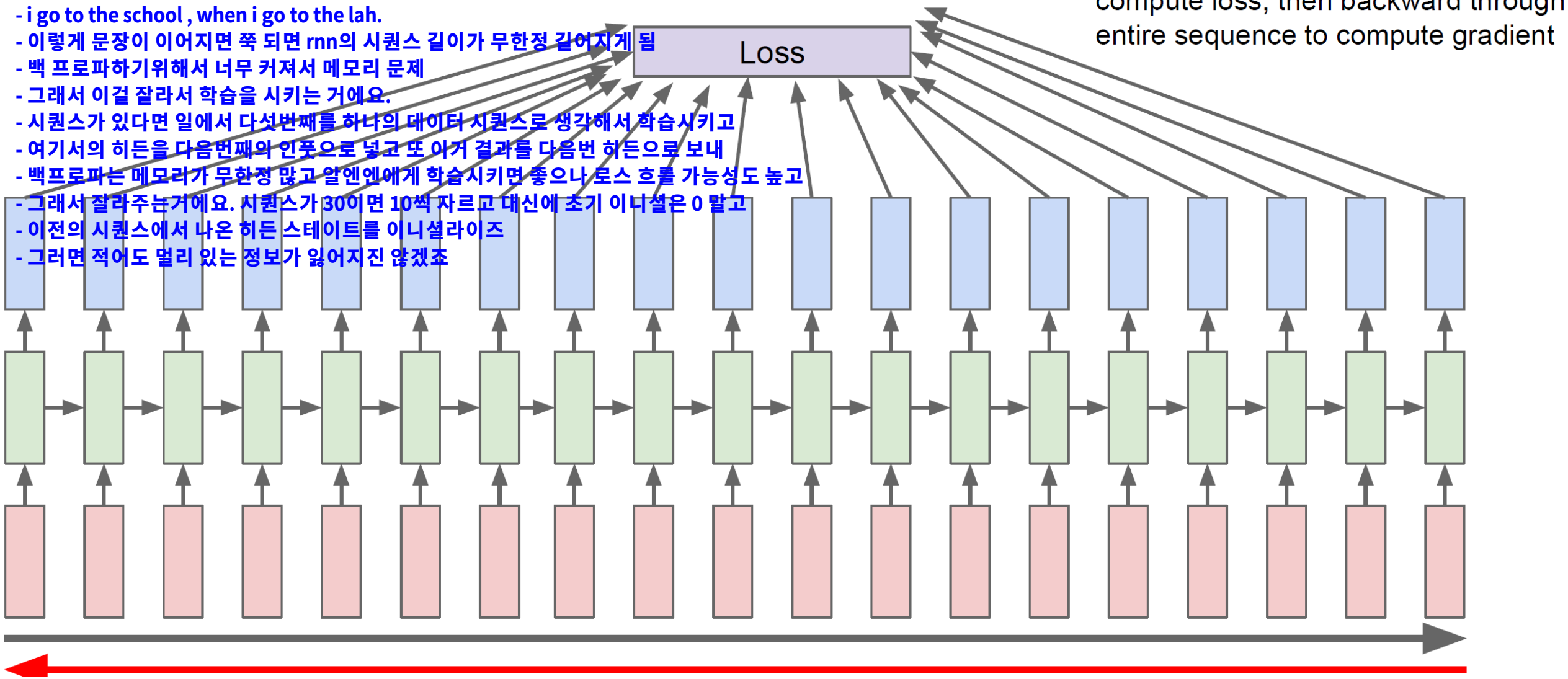
RNN의 C code 생성

```c
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000ffffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

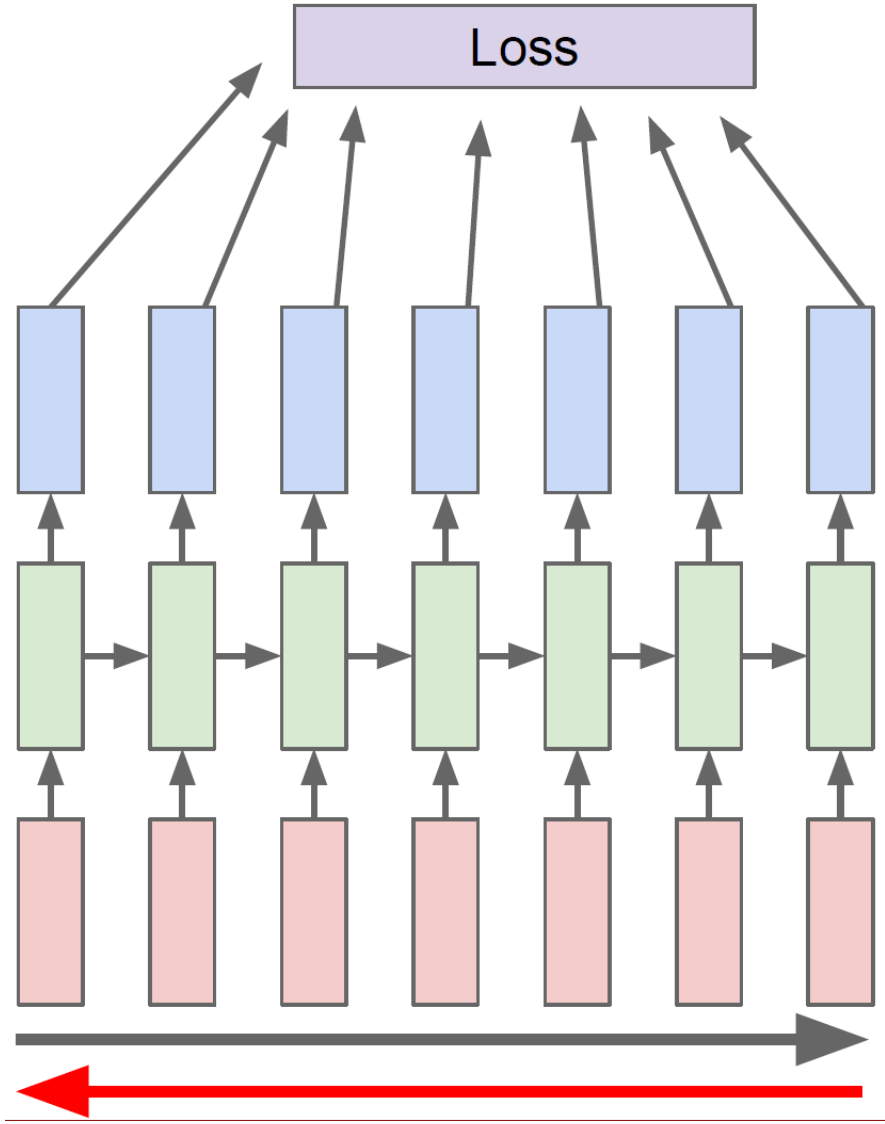
Generated
C code

Backpropagation through Time (BPTT)

BPTT
- Backpropagation through time
- 우리가 문장이 존재하잖아요. 이 랭기지 모델을 한다 했을때
- i go to the school , when i go to the lah.
- 이렇게 문장이 이어지면 쭉 되면 rnn의 시퀀스 길이가 무한정 길어지게 됨
- 백 프로파하기위해서 너무 커져서 메모리 문제
- 그래서 이걸 잘라서 학습을 시키는 거에요.
- 시퀀스가 있다면 일에서 다섯번째를 하나의 데이터 시퀀스로 생각해서 학습시키고
- 여기서의 히든을 다음번째의 인풋으로 넣고 또 이거 결과를 다음번 히든으로 보내
- 백프로파는 메모리가 무한정 많고 알엔엔에게 학습시키면 좋으나 로스 흐를 가능성도 높고
- 그래서 잘라주는거에요. 시퀀스가 30이면 10씩 자르고 대신에 초기 이니셜은 0 말고
- 이전의 시퀀스에서 나온 히든 스테이트를 이니셜라이즈
- 그러면 적어도 멀리 있는 정보가 잃어지진 않겠죠



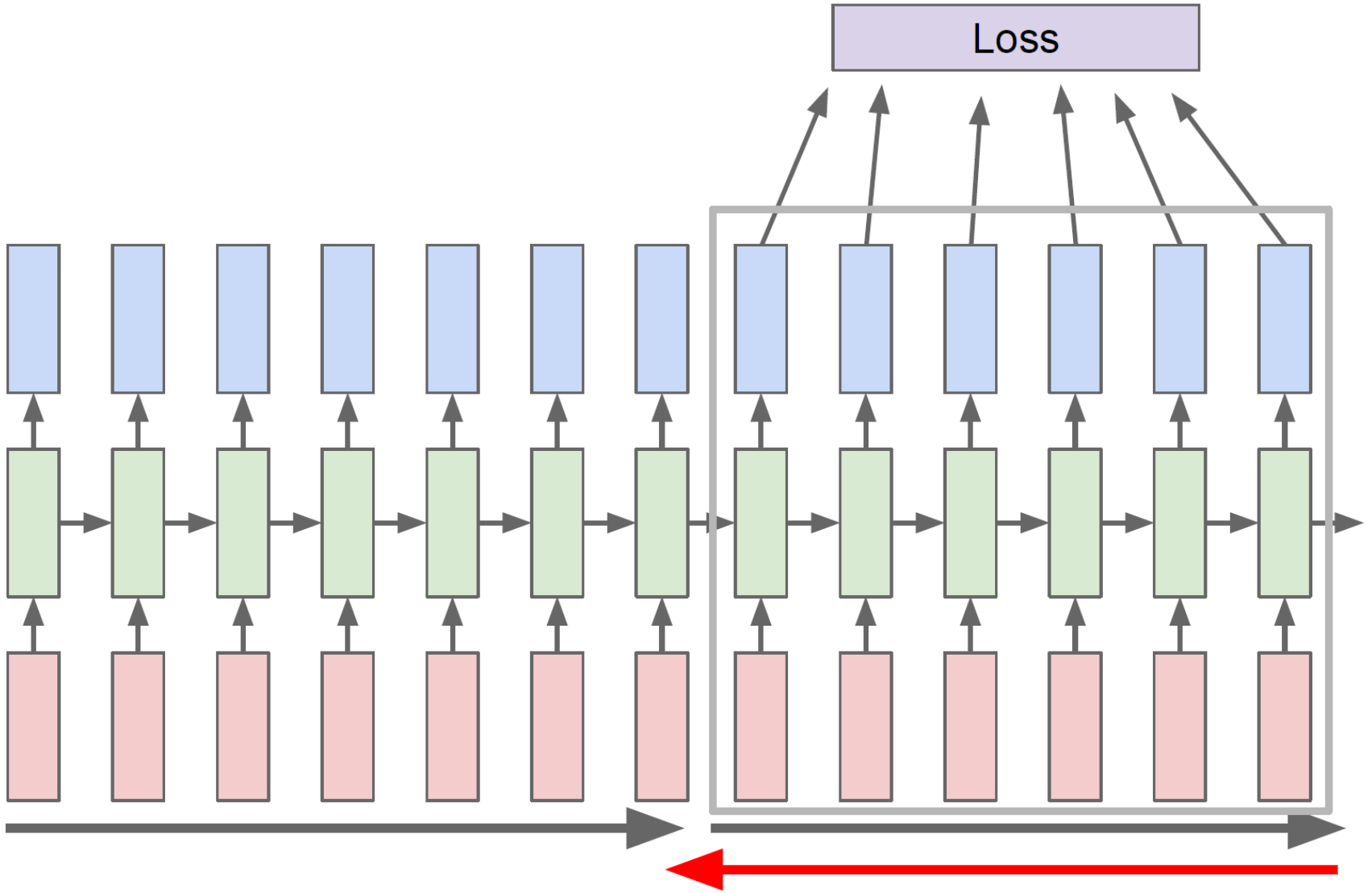
Truncated Backpropagation through Time



Run forward and backward through chunks of the sequence instead of whole sequence

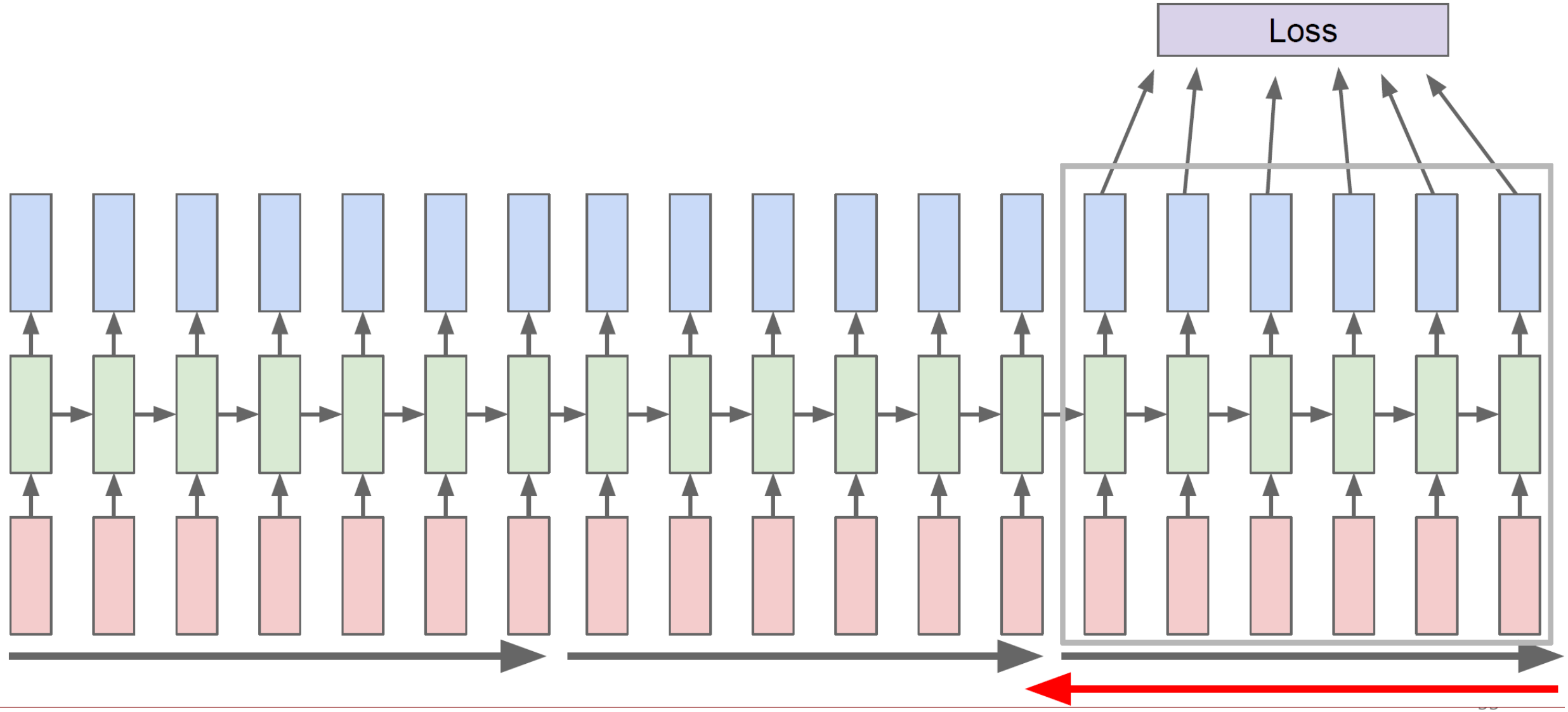
BPTT
- Backpropagation through time
- 우리가 문장이 존재하잖아요. 이 랭기지 모델을 한다 했을때
- i go to the school , when i go to the lah.
- 이렇게 문장이 이어지면 쭉 되면 rnn의 시퀀스 길이가 무한정 길어지게 됨
- 백 프로파하기위해서 너무 커져서 메모리 문제
- 그래서 이걸 잘라서 학습을 시키는 거에요.
- 시퀀스가 있다면 일에서 다섯번째를 하나의 데이터 시퀀스로 생각해서 학습시키고
- 여기서의 히든을 다음번째의 인풋으로 넣고 또 이거 결과를 다음번 히든으로 보내
- 백프로파는 메모리가 무한정 많고 알엔엔에게 학습시키면 좋으나 로스 흐를 가능성도 높고
- 그래서 잘라주는거에요. 시퀀스가 30이면 10씩 자르고 대신에 초기 이니셜은 0 말고
- 이전의 시퀀스에서 나온 히든 스테이트를 이니셜라이즈
- 그러면 적어도 멀리 있는 정보가 잃어지진 않겠죠

Truncated Backpropagation through Time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through Time



min-char-rnn.py

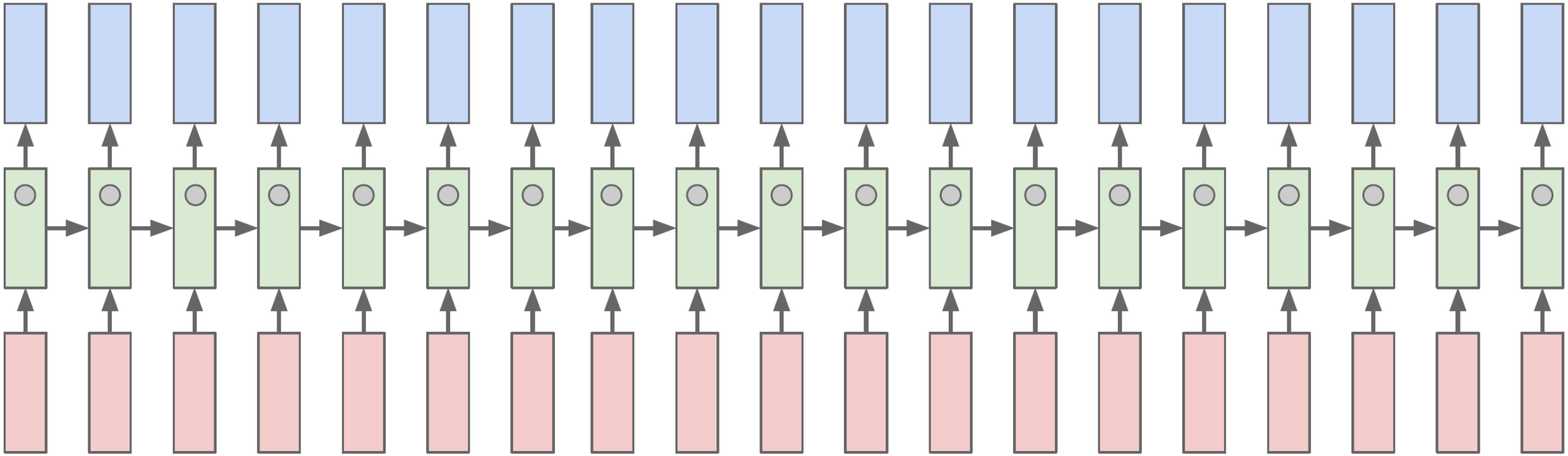
min-char-rnn.py gist: 112 lines of Python

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

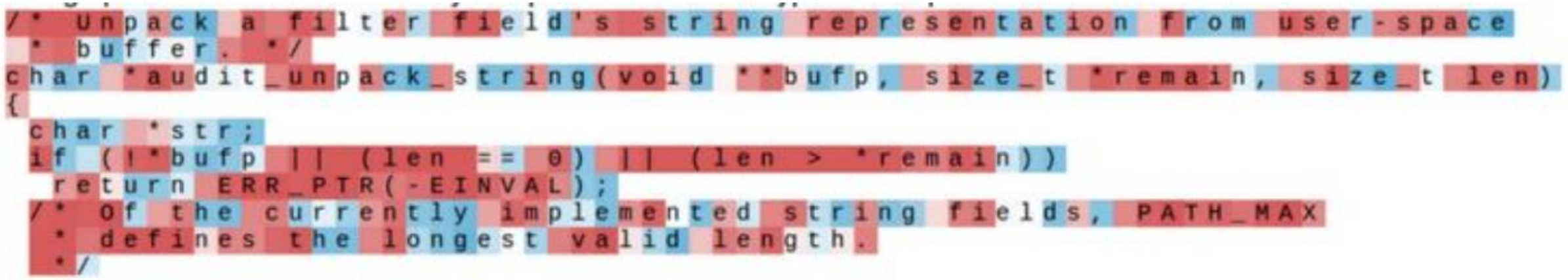
https://gist.github.com/karpathy/d4dee566867f8291f086

Searching for Interpretable Cells



Character-level Language Model

RNN 어떻게 동작하는가

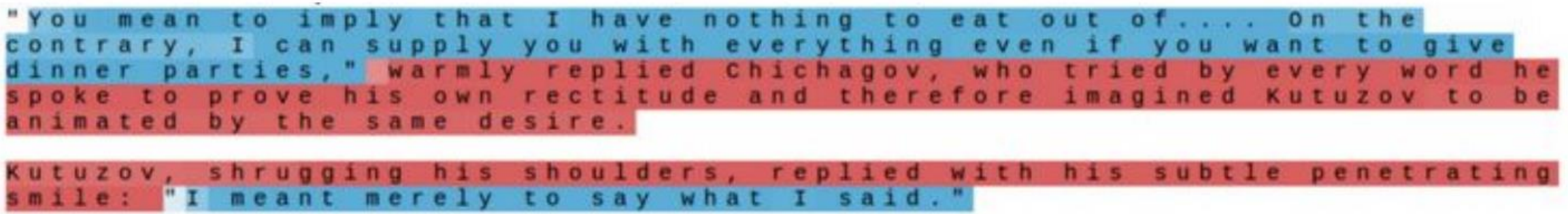


```
/*  Unpack a filter field's string representation from user-space
 *  buffer.  */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /*  Of the currently implemented string fields, PATH_MAX
   *  defines the longest valid length.
   */
```

Character-level Language Model

RNN 어떻게 동작하는가

누군가가 한 테스트
- 인용마크일때 긍정
- 인용 끝나면 빨갛고해
- 그럼 자기와 이전 스테이ㅋ트 봐서 바뀌는게 없으면 그대로 부정으로 하고, 인용이 나오면 긍정이 나오게 하는식
- 그래서 인용마크가 나오면 변경하는 식으로 동작하게 할 수도 있음



quote detection cell

Character-level Language Model

RNN 어떻게 동작하는가

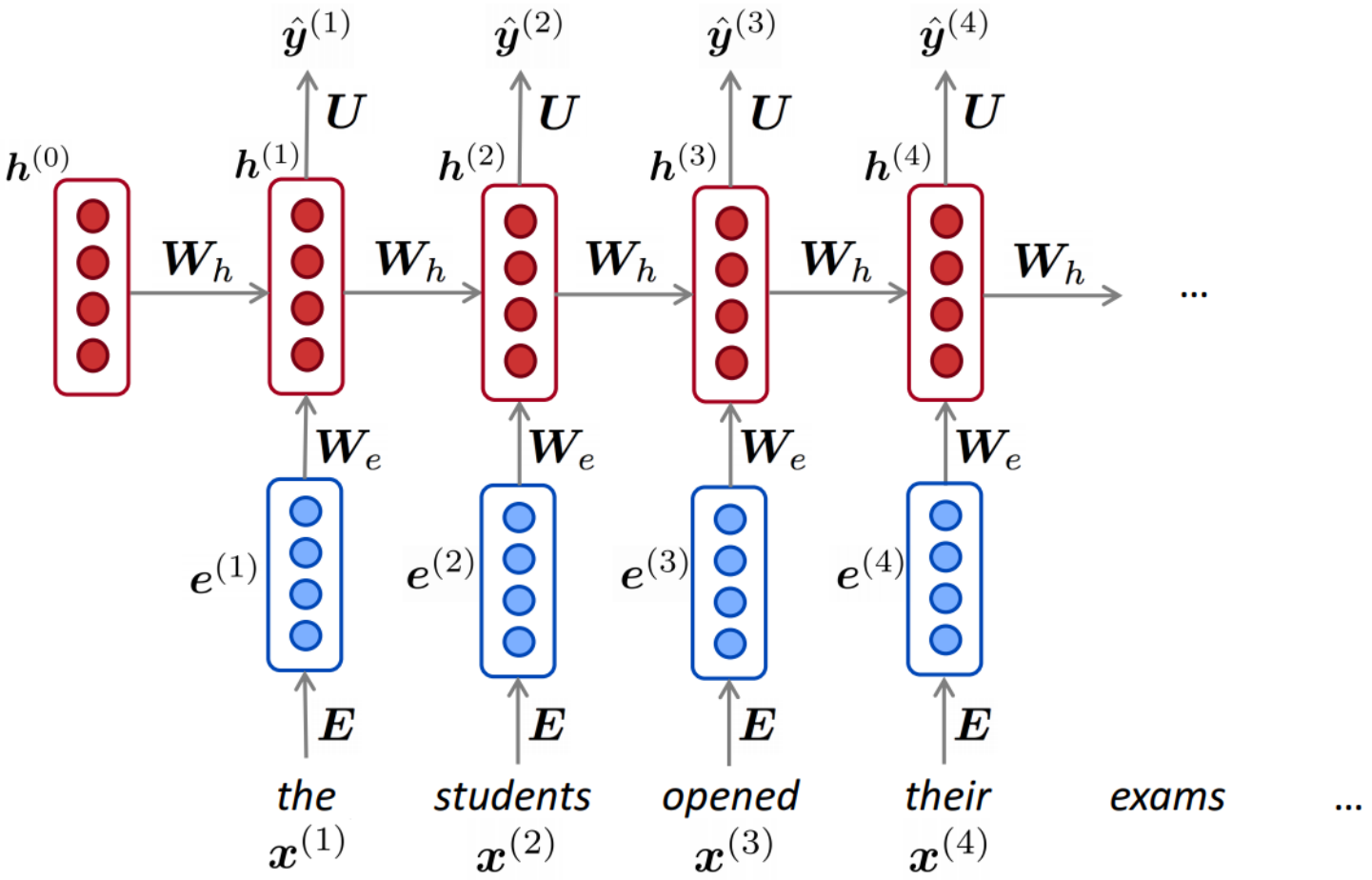
```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

if statement cell

Vanishing Gradient Problem

Gradient Vanishing Problem

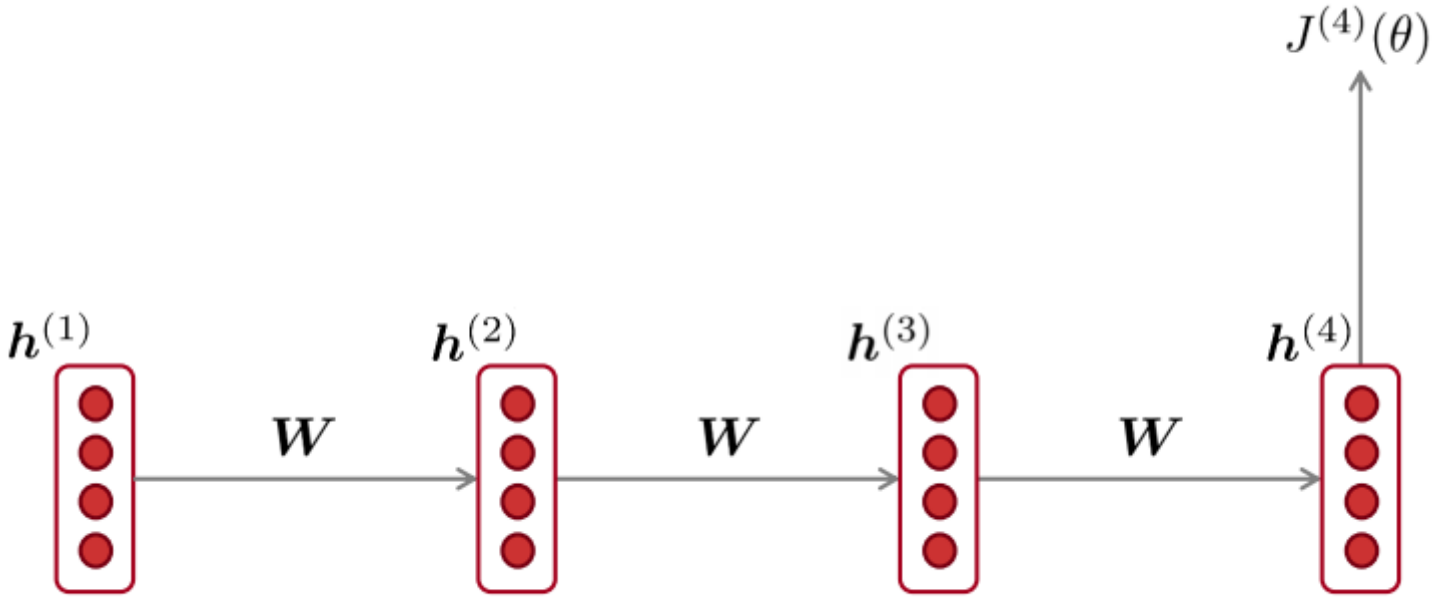
- Multiply the same matrix at each time step during forward prop



RNN 큰 문제 2가지
- 이거로 lstm 나왔어
- 그래디언트 배니싱
- 익스플로
- 그래서 엘이스티엠이나 지알유 사용
- 왜 생기는지는 하이레벨적으로 아는게 좋아요.

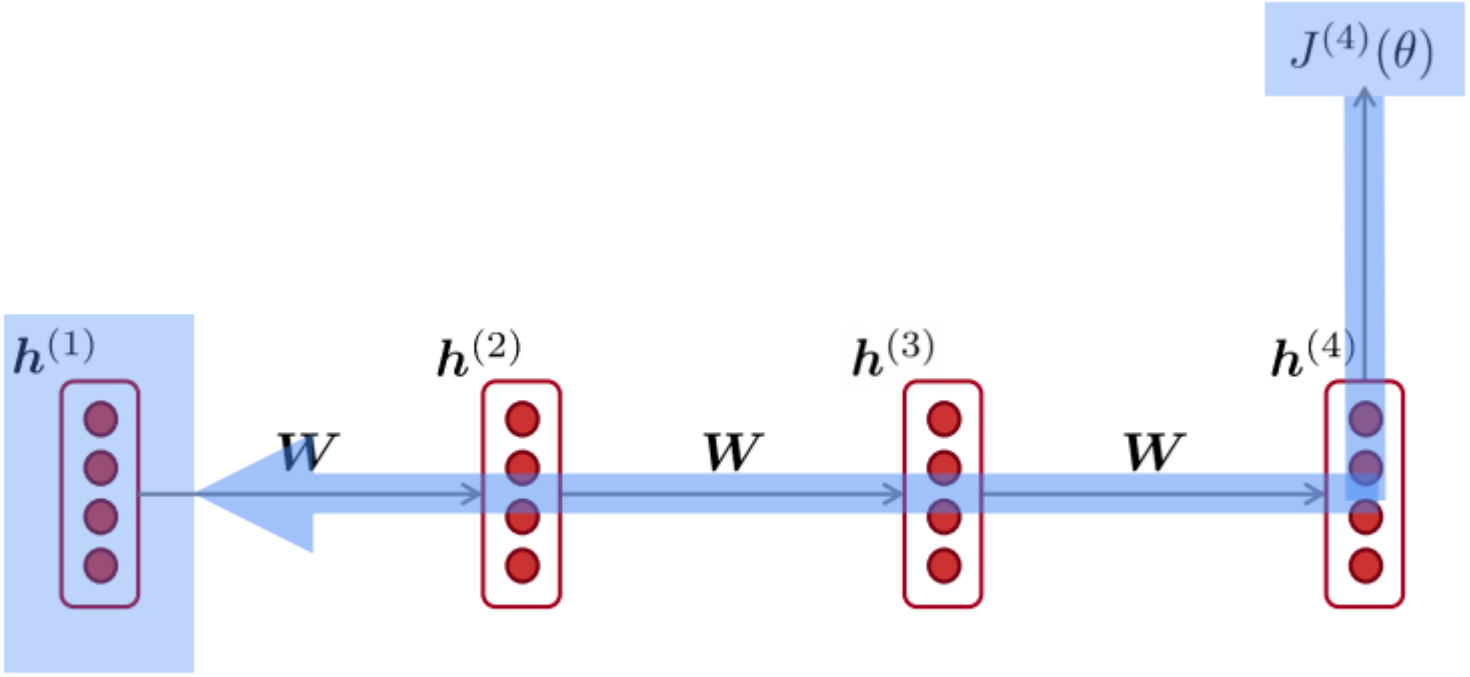
Vanishing Gradient Problem

Gradient Vanishing Problem



Vanishing Gradient Problem

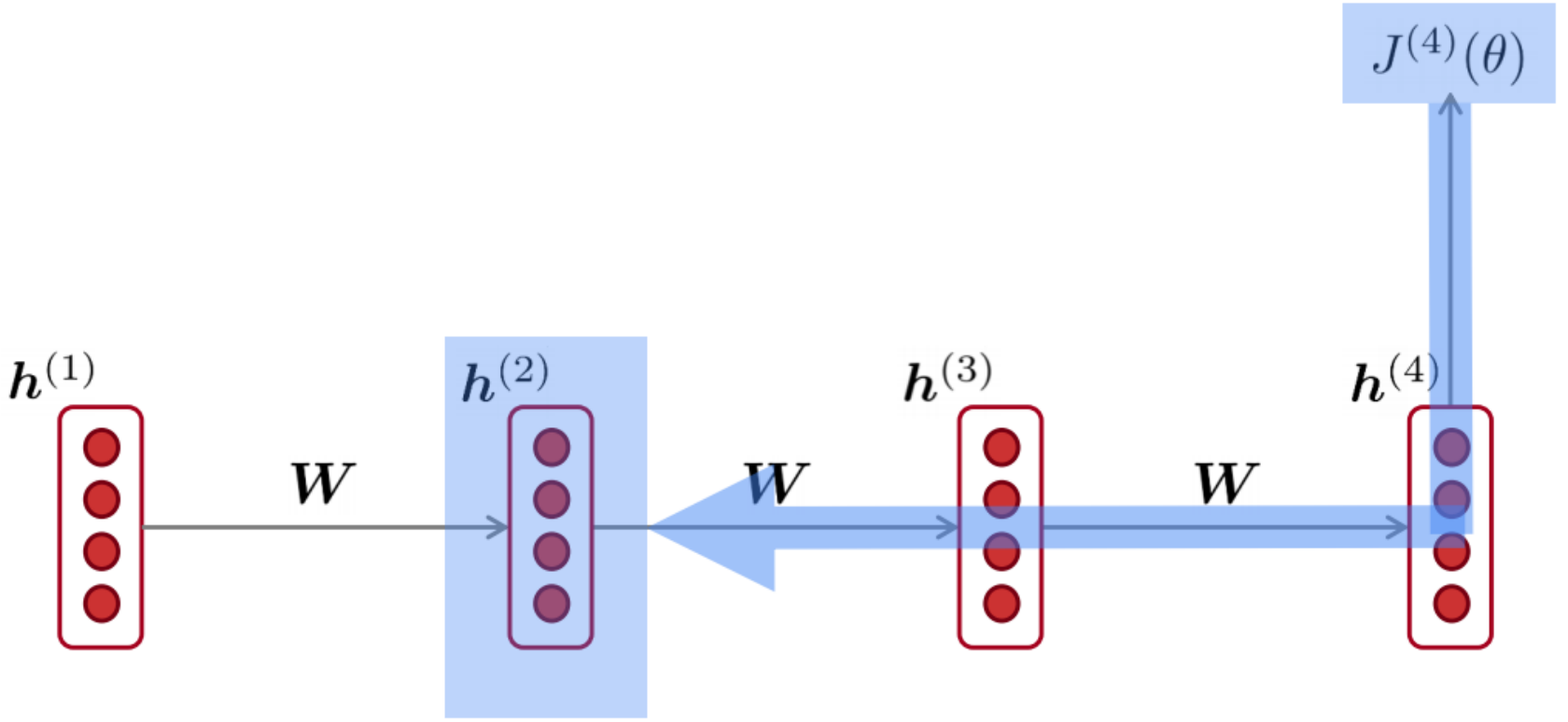
Gradient Vanishing Problem



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \ ?$$

Vanishing Gradient Problem

Gradient Vanishing Problem

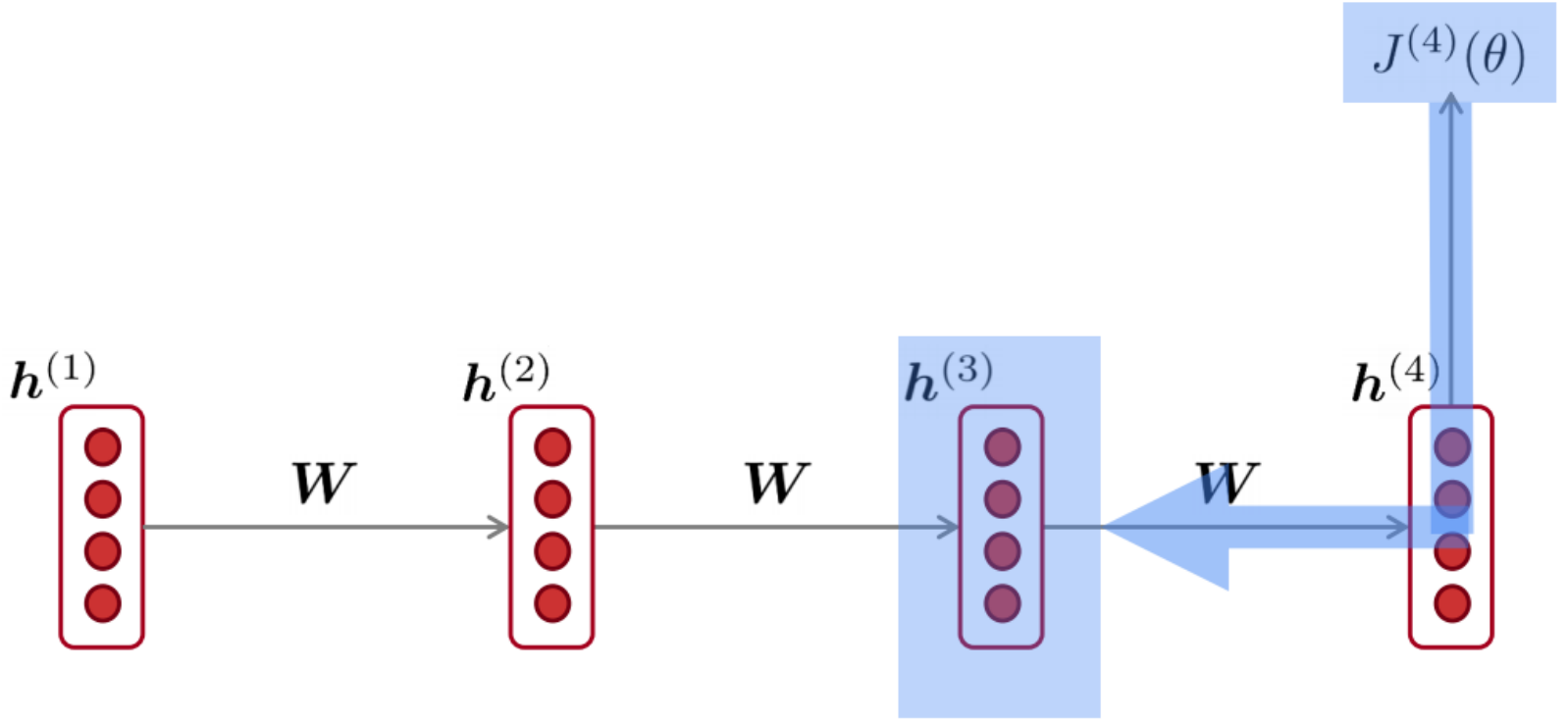


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

Vanishing Gradient Problem

Gradient Vanishing Problem

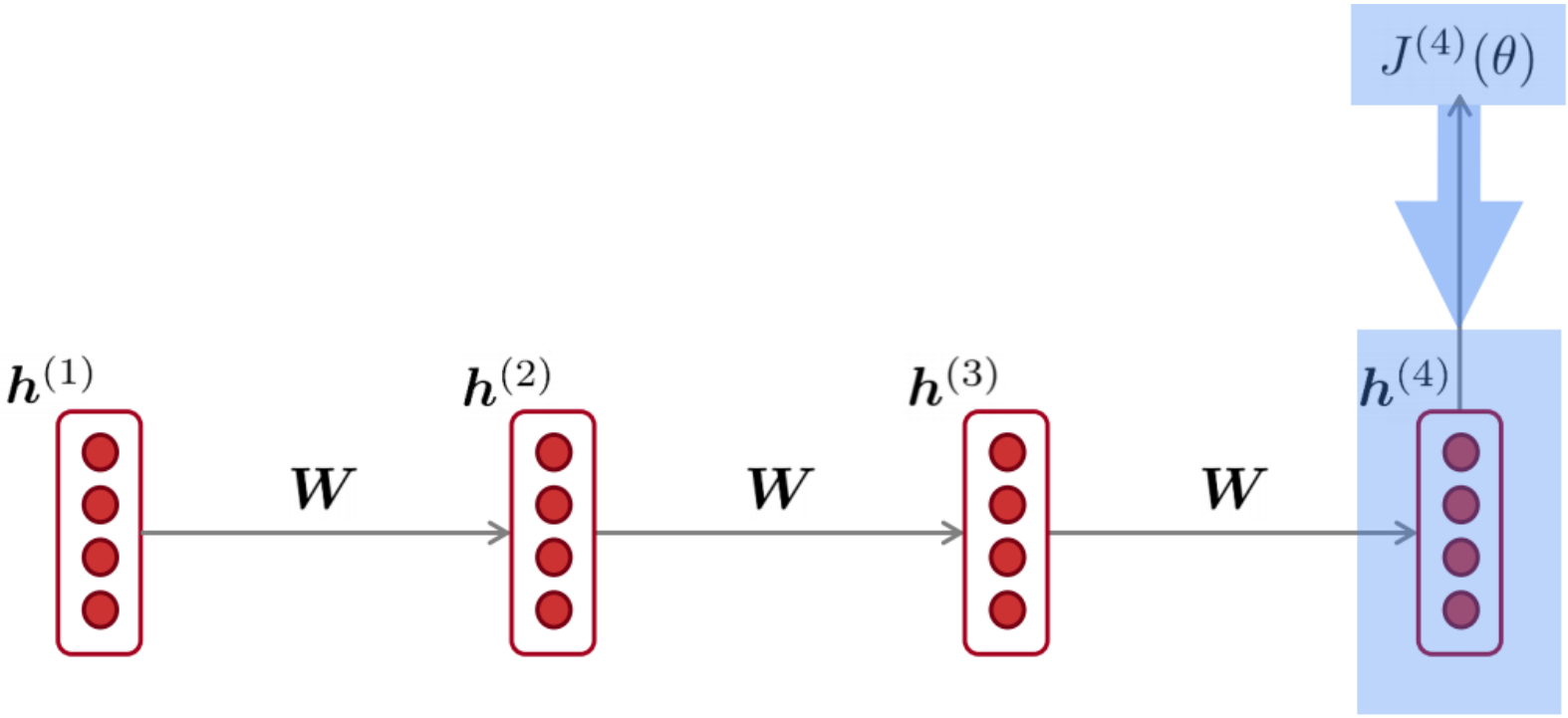


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

Vanishing Gradient Problem

Gradient Vanishing Problem

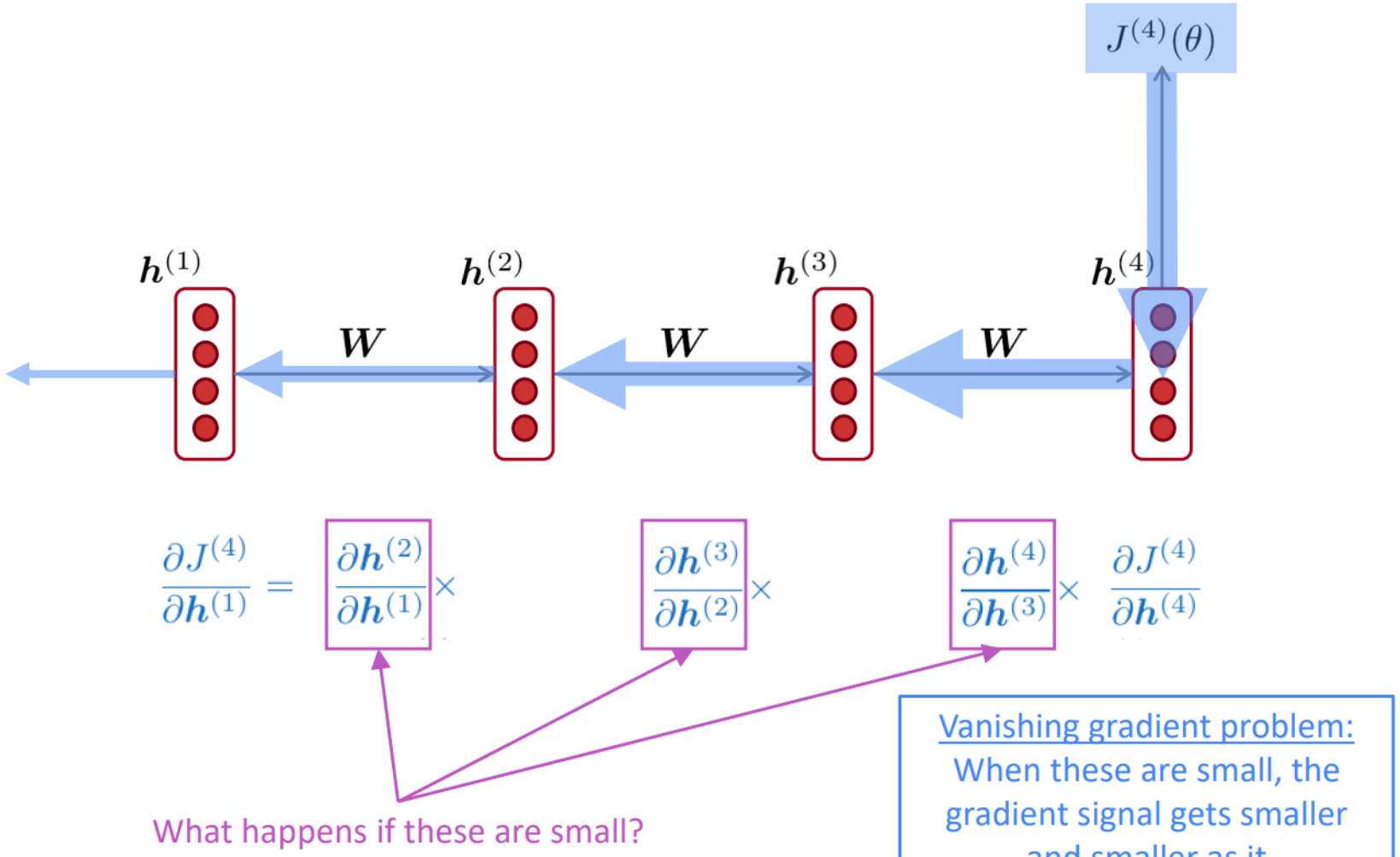


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

chain rule!

Vanishing Gradient Problem

Gradient Vanishing Problem



Vanishing Gradient Problem

Gradient Vanishing Problem

- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$

- Therefore: $$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h$$ (chain rule)

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step *i*, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step *j*.

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$ (chain rule)

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{(i-j)} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)$$ (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

Vanishing Gradient Problem

Gradient Vanishing Problem

- Consider matrix L2 norms:

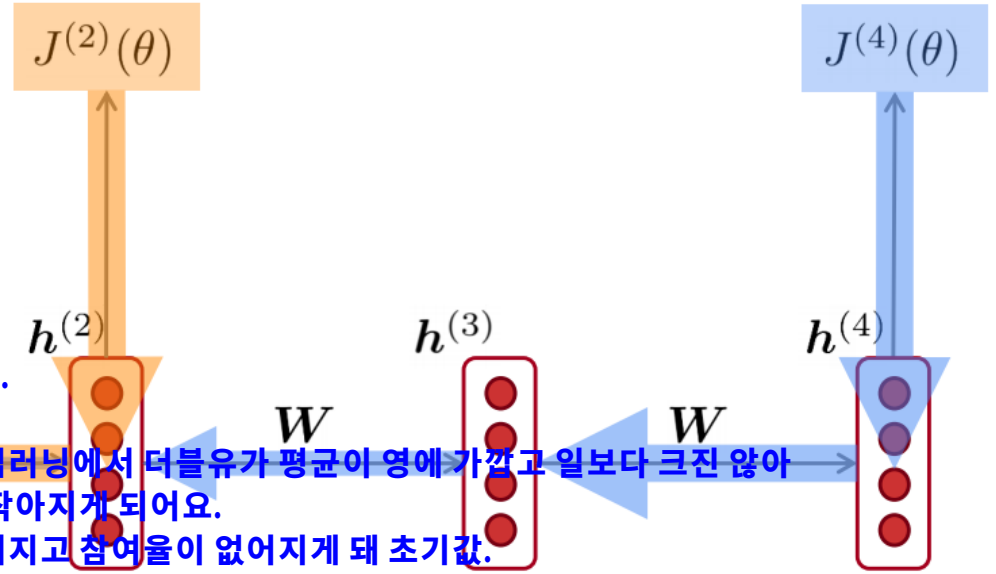
$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \right\| \|\boldsymbol{W}_h\|^{(i-j)} \prod_{j<t\leq i} \left\| \text{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \right\|$$

- Pascanu et al showed that that if the largest eigenvalue of W_h is less than 1, then the gradient $\left\| \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} \right\|$ will shrink exponentially
  - Here the bound is 1 because we have sigmoid nonlinearity

- There's a similar proof relating a largest eigenvalue >1 to exploding gradients

Vanishing Gradient Problem

Gradient Vanishing Problem



RNN 큰 문제 2가지
- 이거로 lstm 나왔어
- 그래디언트 배니싱
- 익스플로
- 그래서 엘이스티엠이나 지알유 사용
- 왜 생기는지는 하이레벨적으로 아는게 좋아요.

체인룰 생각- 배니싱 예시
- 탄에이치에서 웨이트 부분이고 머신러닝 및 딥러닝에서 더블유가 평균이 영에 가깝고 일보다 크진 않아
- 그러면 기본적으로 곱해질수록 저 값은 계속 작아지게 되어요.
- 그래서 알엔엔에서 계속 곱해져서 시퀀스 길어지고 참여율이 없어지게 돼 초기값.
- 그런데 이게 중요해요 예를 들어서 영어에서 수일치 문제 풀면
- 처음에 매니 피플 오브 뭐뭐하고 주격관계대명사 나오고 그러다 주어가 나오면
- 매니였으니까 이제 나올게 정해져있는데 잘못 생성했어. 그건 앞에것 잘 보면 알았을 문제인데
- 이거 수식어 읽다가 초기에 값을 까먹는 식이야.
- 그럼 앞에것 수정하고 싶어도 영향력이 미치지 않으니 바꿀 수가 없는 문제가 생기지

익스플로딩 예시
- 만약에 더블ㅇ 엥치 ㅣ 가 작지만 오히려 계속곱하면 운이 좋게 값이 증가하게 되는거야
- 일보다 크면 오히려 값이 커지겠죠. ㄱ

Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

Vanishing Gradient Problem

Gradient Vanishing Problem

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

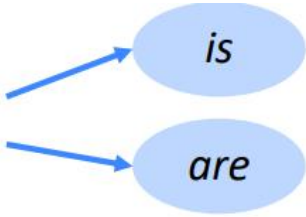
- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

- But if gradient is small, the model can't learn this dependency
  - So the model is unable to predict similar long-distance dependencies at test time

Vanishing Gradient Problem

Gradient Vanishing Problem

- **LM task:** *The writer of the books ___*



- **Correct answer**: *The writer of the books <u>is</u> planning a sequel*
- **Syntactic recency:** *The <u>writer</u> of the books <u>is</u>* (correct)
- **Sequential recency:** *The writer of the <u>books</u> <u>are</u>* (incorrect)
- Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we'd like [Linzen et al 2016]

Vanishing Gradient Problem

Gradient Exploding Problem

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Vanishing Gradient Problem

Gradient Clpping

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

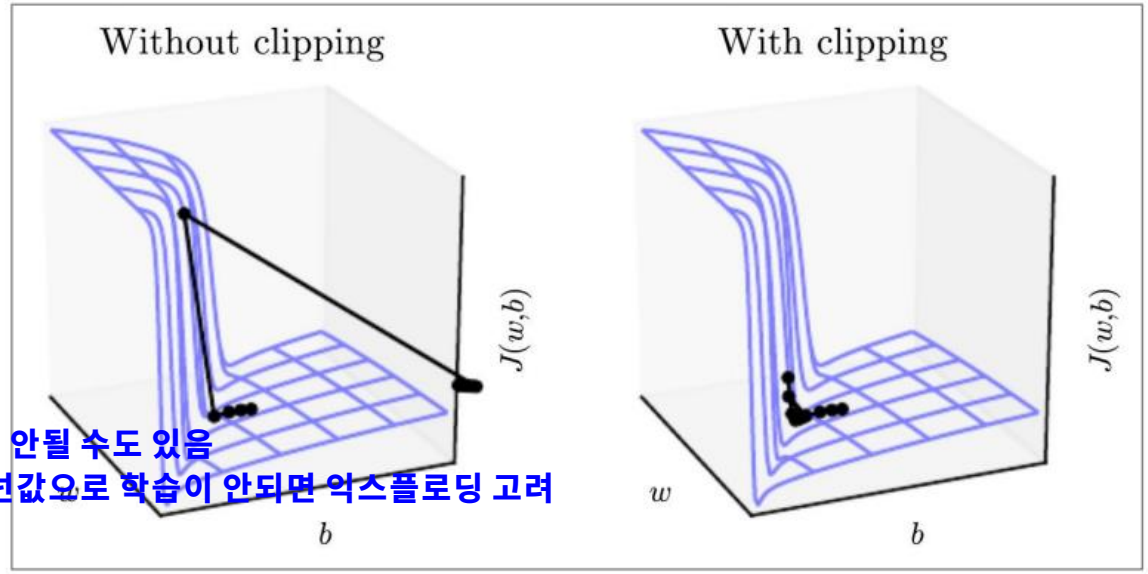
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Intuition: take a step in the same direction, but a smaller step

Vanishing Gradient Problem

Gradient Clipping



그래디언트 크면 안돼
- 너무 깊게 믿으면 발등 찍혀
- 그래서 익스플로딩으로써 업데이트가 안될 수도 있음
- 워스트하면 익스플로가 인피니티 및 넌값으로 학습이 안되면 익스플로딩 고려

그래디언트 익스플로딩 푸릭
- 그래디언트르 클래핑
- 첫번째의 디맨션이 10과 20대의 값이나오면 이 최대를 1로 픽싱하여 클리핑
- 그러면 갑자기 확 뛰어버리는 문제를 해결할 수 있음.

- This shows the loss surface of a simple RNN (hidden state is a scalar not a vector)
- The "cliff" is dangerous because it has steep gradient
- On the left, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- On the right, gradient clipping reduces the size of those steps, so effect is less drastic

Vanishing Gradient Problem

How to fix it ?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*.

- In a vanilla RNN, the hidden state is constantly being rewritten

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$$

- How about a RNN with separate memory?

Vanishing Gradient Problem

Gradient 사라짐 문제가 왜 중요할까?

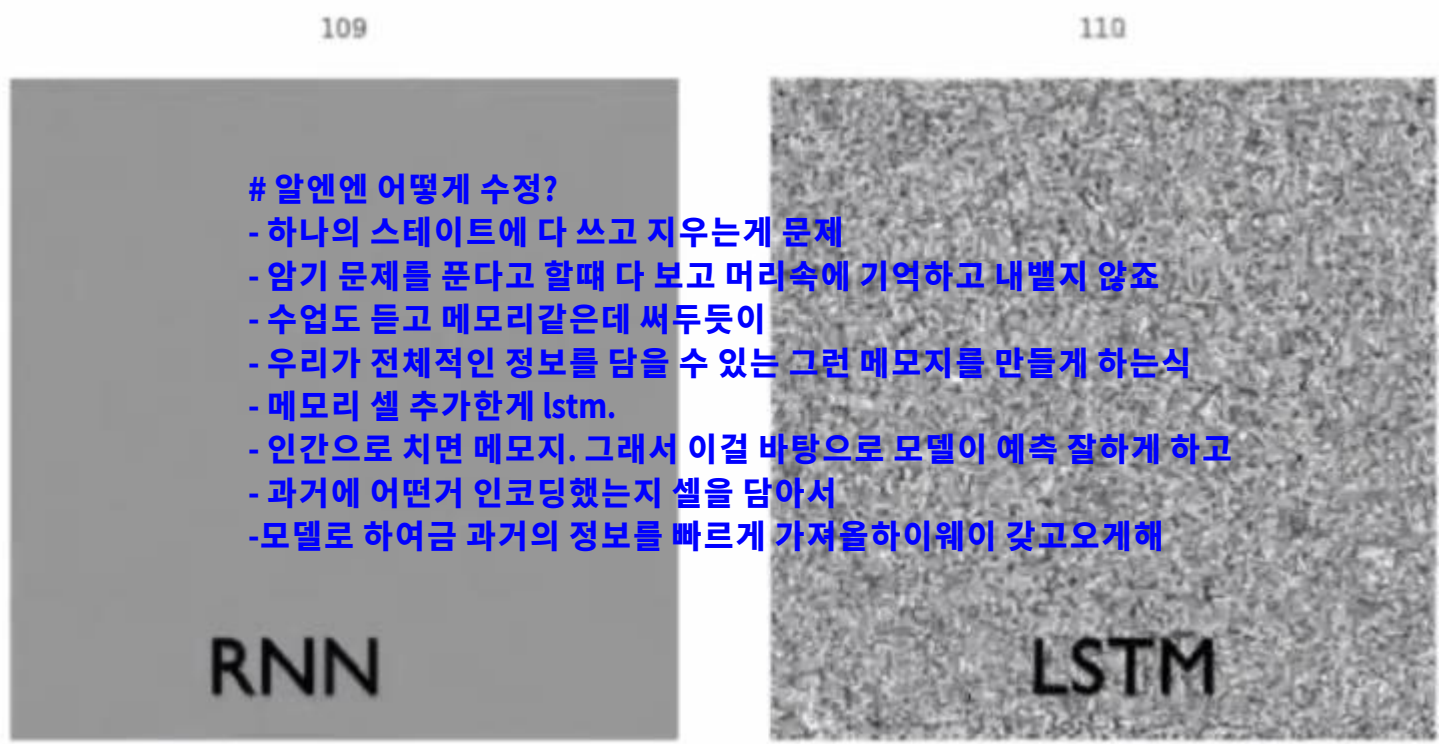
- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word

- Example:

  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

Vanishing Gradient Problem

Gradient 사라짐 문제가 왜 중요할까?



알엔엔 어떻게 수정?
- 하나의 스테이트에 다 쓰고 지우는게 문제
- 암기 문제를 푼다고 할때 다 보고 머리속에 기억하고 내뱉지 않죠
- 수업도 듣고 메모리같은데 써두듯이
- 우리가 전체적인 정보를 담을 수 있는 그런 메모지를 만들게 하는식
- 메모리 셀 추가한게 lstm.
- 인간으로 치면 메모지. 그래서 이걸 바탕으로 모델이 예측 잘하게 하고
- 과거에 어떤거 인코딩했는지 셀을 담아서
-모델로 하여금 과거의 정보를 빠르게 가져올하이웨이 갖고오게해

RNN vs LSTM: Vanishing Gradients - GIF on Imgur

엘에스티엠
- 세개의 게이트(인풋, 포겟, 게이트 게이트)와 탄에이치 하나로 구성이 되어있어요
- 하이레벨로 말하면 이전에 바날라 알엔엔은 모든 부분의 레지스터를 다 읽고 한번에 업뎃하는 식
- 그런데 이거를 그대로 유지해야할때는 어떻게 해야하나.더블류 따로 했어야해
- 모든걸 다 읽고 모든걸 다해(메모를 한번보고 다른 정보 들어오면 다 지우고 다시 쓰는 식)
- 그러면 이걸 잘하려면 원래거에 수정할 부분만 수정하면 되겠지요. 그게 쥐알유와 알에스티엠
- 히든에서 원하는 특정 디맨션만 뽑아와서 수정할 양이나 위치를 정합니다.-
- 그 중에서 덧붙일 부분에 추가적으로 써서 업데이트 하는식
- 그러면 아무것도 안바꾸거나 조금만 수정을 하고자 할때 이전 정보를 잘 유지할 수 있어요
- 메모지를 생각하면 좋아요. 메모지를 다 찢어서 다 적는거 아니면 추가된 정보 일부만 덧붙이는거
- 어떤게 정보 안 잃고 이어나갈 수 있는지. 일정만 수정하는게 더 좋아요
    - 인풋게이트:어디 읽을지
    - 포겟 : 어디 지울지
    -라이트 게이트: 어디를 쓸지
    - 그래서 모델로 하여금 정보를 기억해서 전달하게끔. 이게 엘이스티엠 및 쥐알유 하이레벨적

02

2. LSTM과 GRU

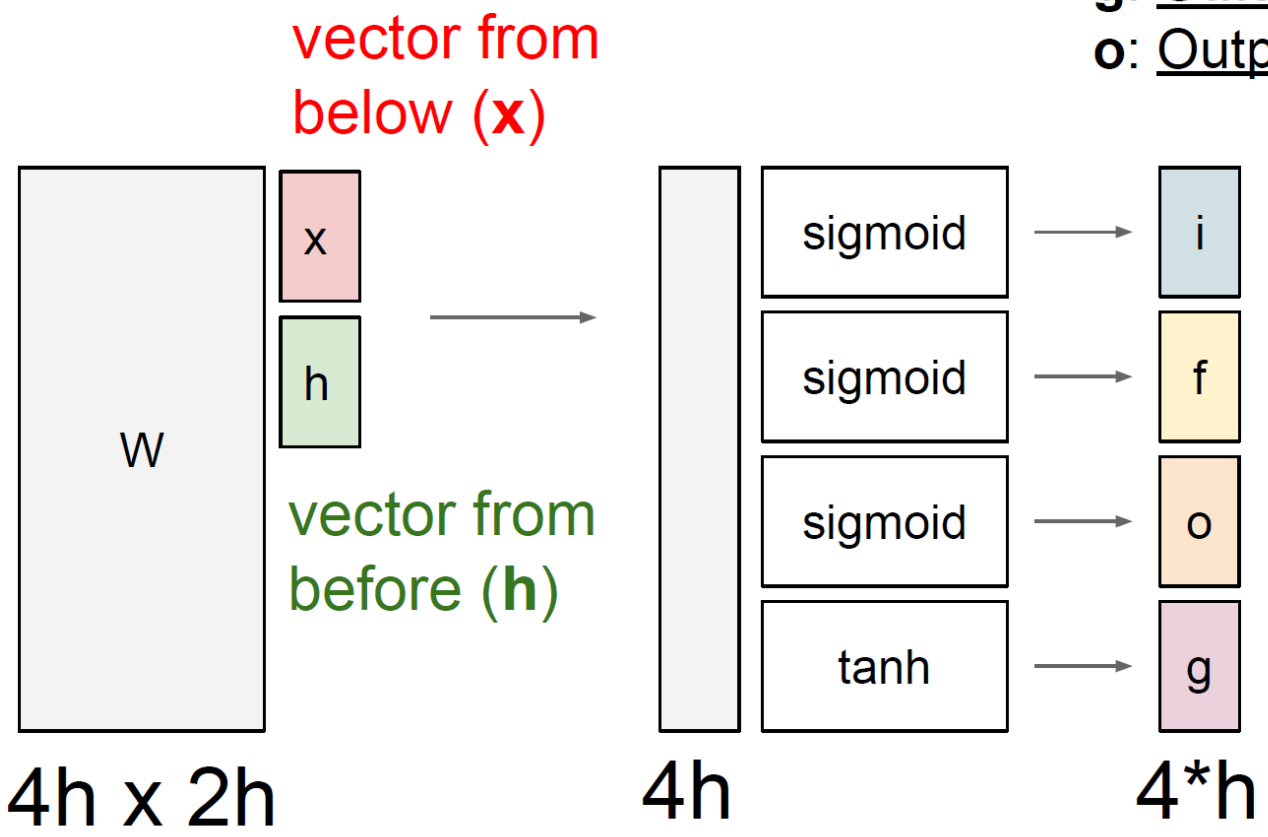
2-1. LSTM이란 무엇인가

2-2. GRU란 무엇인가

LSTM

Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.

- On step *t*, there is a hidden state $\boldsymbol{h}^{(t)}$ *and* a cell state $\boldsymbol{c}^{(t)}$
  - Both are vectors length *n*
  - The cell stores long-term information
  - The LSTM can erase, write and read information from the cell

- The selection of which information is erased/written/read is controlled by three corresponding gates
  - The gates are also vectors length *n*
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between.
  - The gates are dynamic: their value is computed based on the current context

LSTM

We have a sequence of inputs $\boldsymbol{x}^{(t)}$, and we will compute a sequence of hidden states $\boldsymbol{h}^{(t)}$ and cell states $\boldsymbol{c}^{(t)}$. On timestep t:

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

Hidden state: read ("output") some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

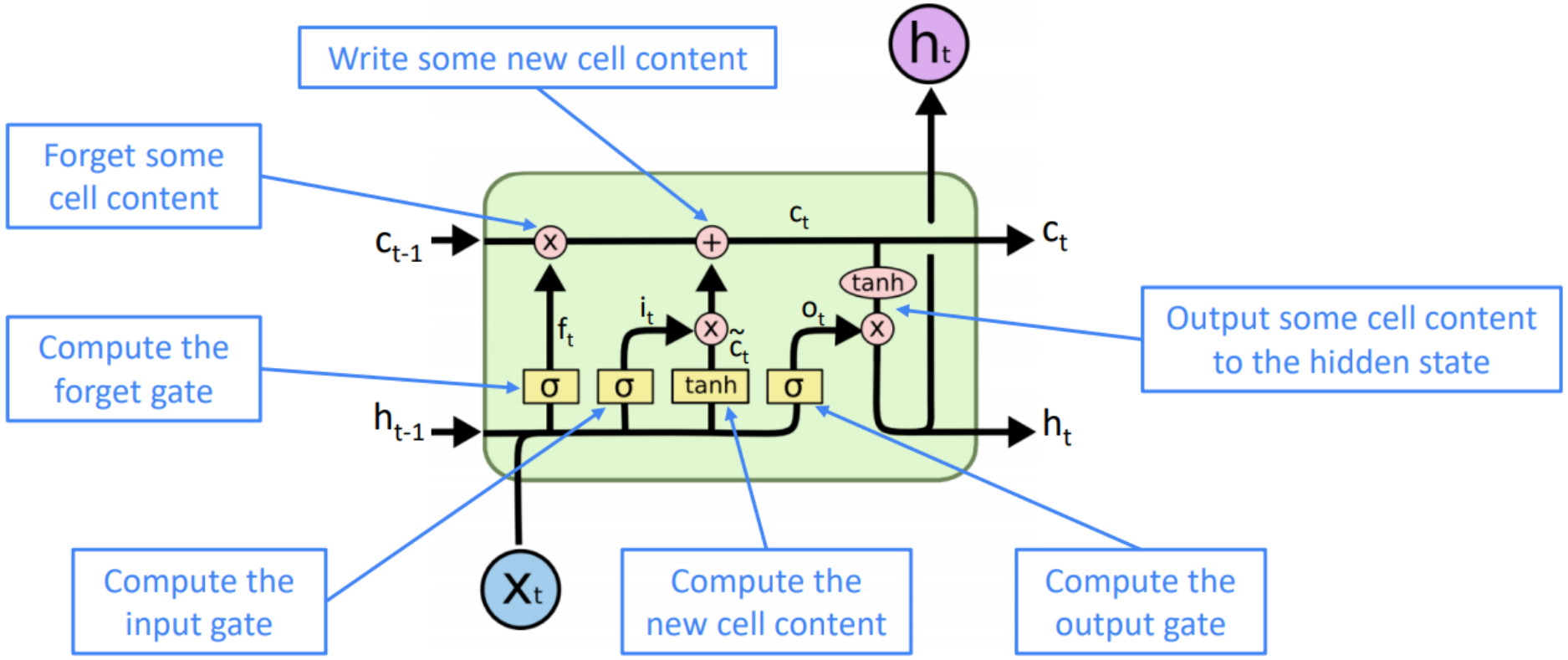
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise product

LSTM

You can think of the LSTM equations visually like this:



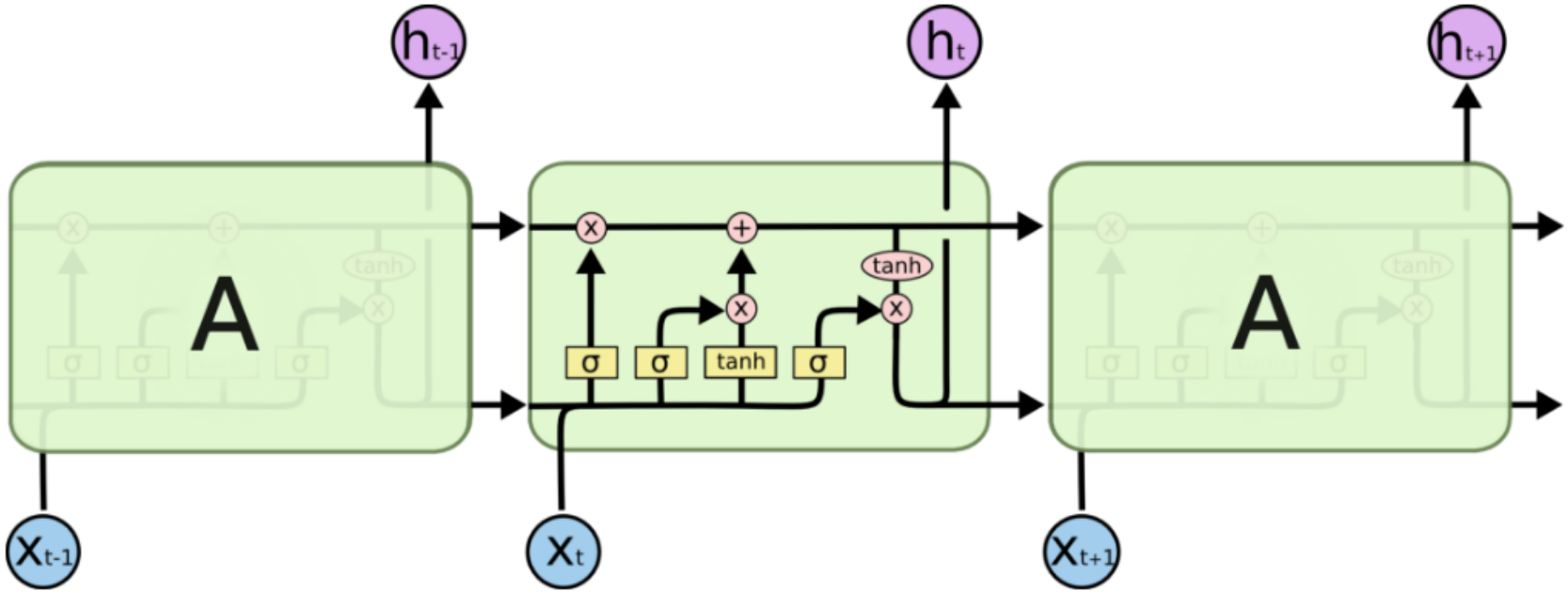
LSTM

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps
  - e.g. if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
  - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

- LSTM doesn't *guarantee* that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

LSTM

LSTM(Long Short-Term Memory)이란 무엇인가

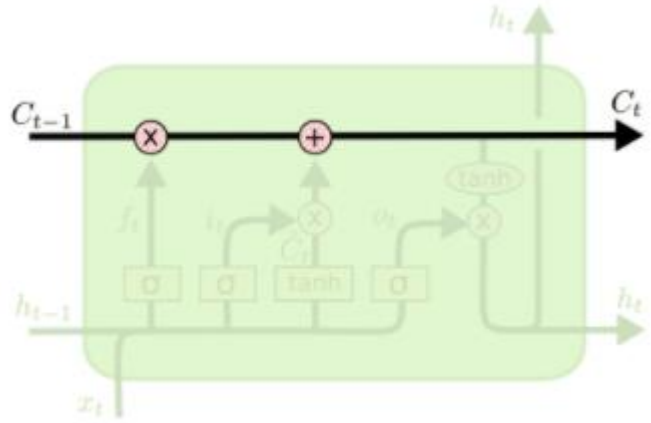


The repeating module in an LSTM contains four interacting layers.

[Understanding LSTM Networks -- colah's blog]

LSTM

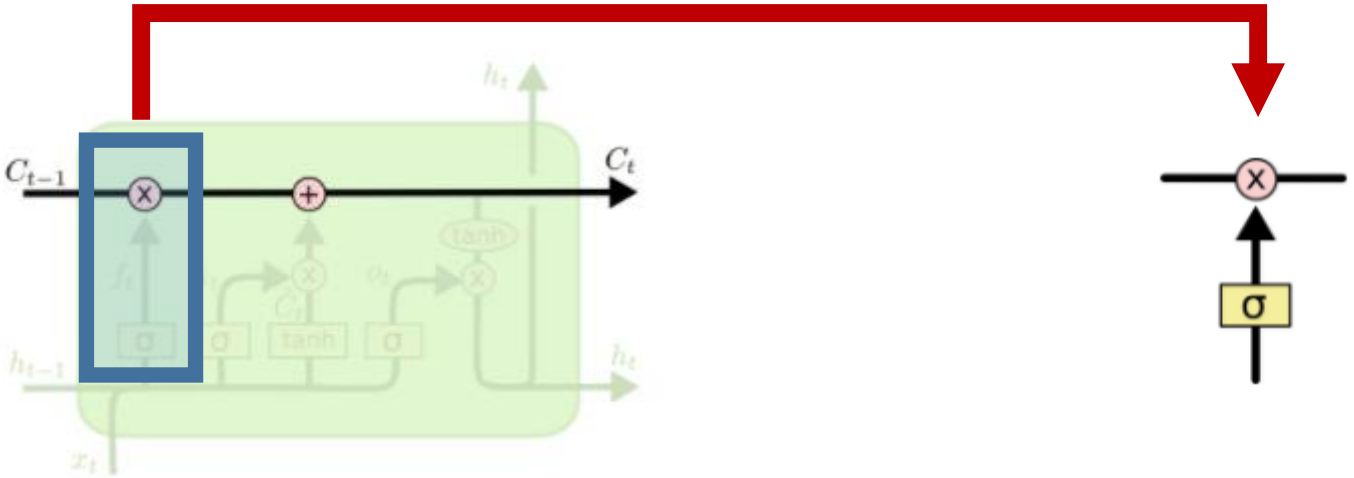
Core Idea: cell state 정보가 아무 변화없이 쭉 흐를 수 있는 구조 -> Long-term dependency 해결



[Understanding LSTM Networks -- colah's blog]

LSTM

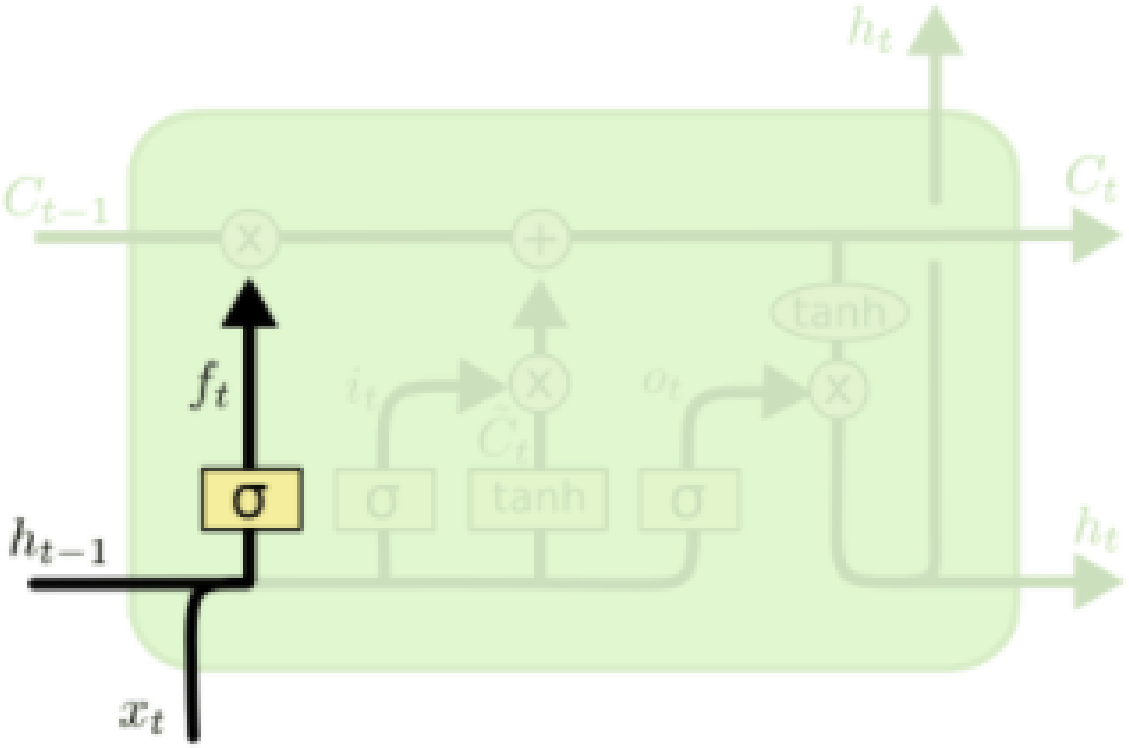
이전에서 넘어온 cell state 정보를 얼마나 흘려보낼지에 대한 수문 (gate)이 존재



Understanding LSTM Networks -- colah's blog

LSTM

Forget gate

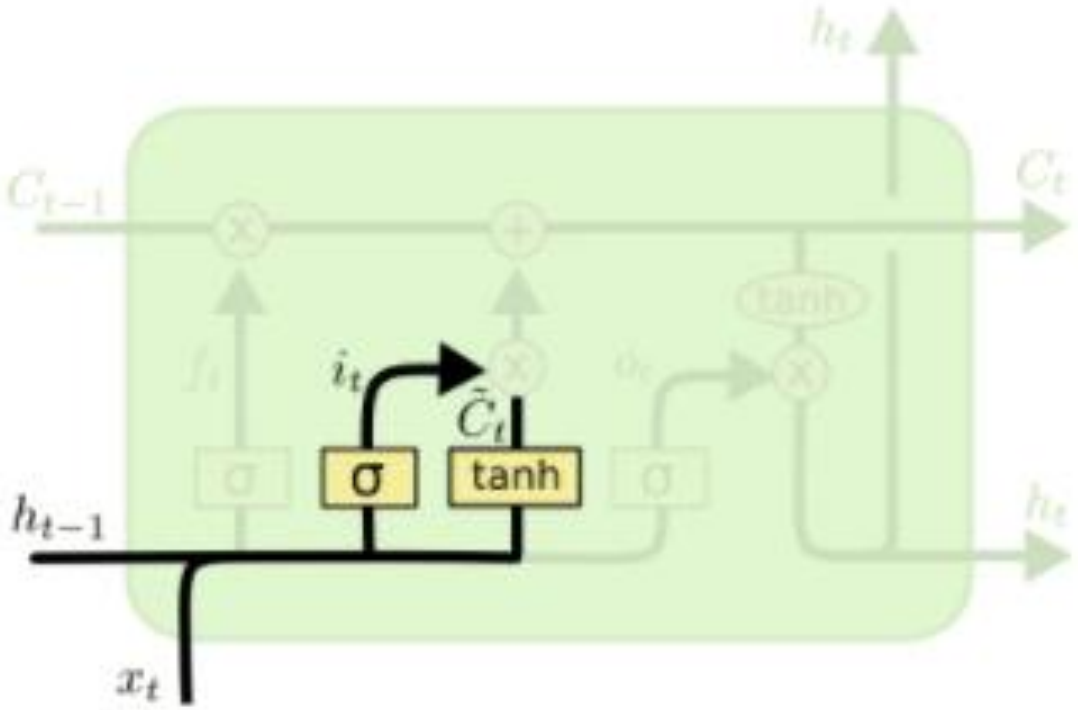


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

[Understanding LSTM Networks -- colah's blog]

LSTM

Cell state에 추가할 정보를 생성하고 여기에, input gate를 통해 일부를 버림



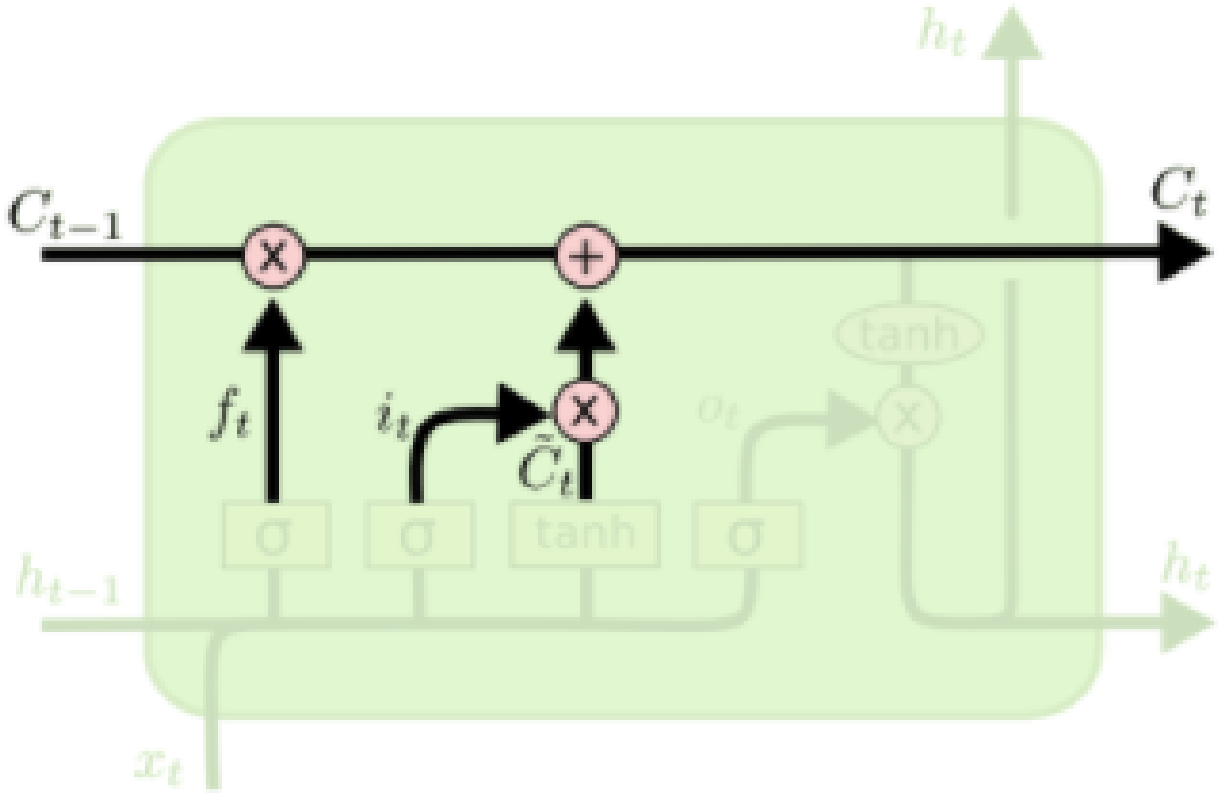
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

[Understanding LSTM Networks -- colah's blog]

LSTM

버릴 것은 버린 (forget gate) 과거에서 넘어온 cell state에 현재 정보를 더해서 현재의 cell state를 생성

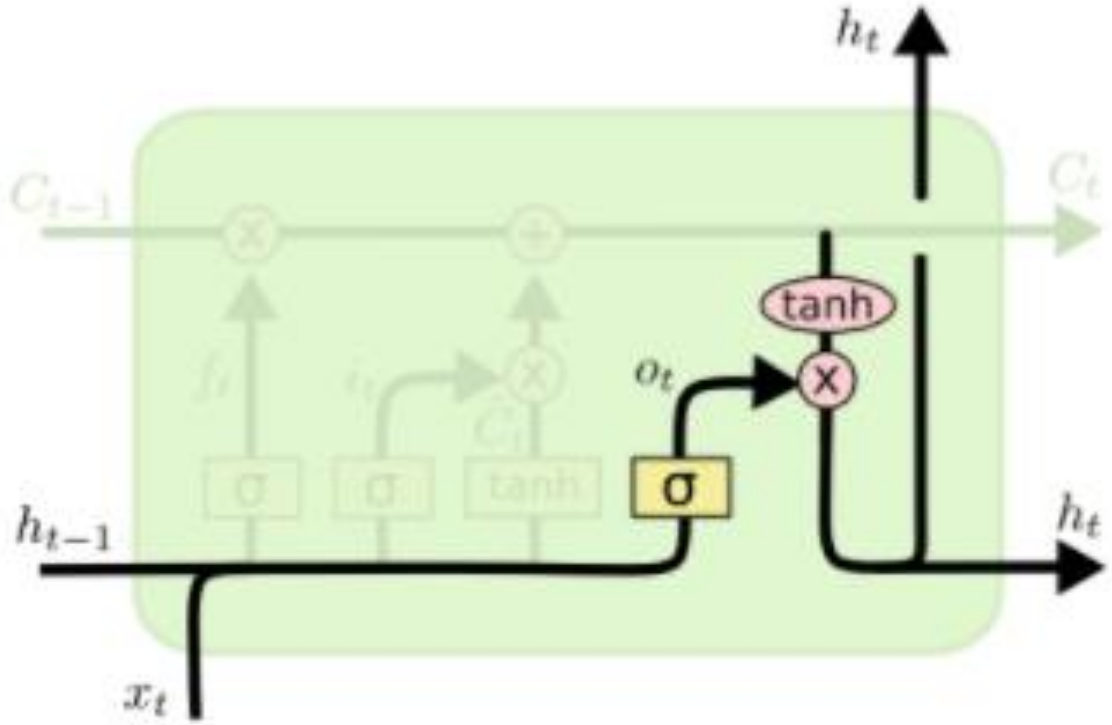


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Understanding LSTM Networks -- colah's blog

LSTM

현재의 cell state를 tanh를 통과하고 여기에 output gate를 통과시켜 현재의 hidden state를 생성.
그 이후, 이 hidden state는 다음 time step으로 넘겨주고, 필요하면 output 쪽이나 next layer로 넘겨줌.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

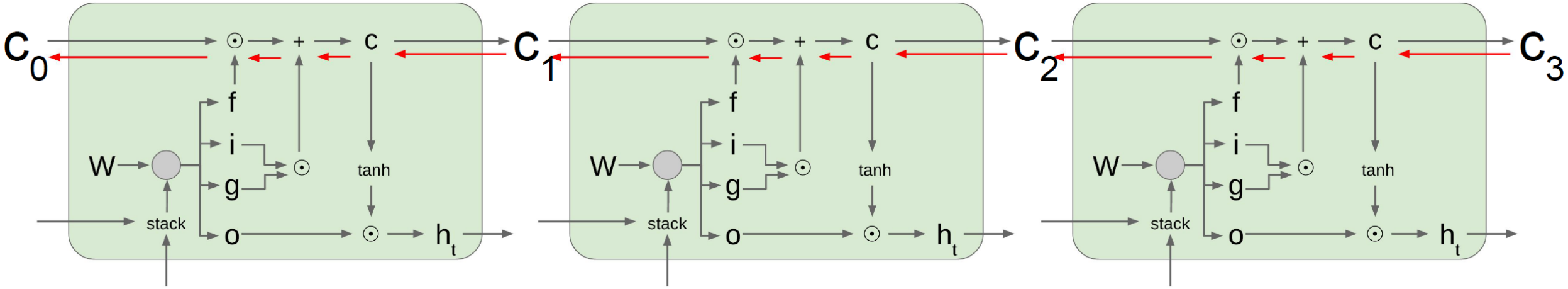
$$h_t = o_t * \tanh\left(C_t\right)$$

Understanding LSTM Networks -- colah's blog

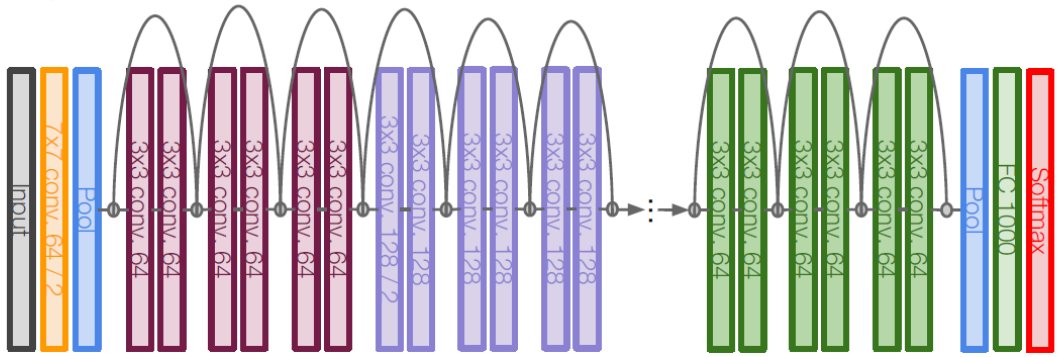
LSTM: Gradient Flow

레스넷이랑 비슷하대요. 레스넷이 뭐지 @_@

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

$$g = T(x, W_T)$$

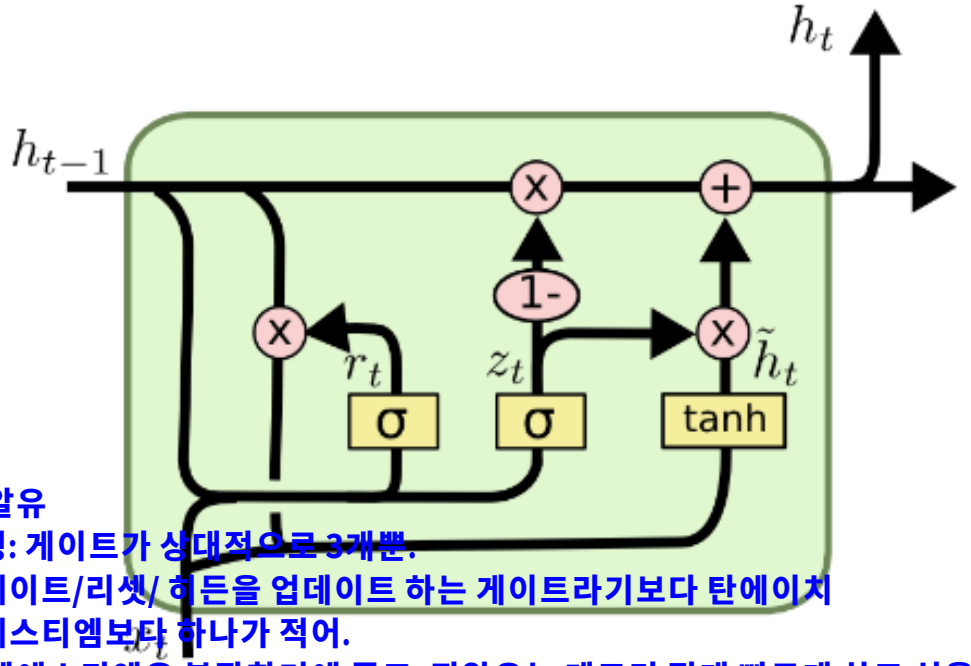
$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

GRU

GRU(Gated Recurrent Unit)란 무엇인가?

 # 프로젝트
- 첫번째로 쥐알유 도입
- 그다으메 엘에스티엠 도입하는게 좋은 선택



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

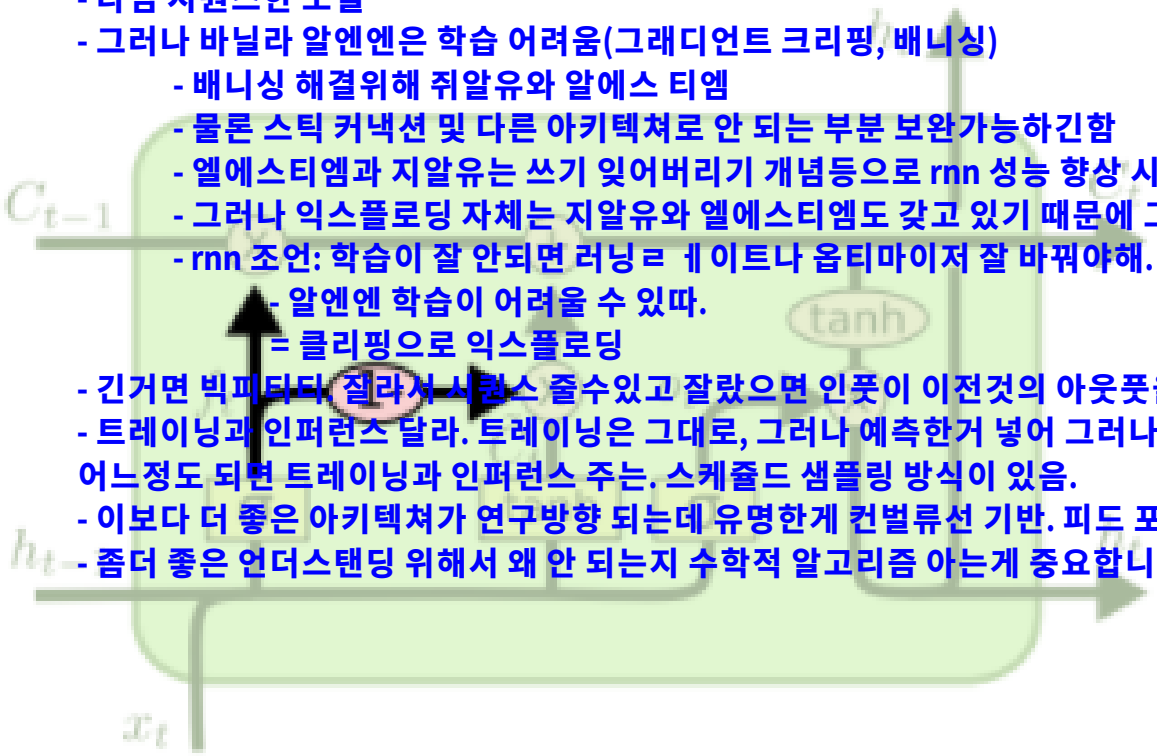
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

 # 쥐알유
- 특징: 게이트가 상대적으로 3개뿐.
- 업데이트/리셋/ 히든을 업데이트 하는 게이트라기보다 탄에이치
- 엘에스티엠보다 하나가 적어.
- 즉 엘에스티엠은 복잡한거에 좋고, 쥐알유는 메모리 적게 빠르게 하고 싶을때 좋은 선택.
- 왜냐하면 쥐알유가 파라미터가 적어서. 그래서 학습도 빠름ㄴ

Understanding LSTM Networks -- colah's blog

GRU

GRU(Gated Recurrent Unit)란 무엇인가?



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

요약
- 알엔엔ㅇ느 플랙서블
- 타임 시퀀스한 모델
- 그러나 바닐라 알엔엔은 학습 어려움(그래디언트 크리핑, 배니싱)
    - 배니싱 해결위해 쥐알유와 알에스 티엠
    - 물론 스틱 커낵션 및 다른 아키텍쳐로 안 되는 부분 보완가능하긴함
    - 엘에스티엠과 지알유는 쓰기 잊어버리기 개념등으로 rnn 성능 향상 시킴.
    - 그러나 익스플로딩 자체는 지알유와 엘에스티엠도 갖고 있기 때문에 그래딩너트 클래핑은 어쩔수업슴
    - rnn 조언: 학습이 잘 안되면 러닝ㄹ ㅔ이트나 옵티마이저 잘 바꿔야해. 그래딩너트 익스플로딩이상할수
        - 알엔엔 학습이 어려울 수 있따.
        - = 클리핑으로 익스플로딩
- 긴거면 빅피터다, 잘라서 시퀀스 줄수있고 잘랐으면 인풋이 이전것의 아웃풋을 넣어서 하나로 학습하듯
- 트레이닝과 인퍼런스 달라. 트레이닝은 그대로, 그러나 예측한거 넣어 그러나 트레이닝으 초반에 아웃풋넣으면 이상할 수 있어. 그래서 처음엔 그라운투스 줘서 안정 주고 어느정도 되면 트레이닝과 인퍼런스 주는. 스케쥴드 샘플링 방식이 있음.
- 이보다 더 좋은 아키텍쳐가 연구방향 되는데 유명한게 컨벌류션 기반. 피드 포워드(언탠션 기반) 기반 모델들
- 좀더 좋은 언더스탠딩 위해서 왜 안 되는지 수학적 알고리즘 아는게 중요합니다.

GRU

GRU

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep *t* we have input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$ (no cell state).

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

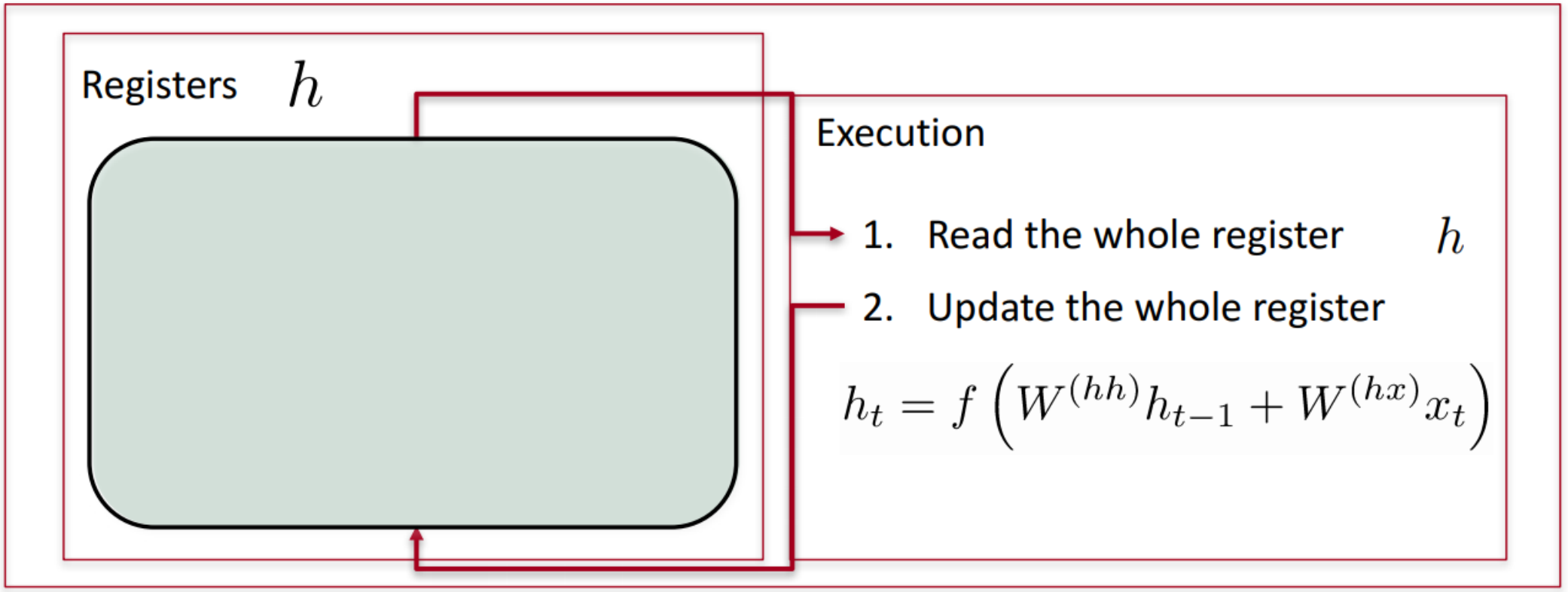
$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

GRU

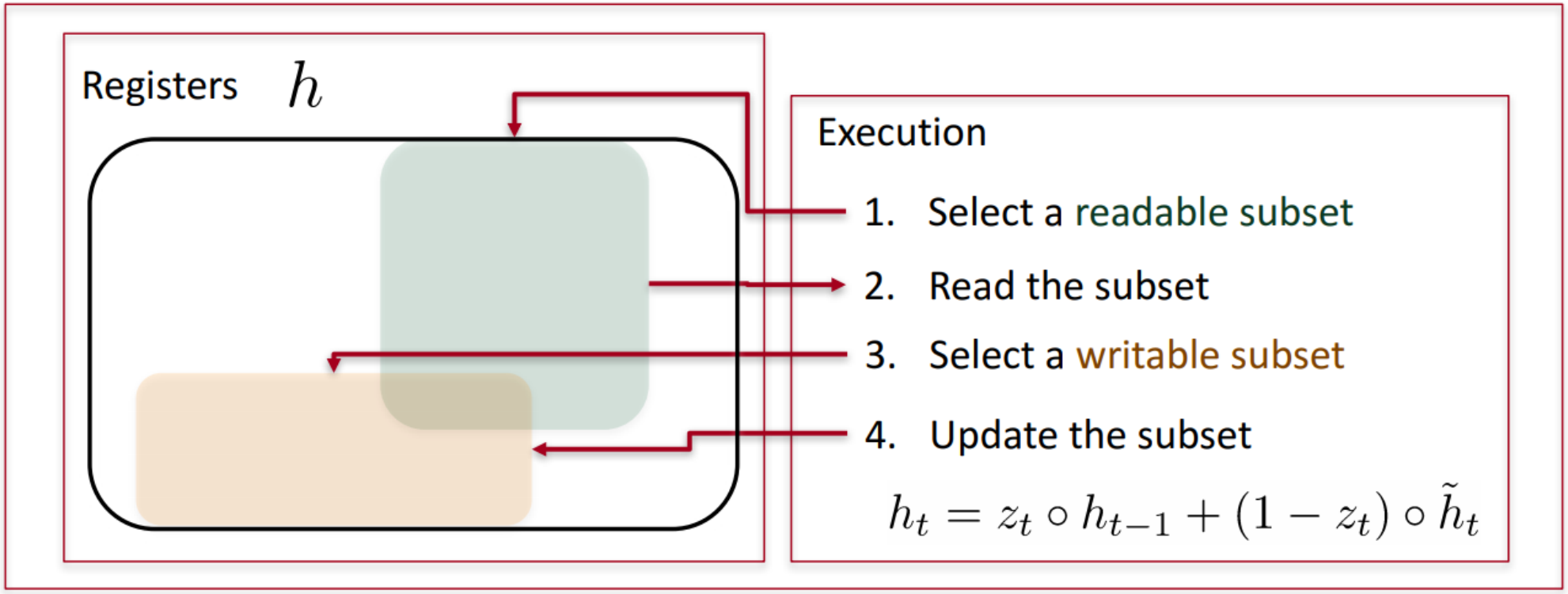
GRU comparison to Standard tanh-RNN

Vanilla RNN ...



GRU

GRU comparison to Standard tanh-RNN



Gated recurrent units are much more versatile and adaptive in which elements of the hidden vector h they update!

GRU

GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- The biggest difference is that GRU is quicker to compute and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

GRU

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

For example:

- Residual connections aka "ResNet"
- Also known as skip-connections
- The identity connection preserves information by default
- This makes deep networks much easier to train

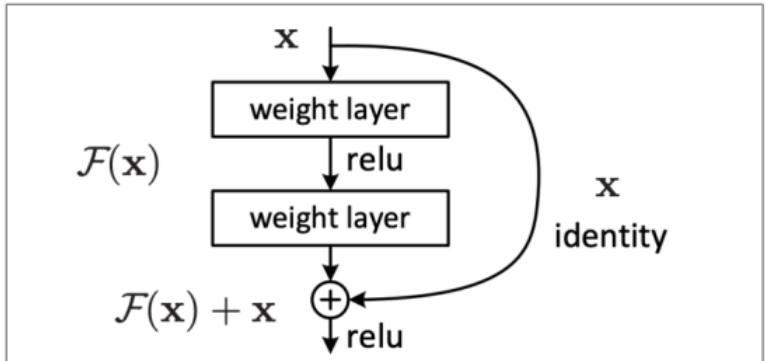


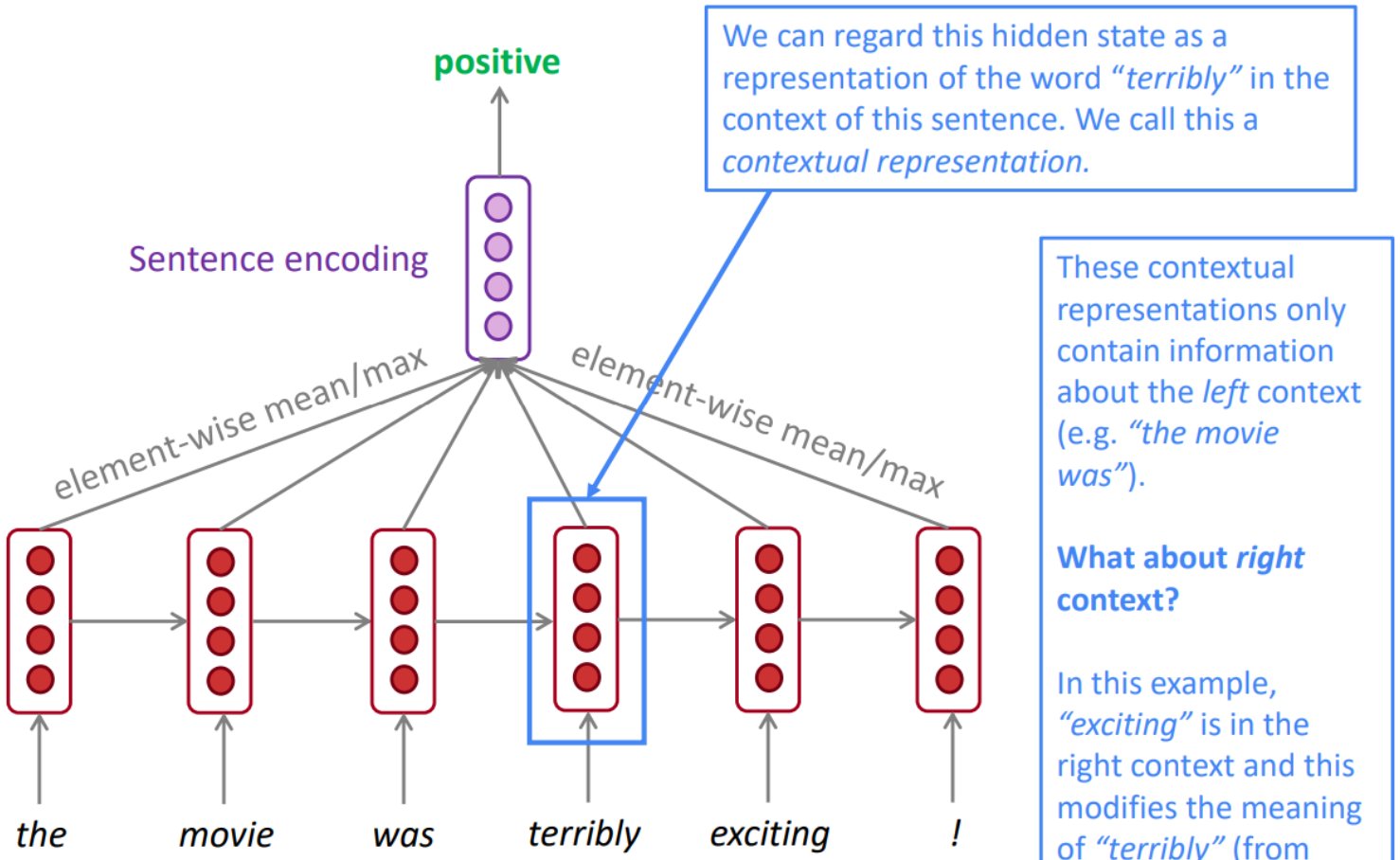
Figure 2. Residual learning: a building block.

RNN/LSTM Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.

GRU

Task: Sentiment Classification



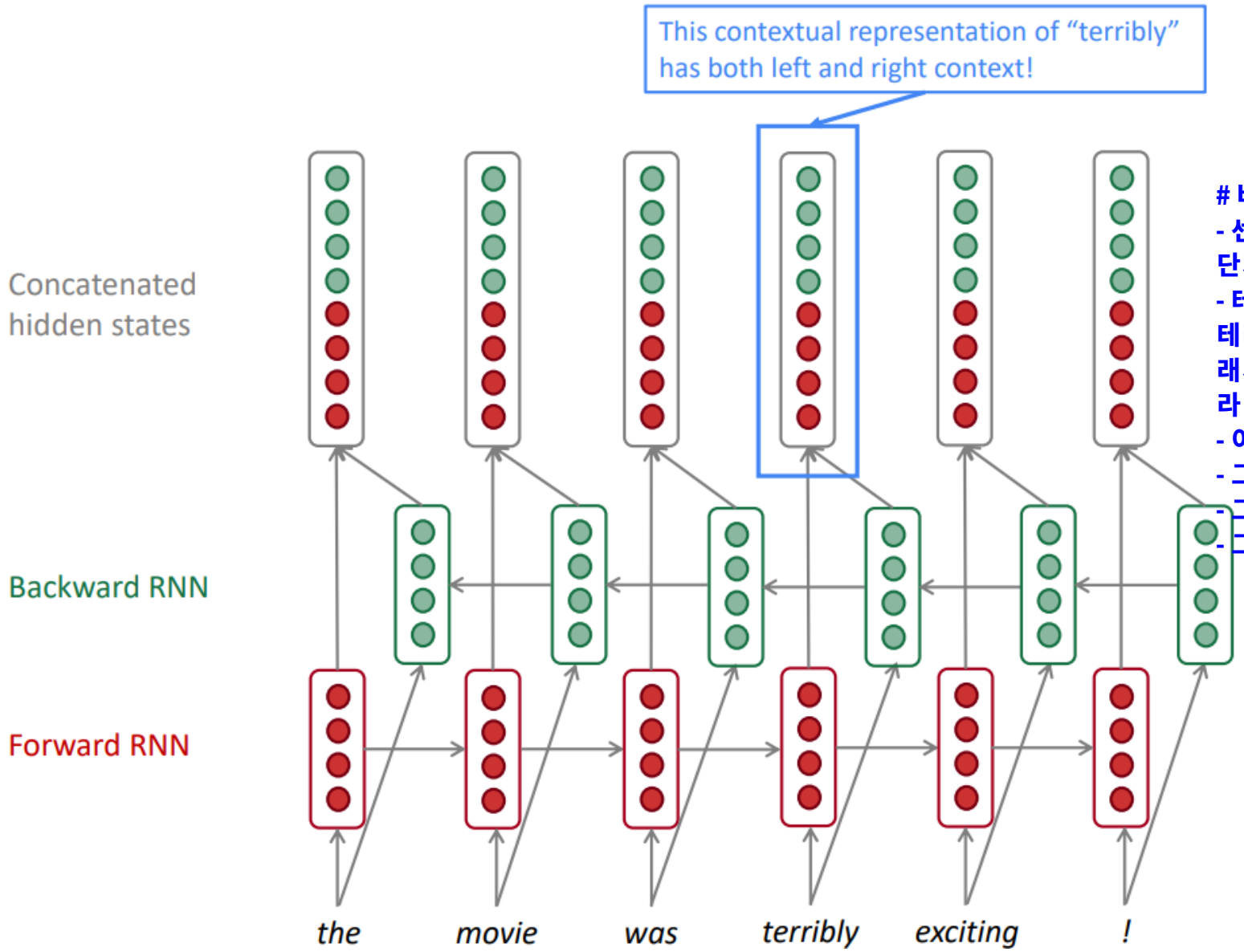
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Bidirectional RNN



바이디렉셔널 ㅇ알엔엔
- 센티맨트 클래시픽하면 알엔엔 단점은 하나의 방향 정보만 있단거
- 테러블에 대한건 익사이팅을 보지 못해. 만약에 더 무비 워스 테러블리 익사이팅에서 테러블리는 더 무비 워서 까지만 봐. 그래서 얘는 부정 네거티브로 읽을 수 있으나, 사실은 익사이팅이라 긍정일 수있어
- 이 정보가 테러블리에게 영향 못줘
- 그래서 알엔엔은 한쪽만 봐서 다른 방향을 못보는 단점이 생겨
- 그리고 익사이팅이라는게 무비한테 영향 못줘
- 그래서 양방향으로 늘리는거야.

Bidirectional RNN

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{FW}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

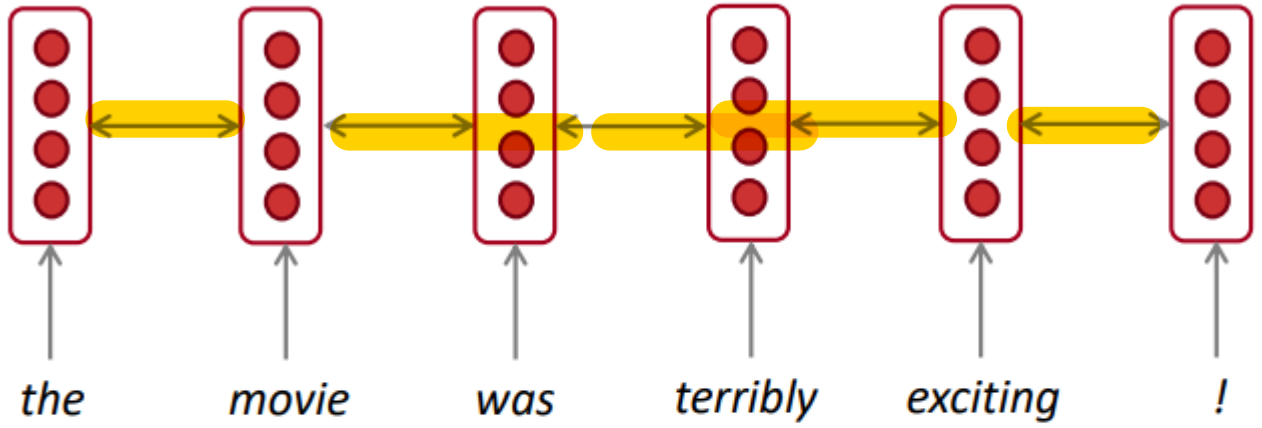
Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boldsymbol{h}^{(t)} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

Bidirectional RNN



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states.

Bidirectional RNN

- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence.
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.

- If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).

- 그래서 알엔엔은 한쪽만 봐서 다른 방향을 못보는 단점이 생겨
- 그리고 익사이팅이라는게 무비한테 영향 못줘
- 그래서 양방향으로 늘리는거야.
- 이런 상호보완을 위해 정방/역방향을 concat해요
- 생성테스크가 아니면 무조건 바이디렉션이 조아요***

= 디스트리부트 레프리테이션
예) 포유류를 구하고자 할때, 포유류라는 벡터가 있고 이게 여러 곳에 퍁뿌려져있음.

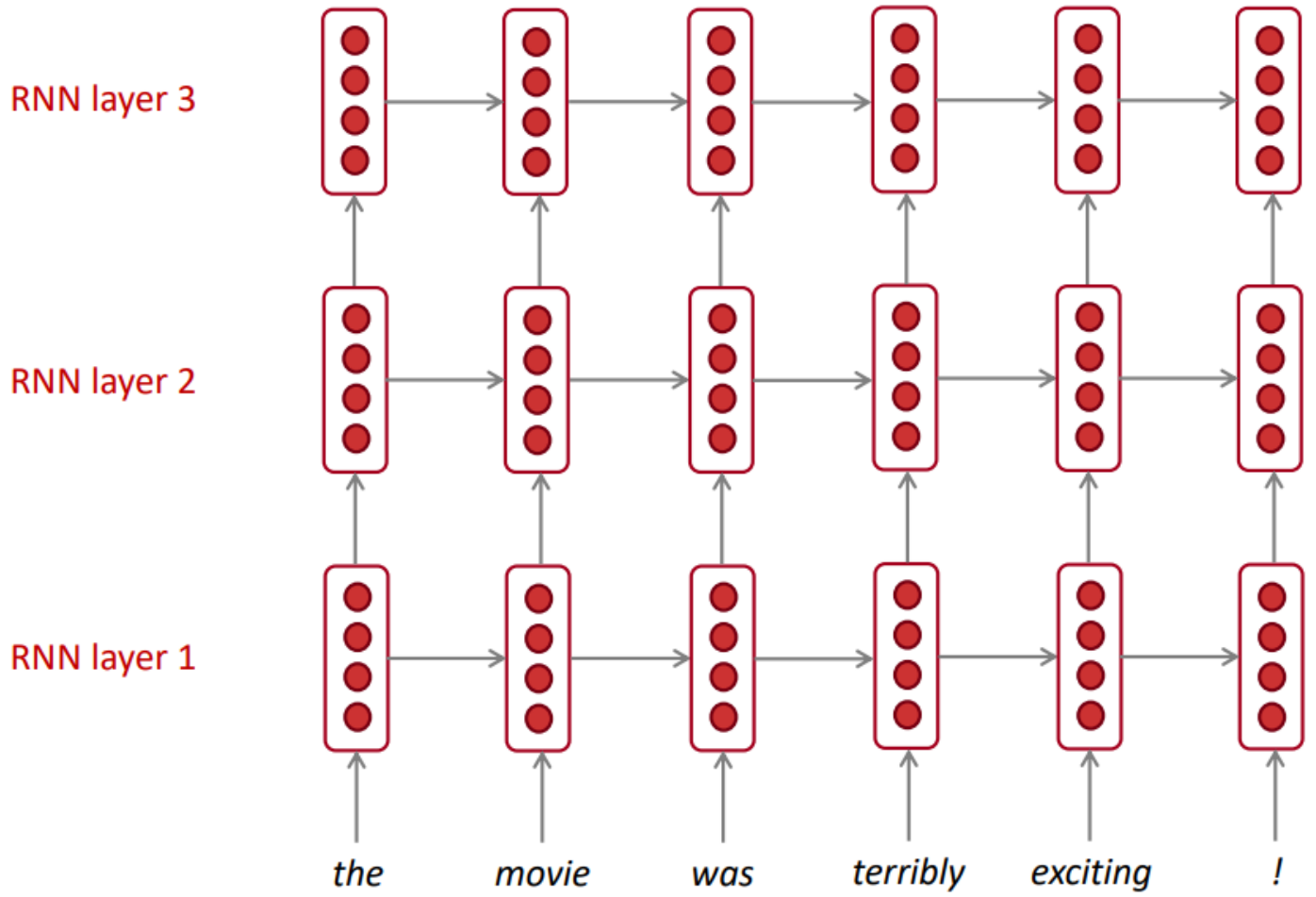
Multi-layer RNNS

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by applying multiple RNNs – this is a multi-layer RNN.
- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.
- Multi-layer RNNs are also called *stacked RNNs*.

Multi-layer RNNS

Multi-layer RNNs

The hidden states from RNN layer *i* are the inputs to RNN layer *i+1*



Multi-layer RNNS

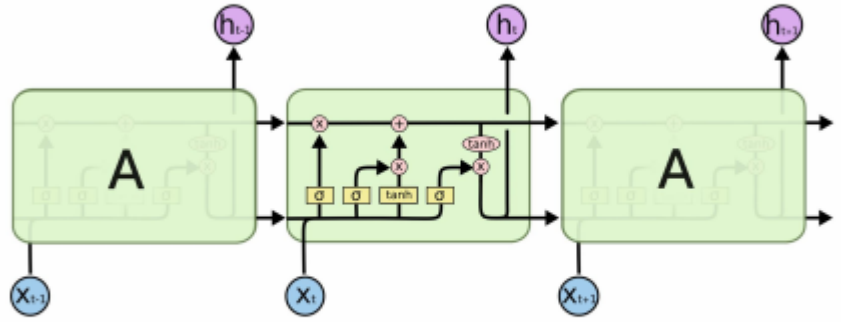
- High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks)

- For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - However, skip-connections/dense-connections are needed to train deeper RNNs (e.g. 8 layers)

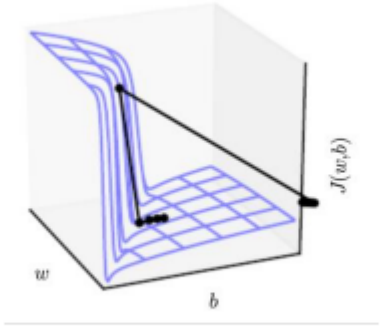
 # 알엔엔 정리
-1. 알에스티엠은 바워풀하고 지알유는 빨라
- 엘이스티엠은 좀더 피쳐적 파워풀.
- 지알유는 좀더 빨라요 파라미터 작아
-2. 이런 애들은 우리가 아는 그래디언트 배니싱을 해결 가능
- 그래서 바닐라 알엔엔ㅇ안써
-2. 우리는 피할수ㅇ쓰는 그래디언트 배니싱. 잘라도 생길수밖에
그래서 클리핑 주곤해요. 그럼 학습이 좀 더 안정화
-3. 바이디렉션이 가능하면 무조건 쓰게 해요. 못 쓰면 안쓰지만
쓰면
-4. 멀티레이어 좋은 선택지
- 그러나 학습이 더 안 되면 몇가지 트릭 선택해야하고
- 컨벌류션이나 (레즈넷) 등을 넣어줄 수 있음
- 그러면 모델 학습이 안정화되고 깊은 레이어 성공가능

Multi-layer RNNS

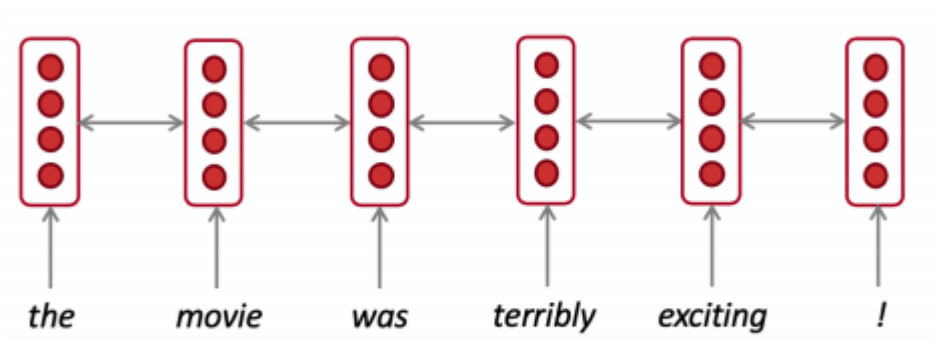
Lots of new information today! What are the practical takeaways?



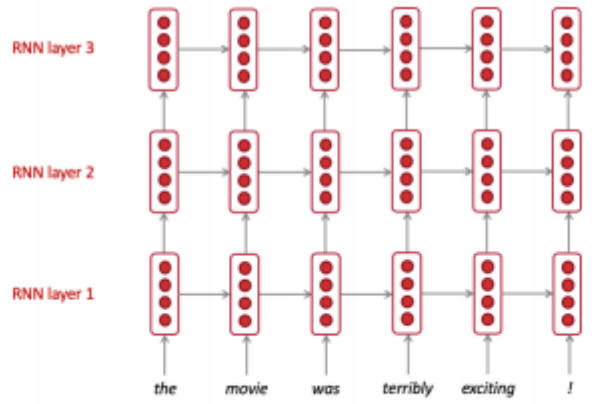
1. LSTMs are powerful but GRUs are faster



2. Clip your gradients



3. Use bidirectionality when possible



4. Multi-layer RNNs are powerful, but you might need skip/dense-connections if it's deep

References

Stanford University CS231n: Convolutional Neural Networks for Visual Recognition

Deep Learning Summer School, Montreal 2016 - VideoLectures.NET

Understanding LSTM Networks -- colah's blog

The Unreasonable Effectiveness of Recurrent Neural Networks

Stanford University CS224d: Deep Learning for Natural Language Processing

#멀티레이어
- 2개에서 4개 안에서 쌓아줍니다.
- 4개가 베스트한 퍼포먼스
- 그러나스키커넥션이나 댄스 커넥션할수있는데 댄스는 잊어도 괜ㅊ낳