# 텐서플로우 설치도 했고
# 튜토리얼도 봤고
# 기초 예제도 짜봤다면

김태훈
carpedm20

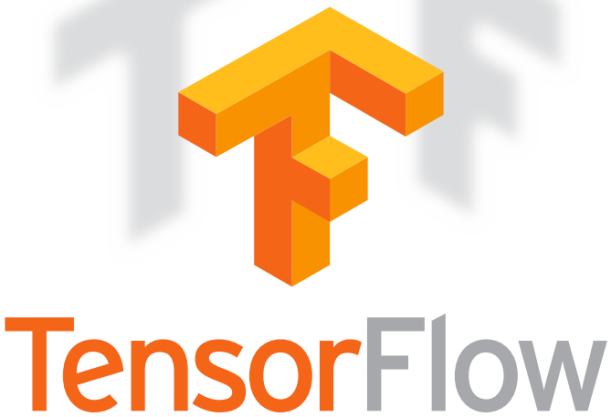
# 저는


**데브시스터즈** 병특

UNIST 학부 졸업

UNIST Statistical Artificial Intelligence Lab,
Lawrence Berkeley Lab에서 연구

http://carpedm20.github.io/


TensorFlow

# 2015년 12월 9일

Google Research Blog

The latest news from Research at Google

---

## TensorFlow - Google's latest machine learning system, open sourced for everyone

Monday, November 09, 2015

Posted by Jeff Dean, Senior Google Fellow, and Rajat Monga, Technical Lead

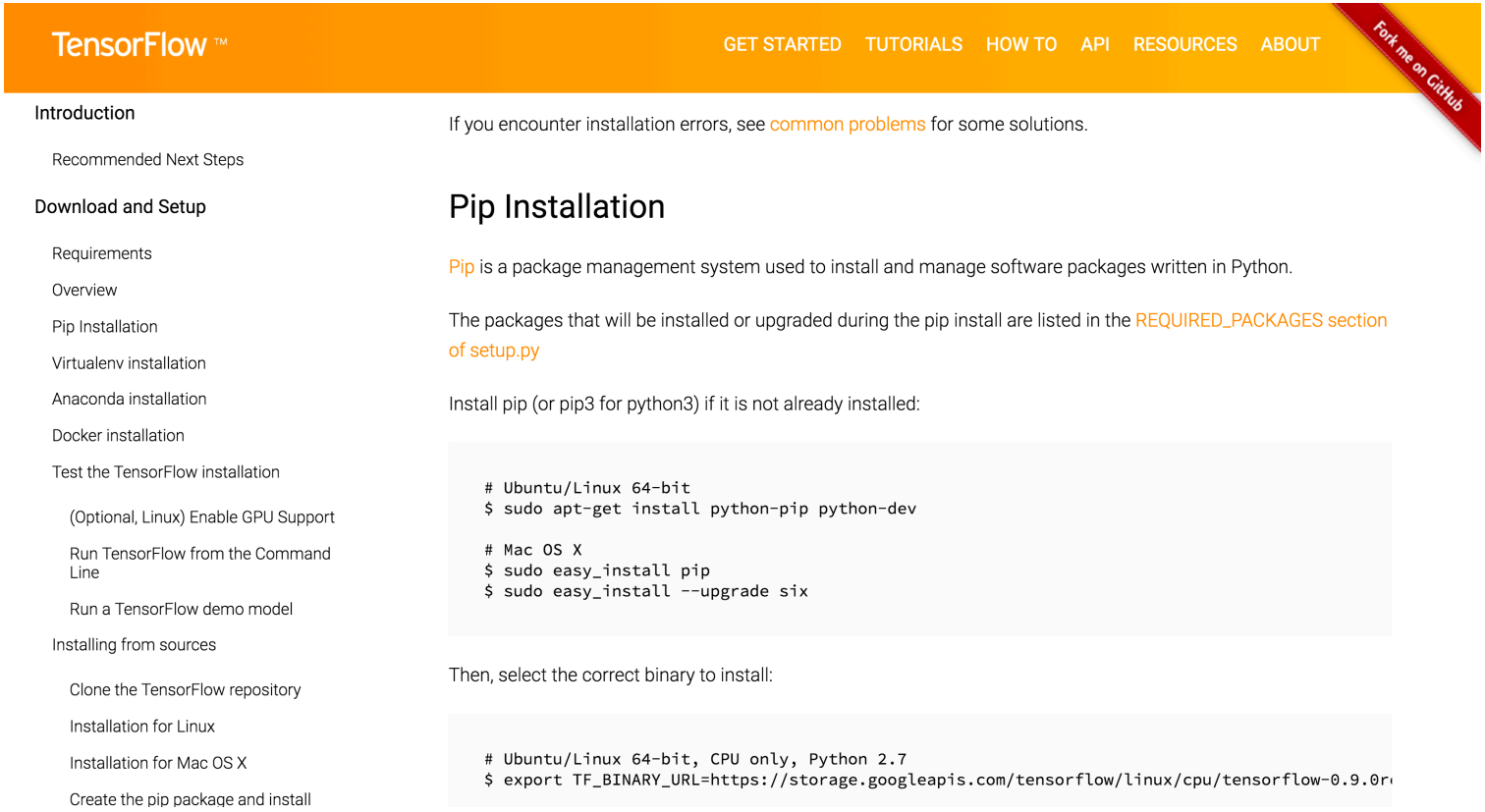
Deep Learning has had a huge impact on computer science, making it possible to explore new frontiers of research and to develop amazingly useful products that millions of people use every day. Our internal deep learning infrastructure DistBelief, developed in 2011, has allowed Googlers to build ever larger neural networks and scale training to thousands of cores in our datacenters. We've used it to demonstrate that concepts like "cat" can be learned from unlabeled YouTube images, to improve speech recognition in the Google app by 25%, and to build image search in Google Photos. DistBelief also trained the Inception model that won Imagenet's Large Scale Visual Recognition Challenge in 2014, and drove our experiments in automated image captioning as well as DeepDream.

# 188일이 지난 지금

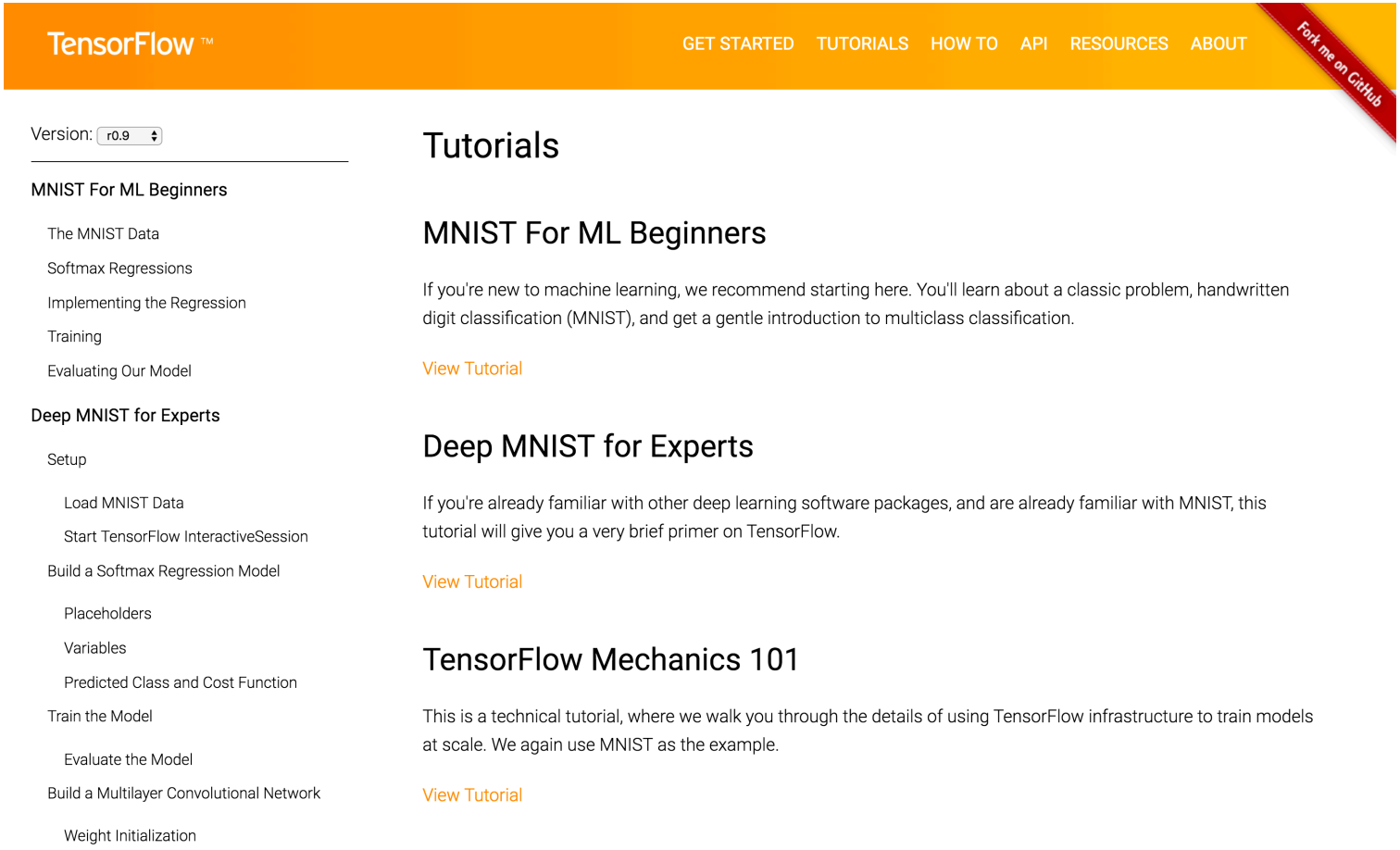
- 텐서플로우 튜토리얼
- 텐서플로우 설치
- 텐서플로우 도커 설치
- 텐서플로우 빌드
- 텐서플로우 개발 환경
- 텐서플로우 기본적인 사용 방법
- 텐서플로우 기초 실습
- 텐서플로우 …

But what's **next**?

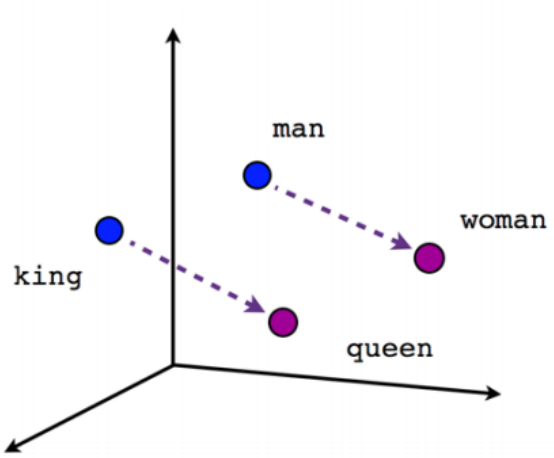
# 텐서플로우 설치도 했고


TensorFlow ™
GET STARTED
TUTORIALS
HOW TO
API
RESOURCES
ABOUT
Fork me on GitHub
Introduction
Recommended Next Steps
Download and Setup
Requirements
Overview
Pip Installation
Virtualenv installation
Anaconda installation
Docker installation
Test the TensorFlow installation
(Optional, Linux) Enable GPU Support
Run TensorFlow from the Command Line
Run a TensorFlow demo model
Installing from sources
Clone the TensorFlow repository
Installation for Linux
Installation for Mac OS X
Create the pip package and install
If you encounter installation errors, see common problems for some solutions.
Pip Installation
Pip is a package management system used to install and manage software packages written in Python.
The packages that will be installed or upgraded during the pip install are listed in the REQUIRED_PACKAGES section of setup.py
Install pip (or pip3 for python3) if it is not already installed:
# Ubuntu/Linux 64-bit
$ sudo apt-get install python-pip python-dev
# Mac OS X
$ sudo easy_install pip
$ sudo easy_install --upgrade six
Then, select the correct binary to install:
# Ubuntu/Linux 64-bit, CPU only, Python 2.7
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.9.0r

# 튜토리얼도 봤고


TensorFlow ™
GET STARTED
TUTORIALS
HOW TO
API
RESOURCES
ABOUT
Fork me on GitHub
Version: r0.9
MNIST For ML Beginners
The MNIST Data
Softmax Regressions
Implementing the Regression
Training
Evaluating Our Model
Deep MNIST for Experts
Setup
Load MNIST Data
Start TensorFlow InteractiveSession
Build a Softmax Regression Model
Placeholders
Variables
Predicted Class and Cost Function
Train the Model
Evaluate the Model
Build a Multilayer Convolutional Network
Weight Initialization
Tutorials
MNIST For ML Beginners
If you're new to machine learning, we recommend starting here. You'll learn about a classic problem, handwritten digit classification (MNIST), and get a gentle introduction to multiclass classification.
View Tutorial
Deep MNIST for Experts
If you're already familiar with other deep learning software packages, and are already familiar with MNIST, this tutorial will give you a very brief primer on TensorFlow.
View Tutorial
TensorFlow Mechanics 101
This is a technical tutorial, where we walk you through the details of using TensorFlow infrastructure to train models at scale. We again use MNIST as the example.
View Tutorial

# 기초 예제도 짜봤다면 (보는것만이 전부는 아니죠)



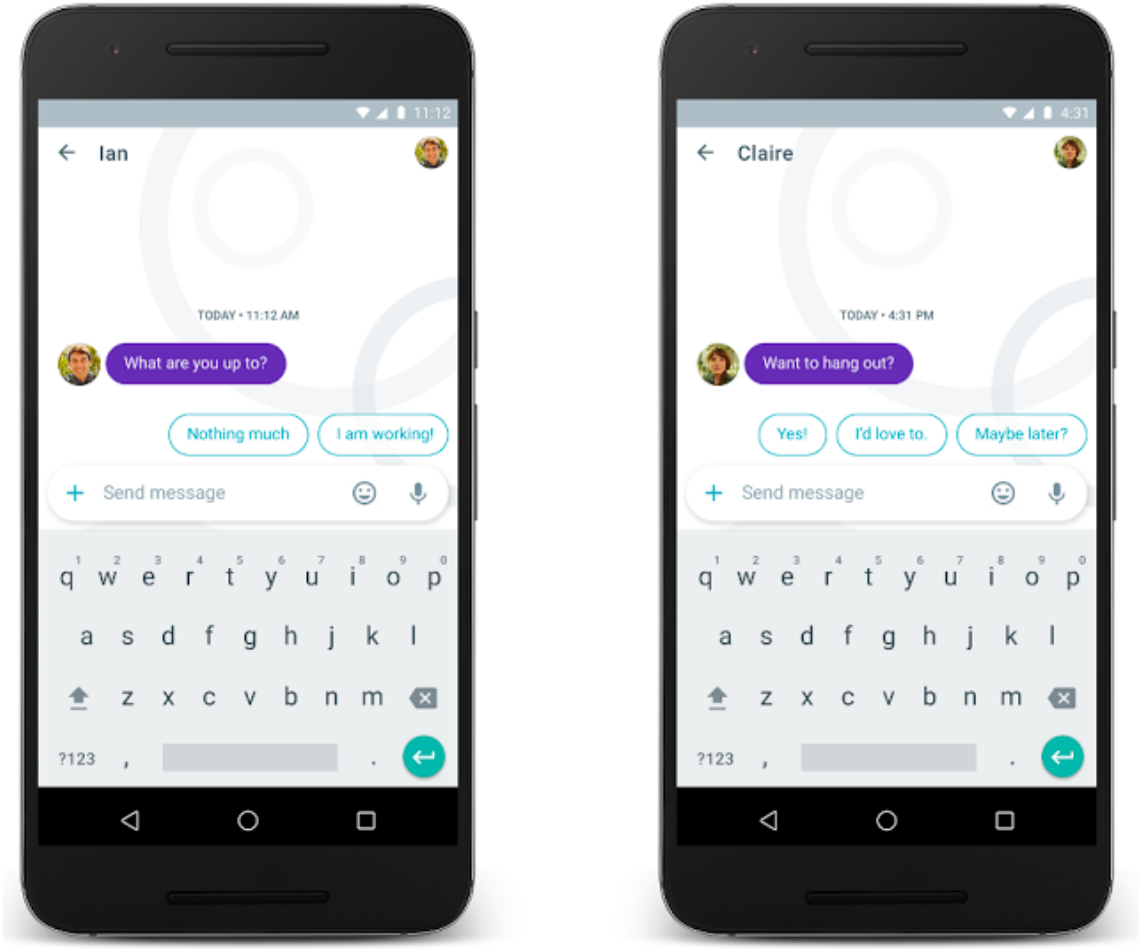

Word2vec


Neural Style transfer


# 이제 나만의 모델을 짜고 싶다

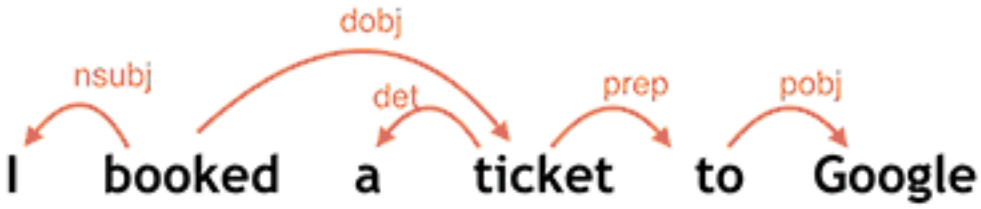
이제 CNN, LSTM도 뭔지 알겠고 코드도 짤 수 있다

**그럼 어떤 문제를 풀어볼까?**




https://research.googleblog.com/2016/05/chat-smarter-with-allo.html

# Dependency Parsing




https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

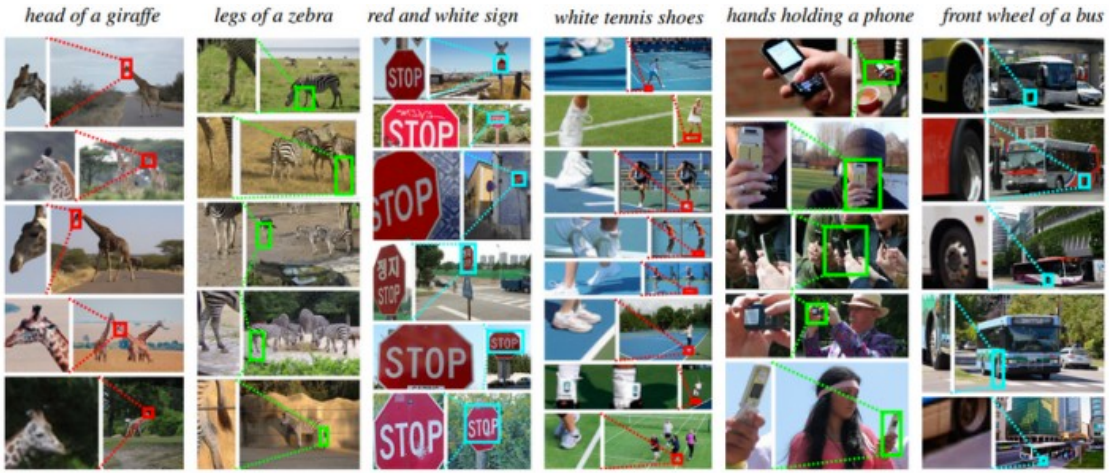
https://research.googleblog.com/2016/06/motion-stills-create-beautiful-gifs.html

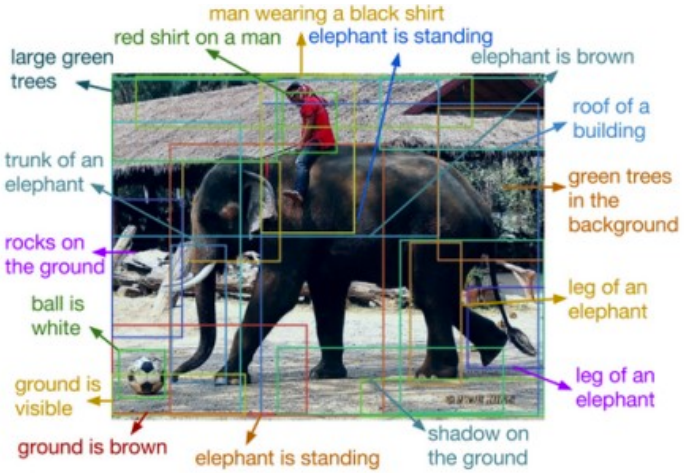
그럼 **convolution** 몇 개 쌓고 끝에 **fully connected** 붙여서…

그냥 multilayer **RNN 혹은 LSTM**에다가 단어를 input으로 넣어서…

하지만 모델, loss나 training에 대한 깊은 고민을 하지 않으면

사진에서 object를 찾아서 그것을 문장으로 설명할 수  없고

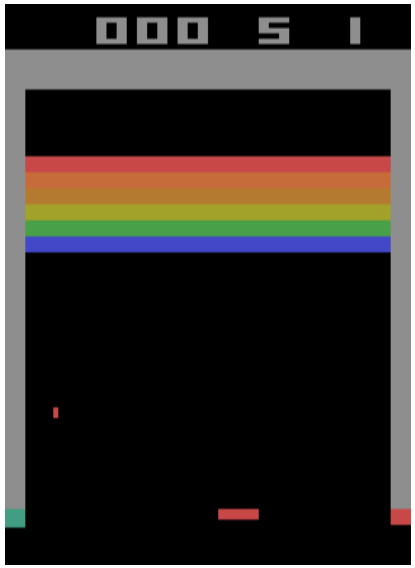



http://sanghyukchun.github.io/92/

하지만 모델, loss나 training에 대한 깊은 고민을 하지 않으면

벽돌 깨기도 제대로 못하며




https://www.nervanasys.com/demystifying-deep-reinforcement-learning/

하지만 모델, loss나 training에 대한 깊은 고민을 하지 않으면

사람 얼굴을 이해하고 얼굴 사진을 만드는 모델도 못 만든다

http://carpedm20.github.io/faces/


매일 매일 딥러닝으로

마법과 같은 결과를 보여주는 그들


그들의 생각을 알아야 한다

즉, 그들의 논문을 이해하고 내것으로 만드는것이 필수적이다


TensorFlow 가 공개된 후 시작한 딥러닝 공부

가장 효과적인 방법은 그들의 논문을 구현하는 것

지금까지 제가 구현한 논문은

# 지금까지 제가 구현한 논문은

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- Deep Visual Analogy-Making

Computer Vision

# 지금까지 제가 구현한 논문은

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- Deep Visual Analogy-Making

Computer Vision

- End-To-End Memory Networks
- Neural Variational Inference for Text Processing
- Character-Aware Neural Language Models
- Teaching Machines to Read and Comprehend

Natural Language Processing

# 지금까지 제가 구현한 논문은

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- Deep Visual Analogy-Making

Computer Vision

---

- End-To-End Memory Networks
- Neural Variational Inference for Text Processing
- Character-Aware Neural Language Models
- Teaching Machines to Read and Comprehend

Natural Language Processing

---

- Playing Atari with Deep Reinforcement Learning
- Human-level control through deep reinforcement learning
- Deep Reinforcement Learning with Double Q-learning
- Dueling Network Architectures for Deep Reinforcement Learning
- Language Understanding for Text-based Games Using Deep Reinforcement Learning
- Asynchronous Methods for Deep Reinforcement Learning

Reinforcement Learning

# 지금까지 제가 구현한 논문은

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- Deep Visual Analogy-Making — Computer Vision

---

- End-To-End Memory Networks
- Neural Variational Inference for Text Processing
- Character-Aware Neural Language Models
- Teaching Machines to Read and Comprehend — Natural Language Processing

---

- Playing Atari with Deep Reinforcement Learning
- Human-level control through deep reinforcement learning
- Deep Reinforcement Learning with Double Q-learning
- Dueling Network Architectures for Deep Reinforcement Learning
- Language Understanding for Text-based Games Using Deep Reinforcement Learning
- Asynchronous Methods for Deep Reinforcement Learning — Reinforcement Learning

---

- Neural Turing Machines — Algorithm Learning

# 재밌었던 논문은

- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks # 모델이 흥미로움
- Deep Visual Analogy-Making

---

- End-To-End Memory Networks # 간단하면서도 쉽고 결과도 재밌음
- Neural Variational Inference for Text Processing
- Character-Aware Neural Language Models
- Teaching Machines to Read and Comprehend

---

- Playing Atari with Deep Reinforcement Learning # RL 입문으로 괜찮
- Human-level control through deep reinforcement learning
- Deep Reinforcement Learning with Double Q-learning
- Dueling Network Architectures for Deep Reinforcement Learning
- Language Understanding for Text-based Games Using Deep Reinforcement Learning
- Asynchronous Methods for Deep Reinforcement Learning # 구현이 challenging함

---

- Neural Turing Machines # 모델이 흥미로움

# 모든 코드는 Github에

- https://github.com/carpedm20/DCGAN-tensorflow
- https://github.com/carpedm20/visual-analogy-tensorflow — Computer Vision

---

- https://github.com/carpedm20/MemN2N-tensorflow
- https://github.com/carpedm20/variational-text-tensorflow
- https://github.com/carpedm20/lstm-char-cnn-tensorflow
- https://github.com/carpedm20/attentive-reader-tensorflow — Natural Language Processing

---

- https://github.com/devsisters/DQN-tensorflow
- https://github.com/devsisters/DQN-tensorflow
- https://github.com/carpedm20/deep-rl-tensorflow
- https://github.com/carpedm20/deep-rl-tensorflow
- https://github.com/carpedm20/text-based-game-rl-tensorflow
- https://github.com/devsisters/async-rl-tensorflow — Reinforcement Learning

---

- https://github.com/carpedm20/NTM-tensorflow — Algorithm Learning

# 오늘 소개할 논문들은 두개

Computer Vision

- End-To-End Memory Networks

Natural Language Processing

- Asynchronous Methods for Deep Reinforcement Learning

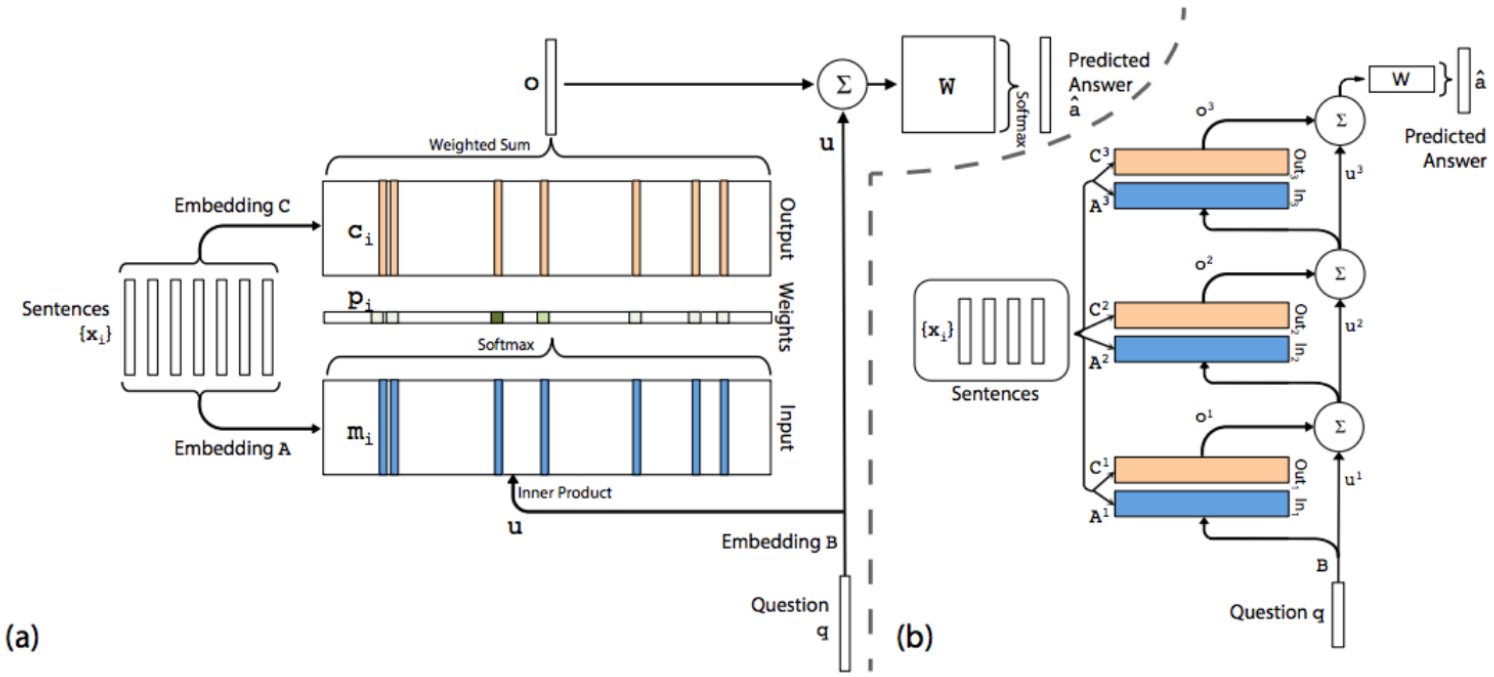
Reinforcement Learning

Algorithm Learning

# End-to-End Memory Network

# 핵심: CNN, RNN에서 벗어나자

# End-to-End Memory Network (E2E)

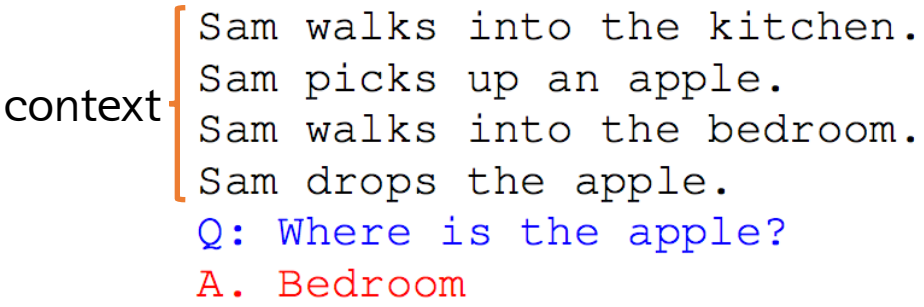



Sukhbaatar, S., Weston, J., & Fergus, R. (NIPS 2015)

https://github.com/carpedm20/MemN2N-tensorflow

# 해결한 문제

1. bAbI


context
Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.
Q: Where is the apple?
A. Bedroom

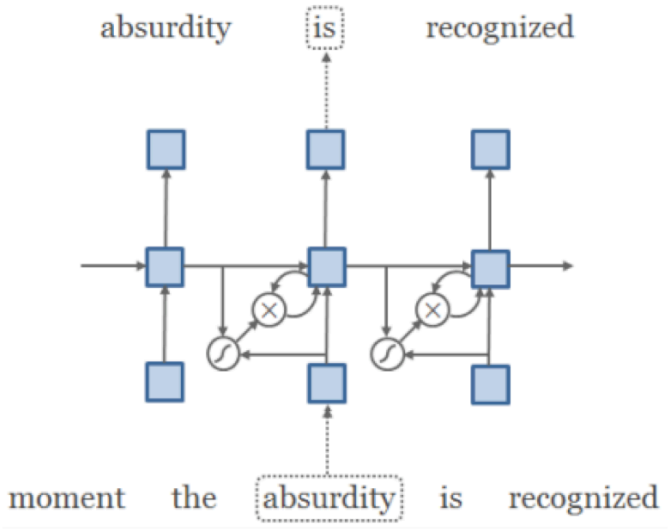
# 해결한 문제

1. bAbI

context

```
Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.
Q: Where is the apple?
A. Bedroom
```

2. Language model



# 한 문장 정리

Recurrent attention model

with external memory

# 필요한 사전 지식

Recurrent Neural Network

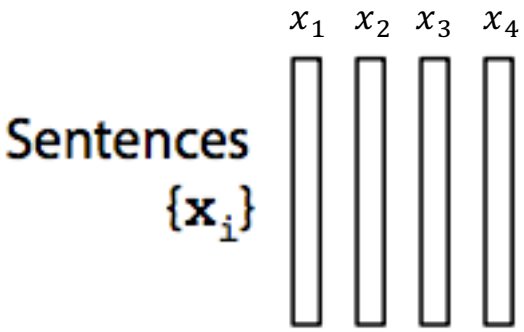
Recurrent attention model

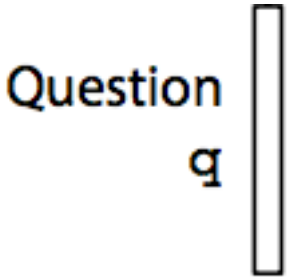
Soft and hard attention in Neural Network

with external memory

Internal and external memory access in Neural Network

# Input: Context 문장들과 질문

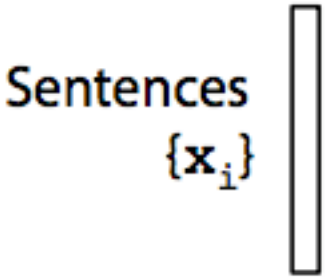



$x_1 =$ Mary journeyed to the den.
$x_2 =$ Mary went back to the kitchen.
$x_3 =$ John journeyed to the bedroom.
$x_4 =$ Mary discarded the milk.

Q: Where was the milk before the den?

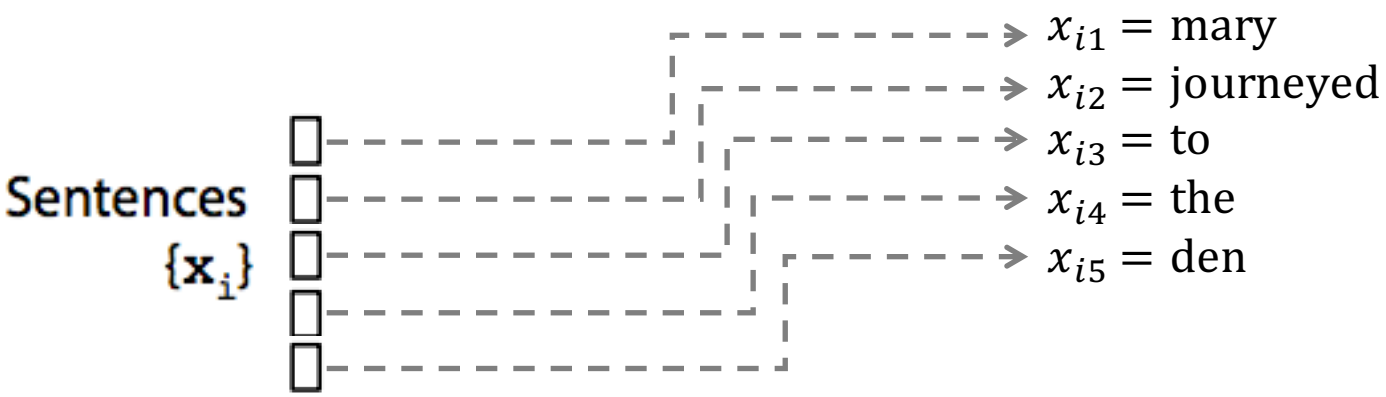
# 문장 하나: $x_i$




$x_i$ = Mary journeyed to the den.

# 문장 하나: 단어의 조합

$$x_i = \{x_{i1}, x_{i2,} \ldots, x_{in}\}$$

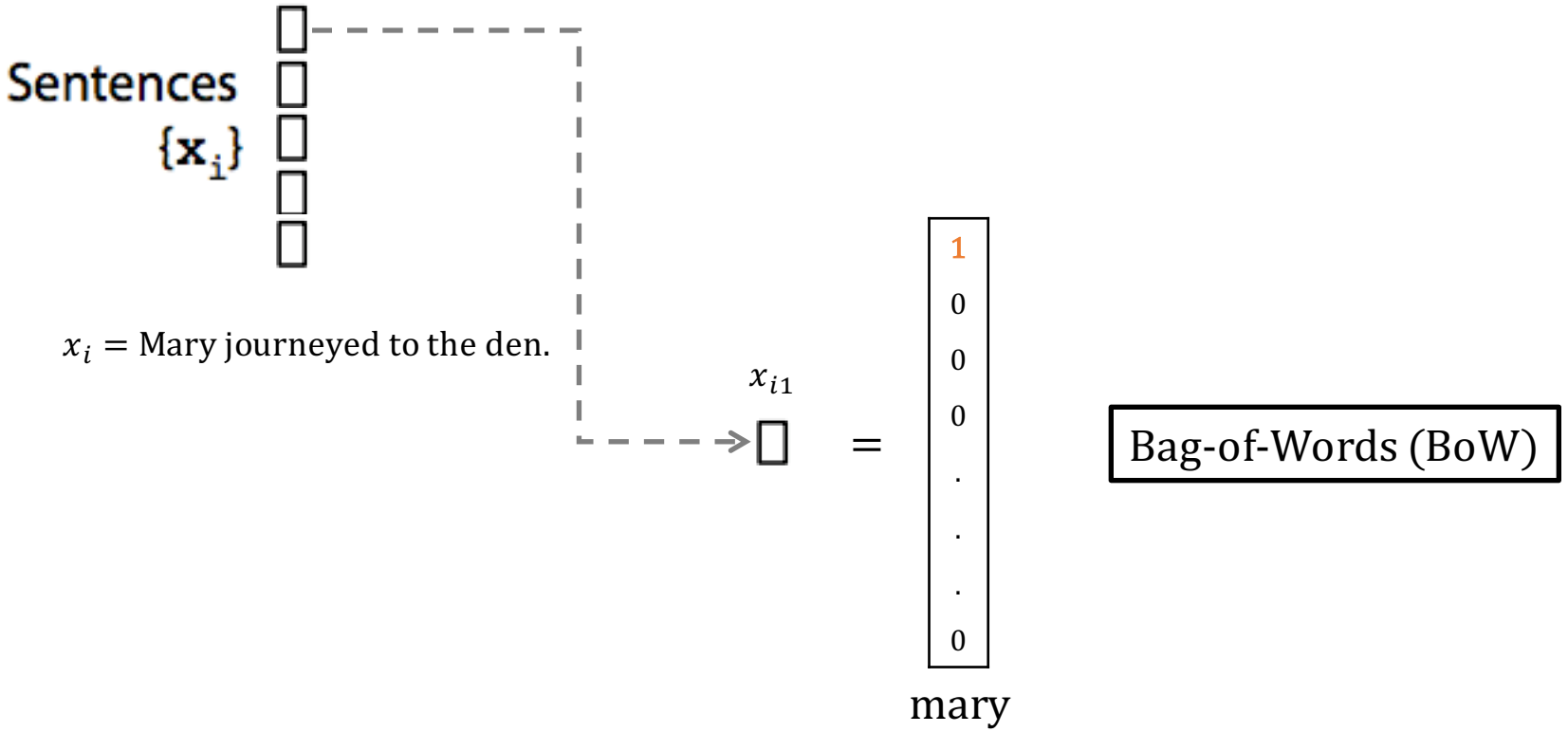



$x_i$ = Mary journeyed to the den.

# 단어 하나: Bag-of-Words (BoW)로 표현

$$x_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\}$$

$x_{i1} = \text{mary}$



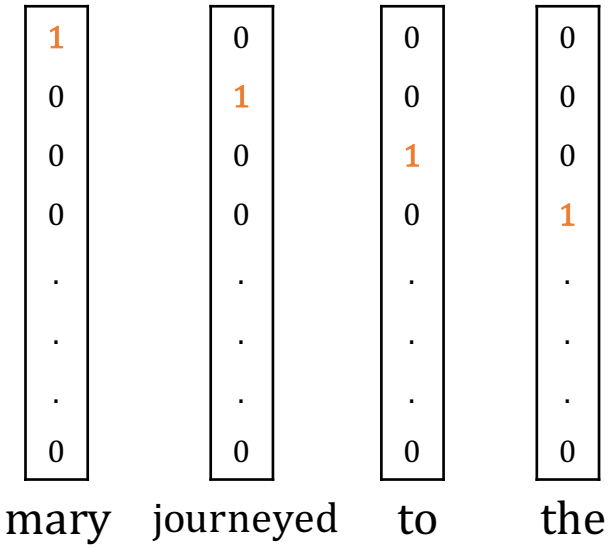
# 문장 하나: BoW의 set

$$x_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\}$$

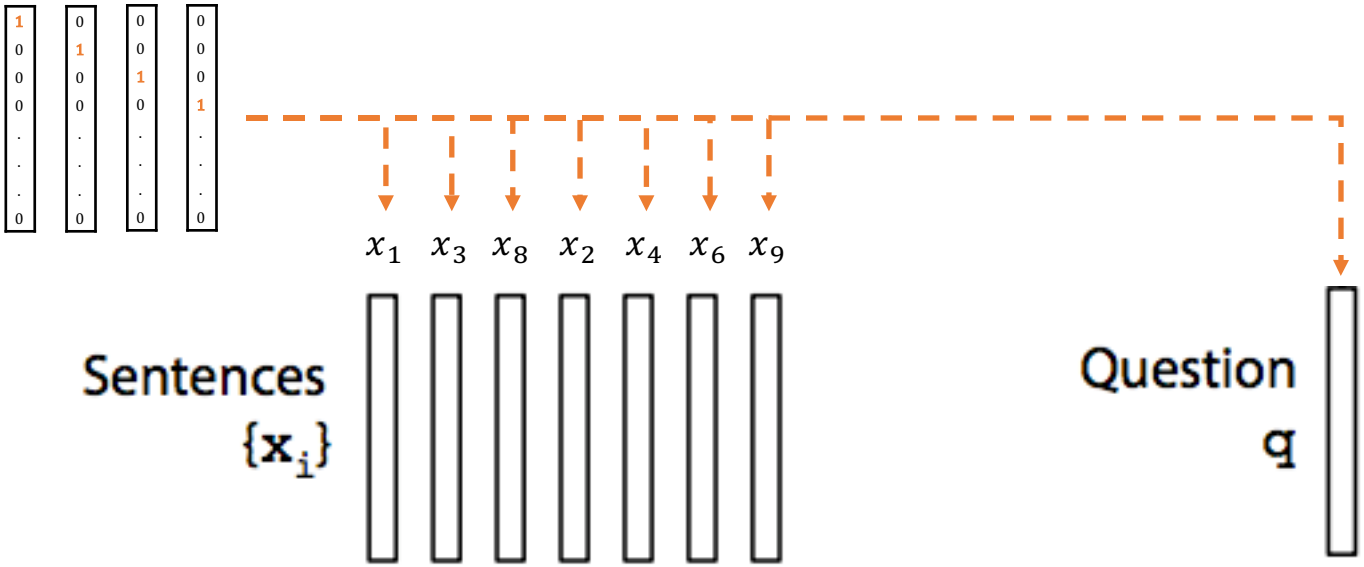
$x_{i1}$ = mary
$x_{i2}$ = journeyed
$x_{i3}$ = to
$x_{i4}$ = the
$x_{i5}$ = den

Sentences $\{x_i\}$

$x_i$ = Mary journeyed to the den.



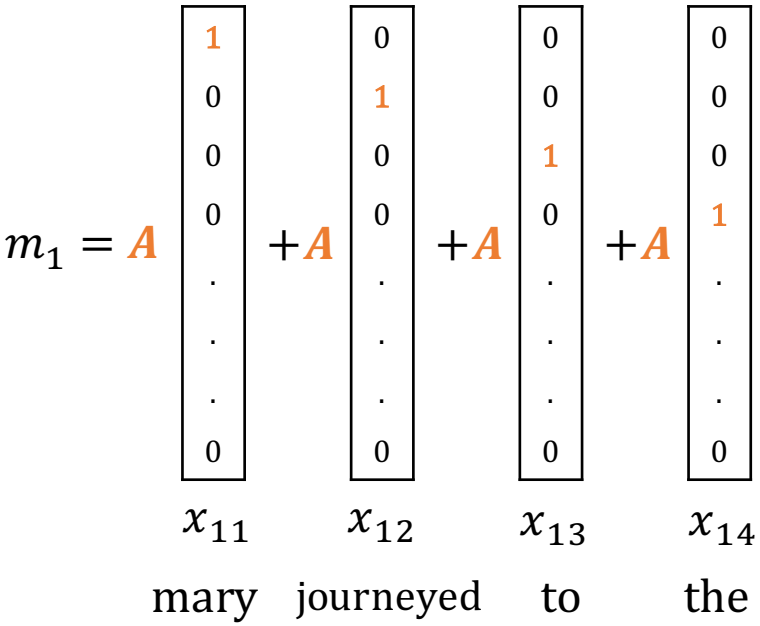
# Input: BoW로 표현된 Context 문장들과 질문




$x_1$ = Mary journeyed to the den.
$x_2$ = Mary went back to the kitchen.
$x_3$ = John journeyed to the bedroom.
$x_4$ = Mary discarded the milk.

Q: Where was the milk before the den?

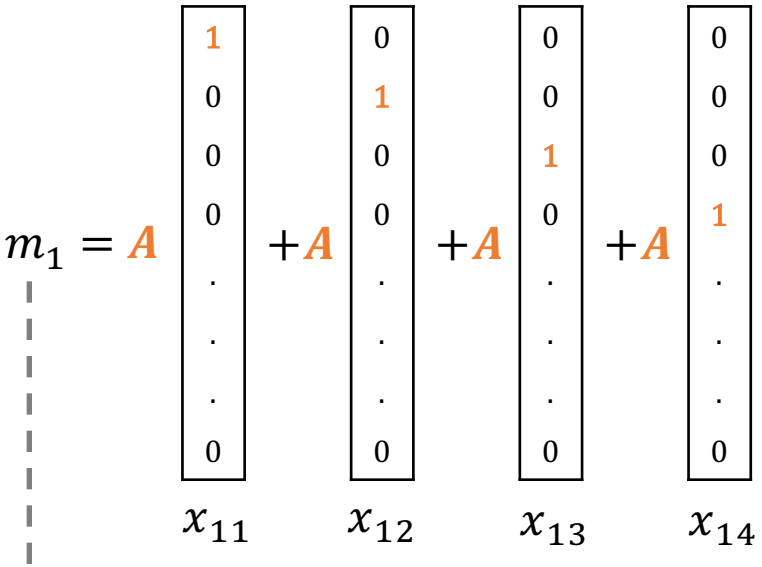
# Memory: 문장 하나로부터 만들어진다




$$m_i = \sum_j A x_{ij}$$

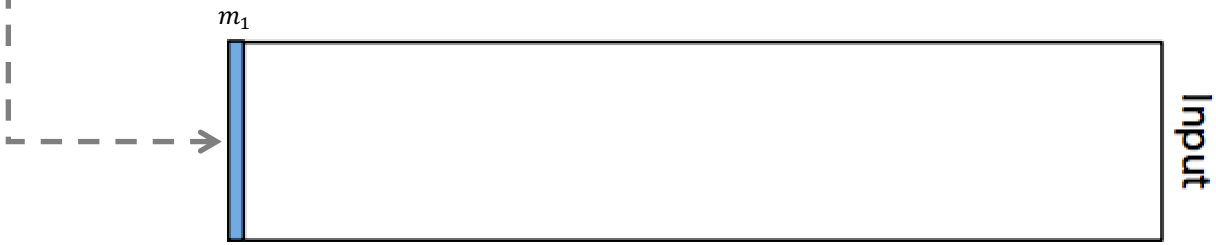
$A$: embedding matrix

# Recurrent attention model with external memory

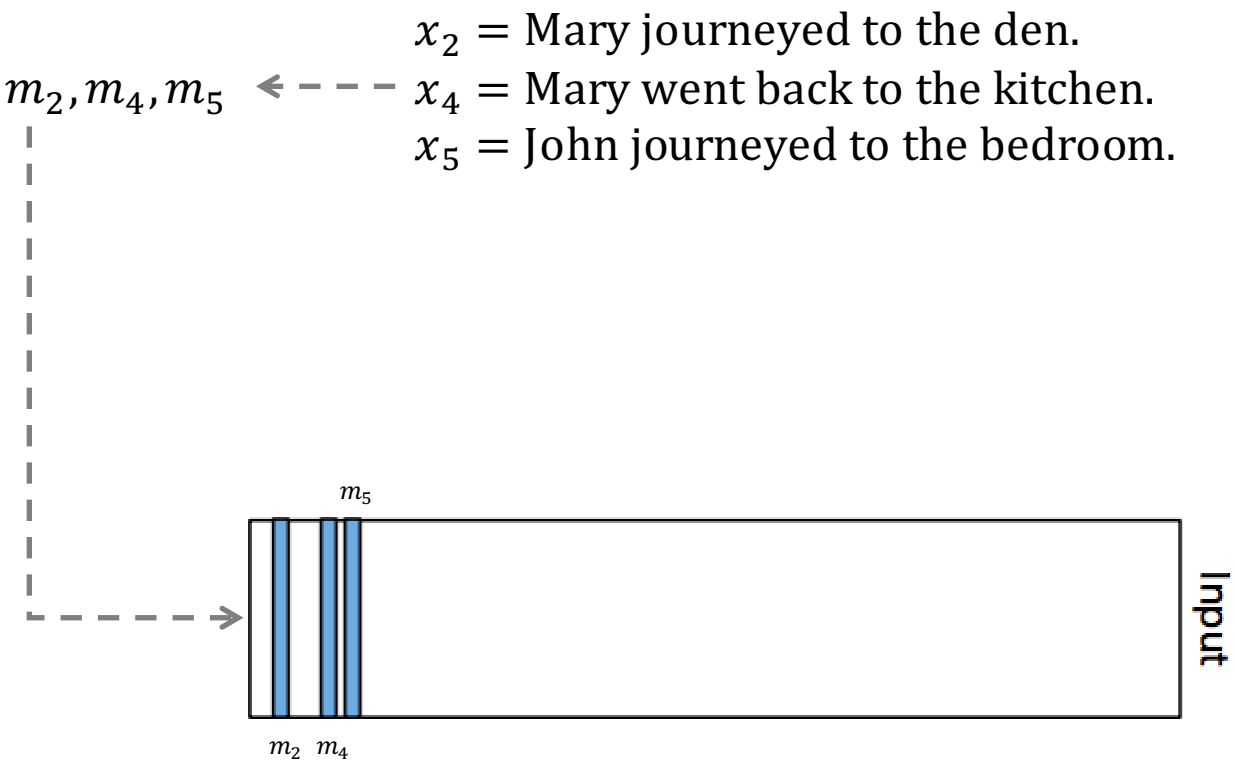



$$m_i = \sum_j A x_{ij}$$

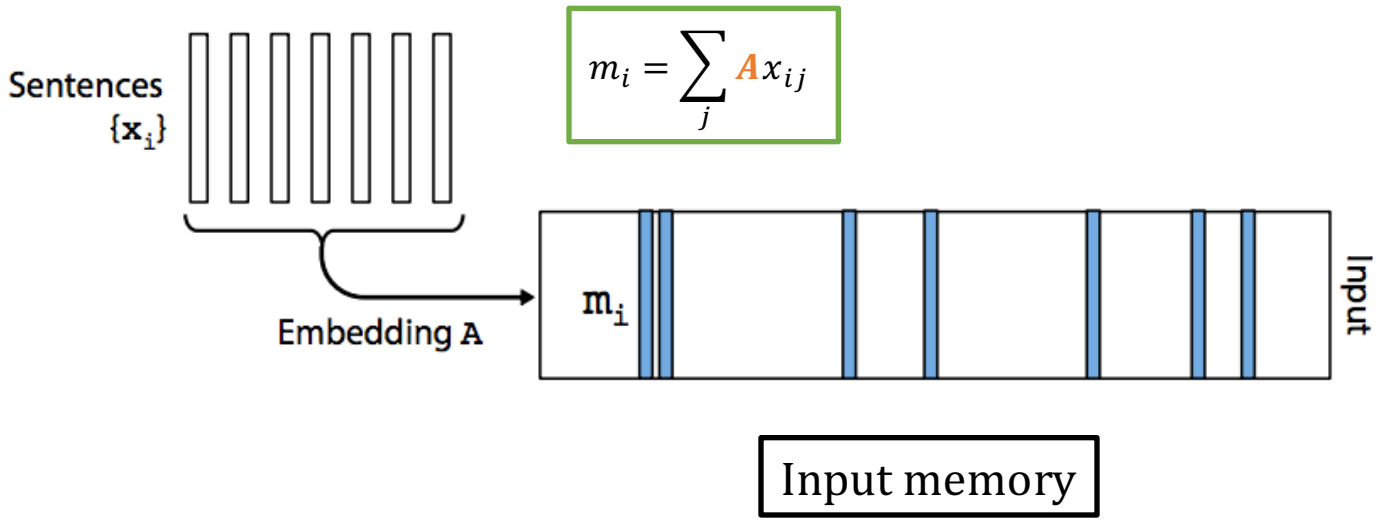
$A$: embedding matrix



# Memory: 한 task에는 여러개가 사용된다

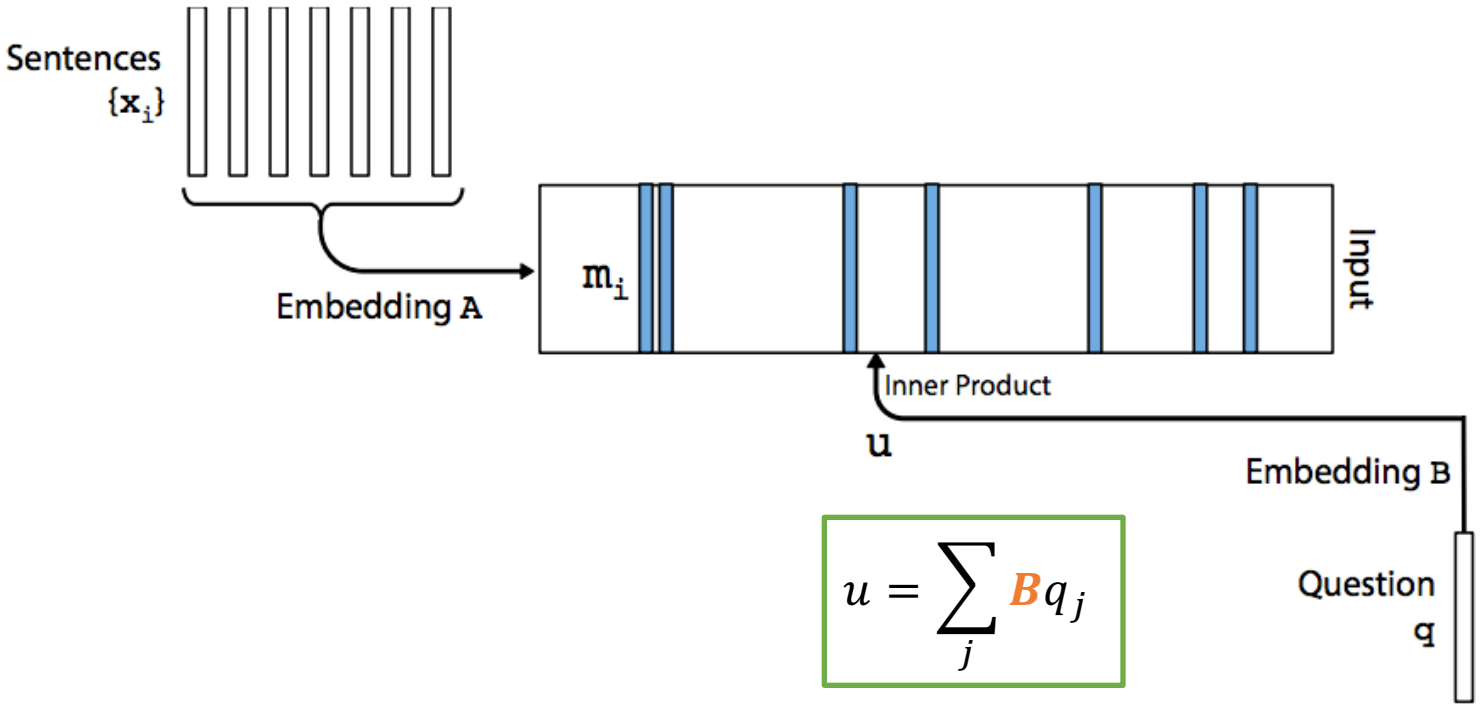


# Memory: 필요한 것만 Input으로 사용




실제로 메모리의 본체는 embedding matrix인 $A$이며, $A$가 점차 학습된다

# Recurrent attention model with external memory

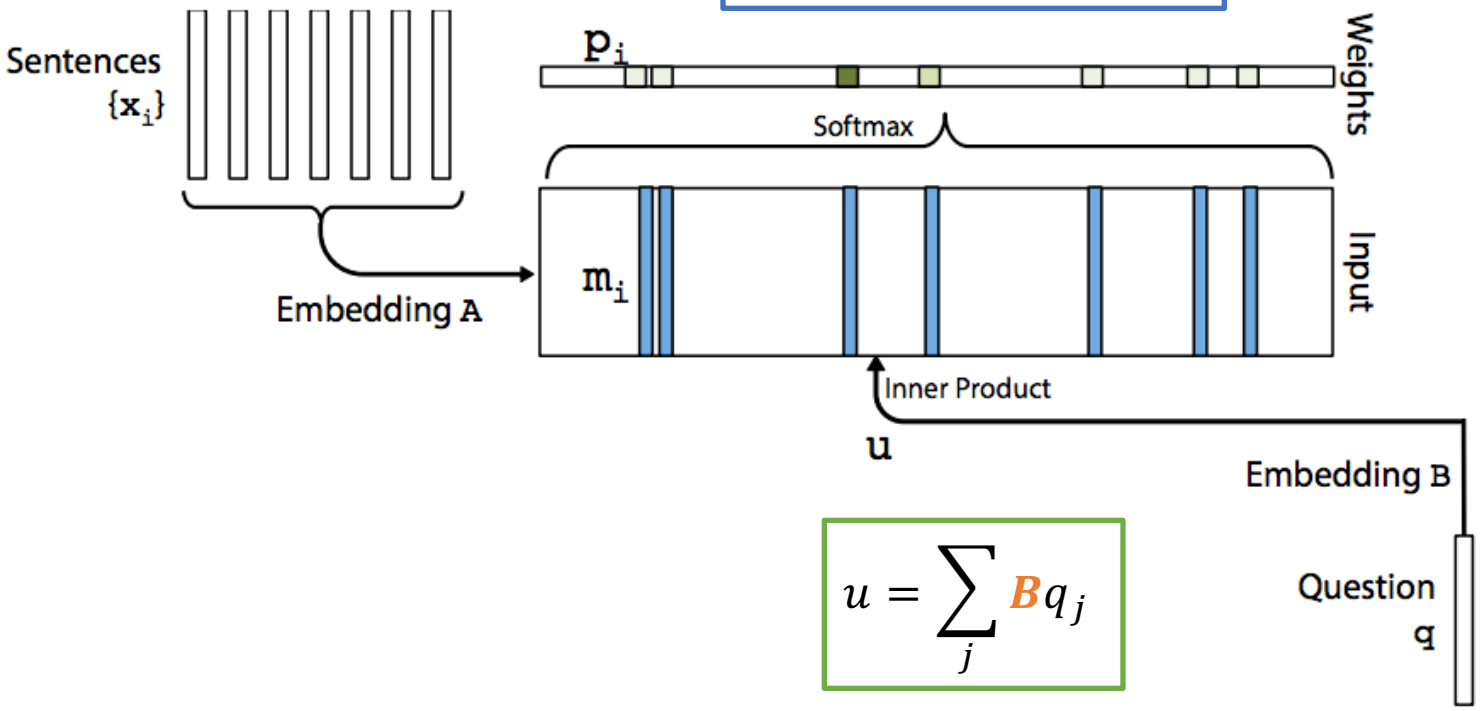



$$u = \sum_j \boldsymbol{B} q_j$$

# Recurrent attention model with external memory

얼마나 집중할 것인지

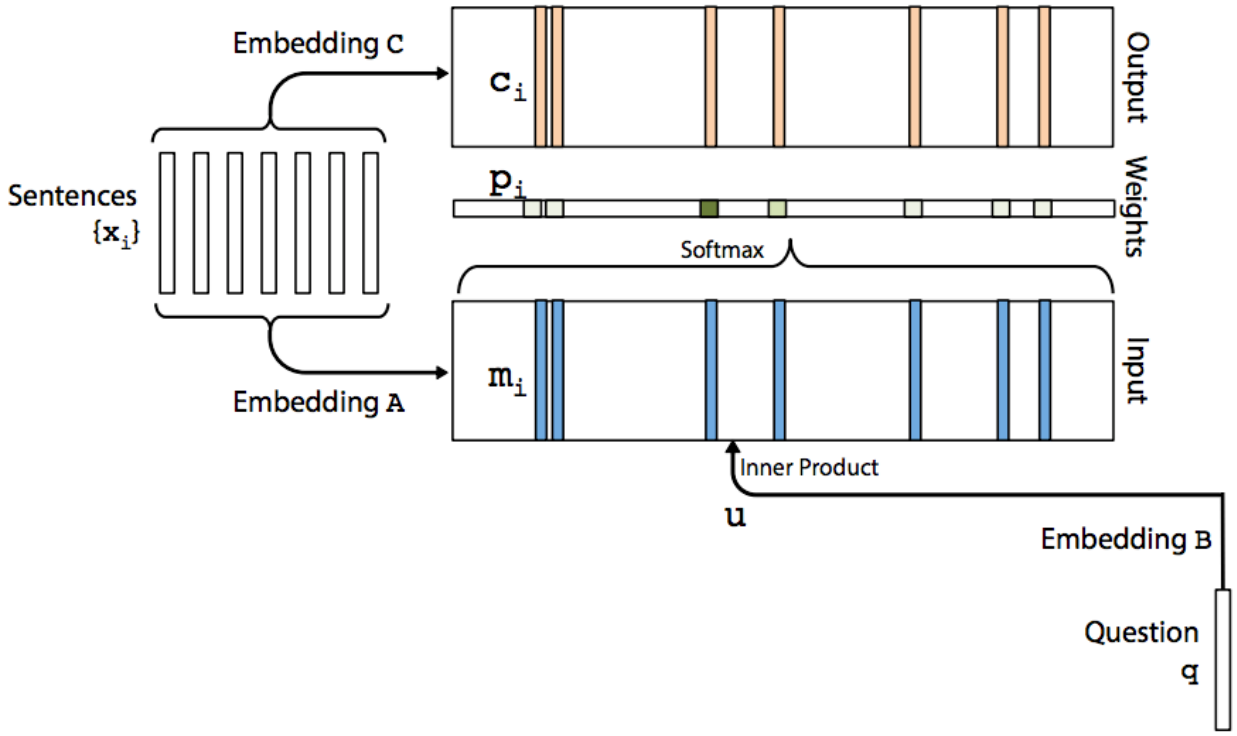
$$p_i = Softmax(u^T m_t)$$



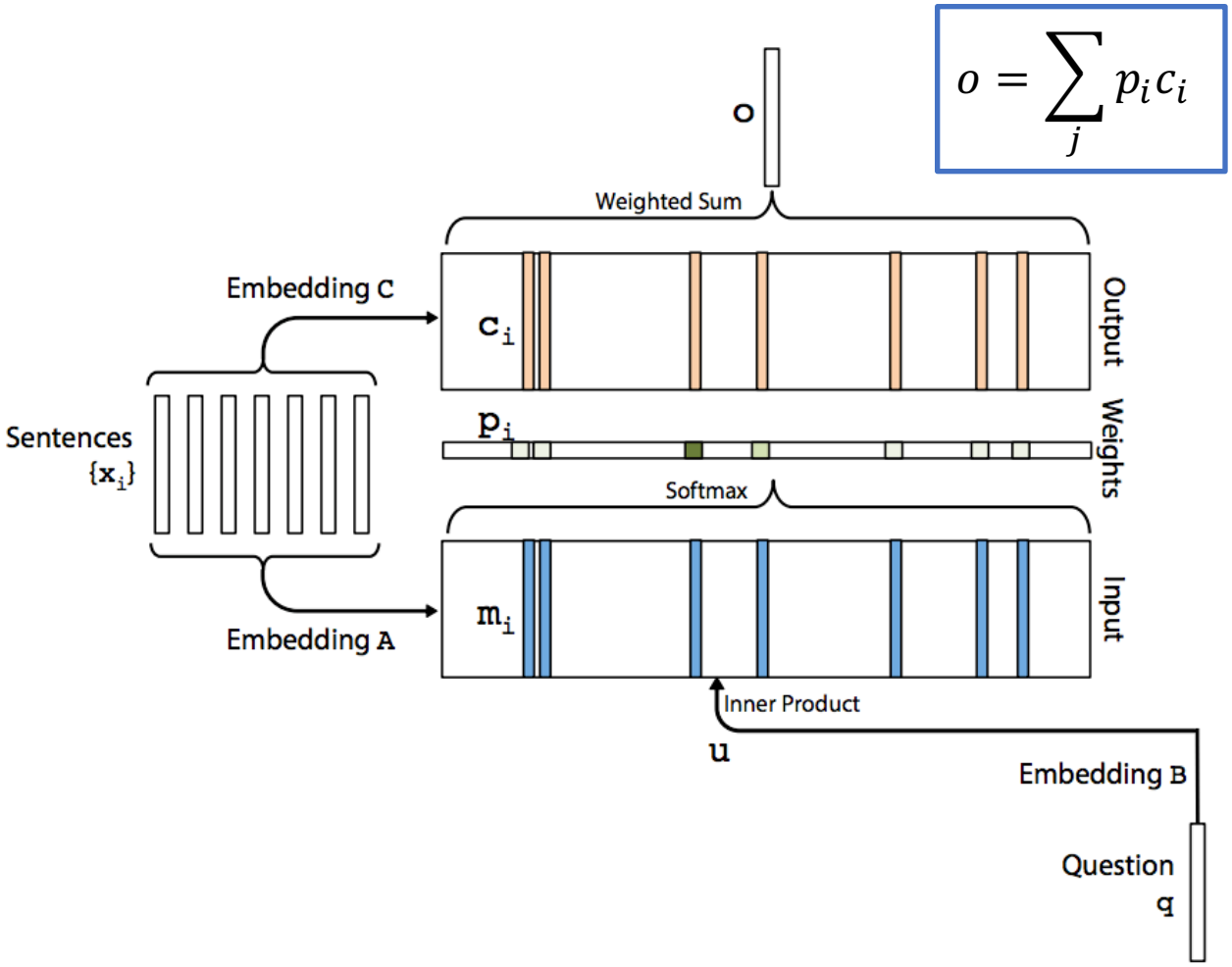

$$u = \sum_j \boldsymbol{B} q_j$$

# Recurrent attention model with external memory

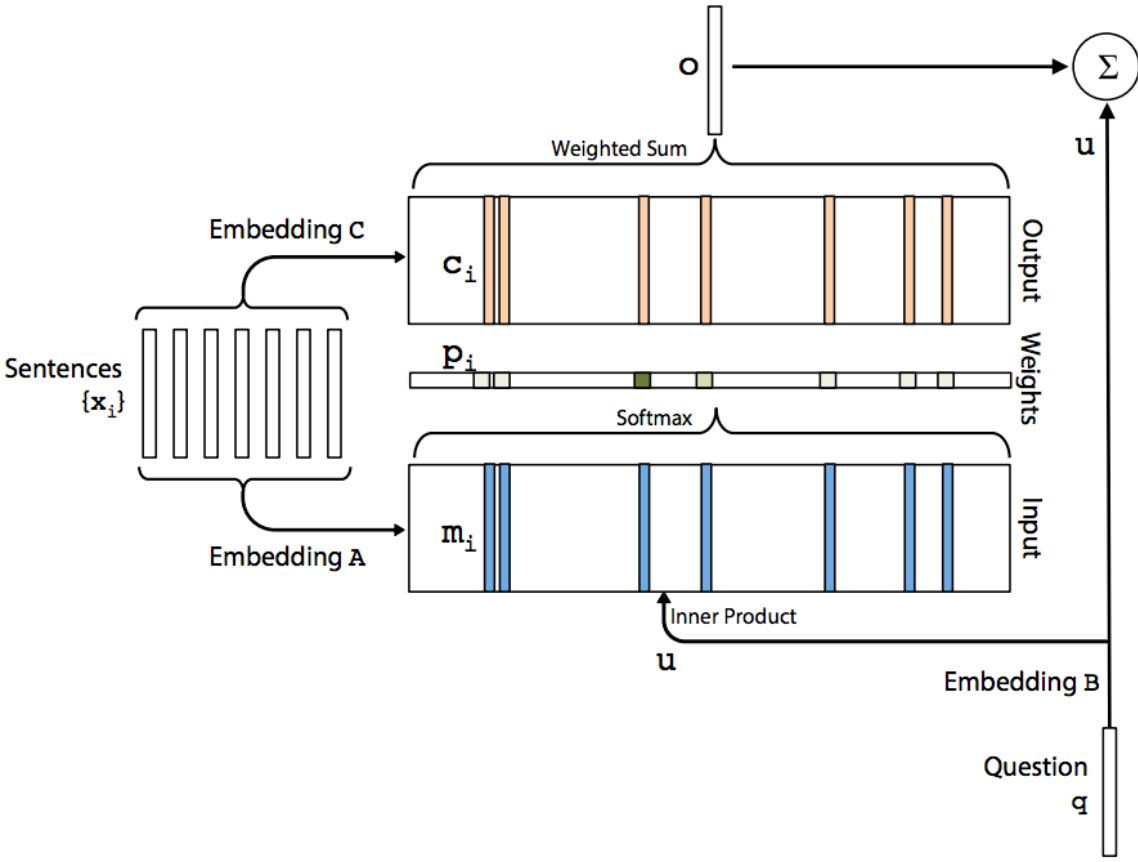
$$c_i = \sum_j C x_{ij}$$



# Recurrent attention model with external memory



# Output: 요약된 정보 $o$ + 질문 정보 $u$

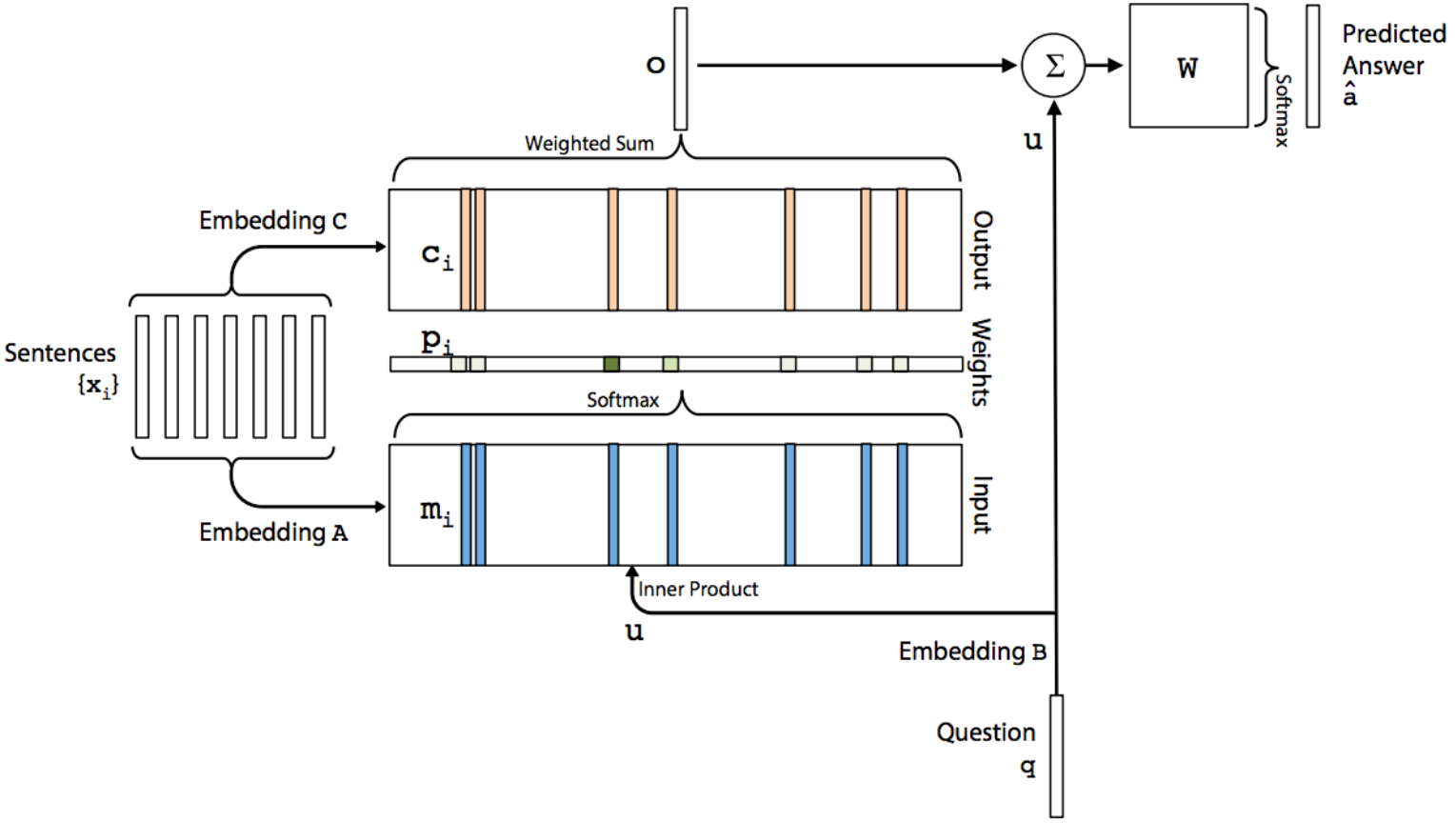



$o + u$

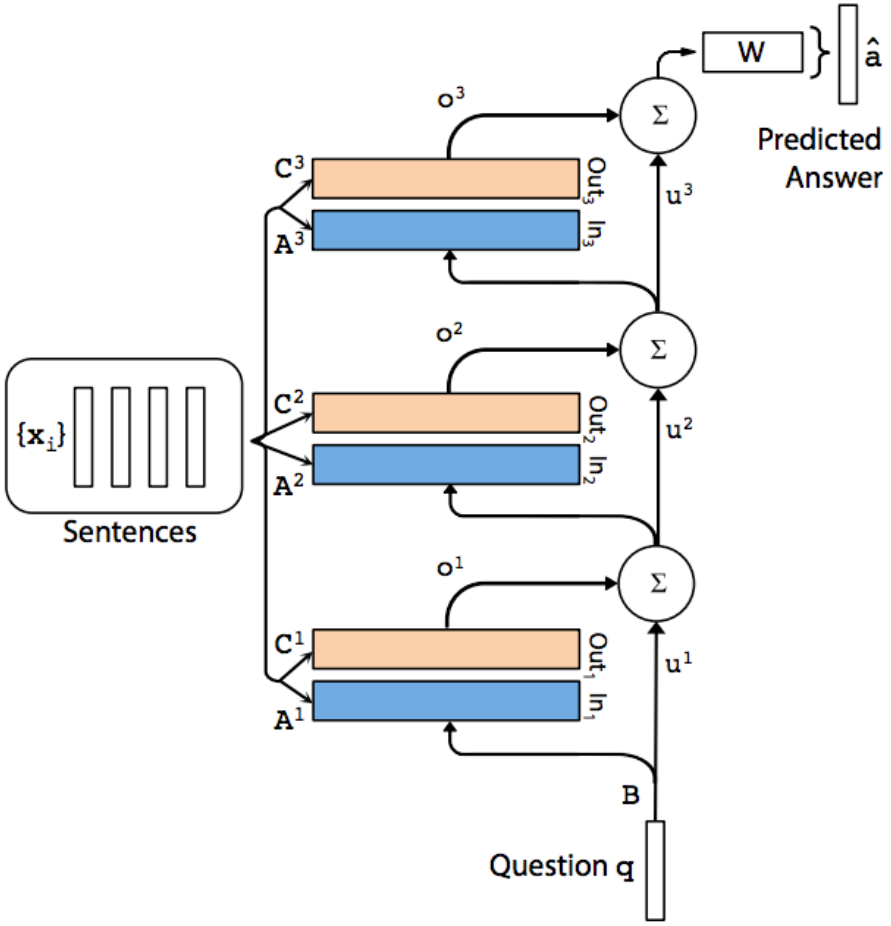
$o, u$ 둘 다 고려해 답을 도출

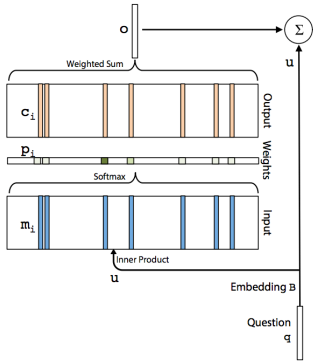
# Output: 실제로 정답 단어 $\hat{a}$

$\hat{a}$=Softmax($W$(o+u))

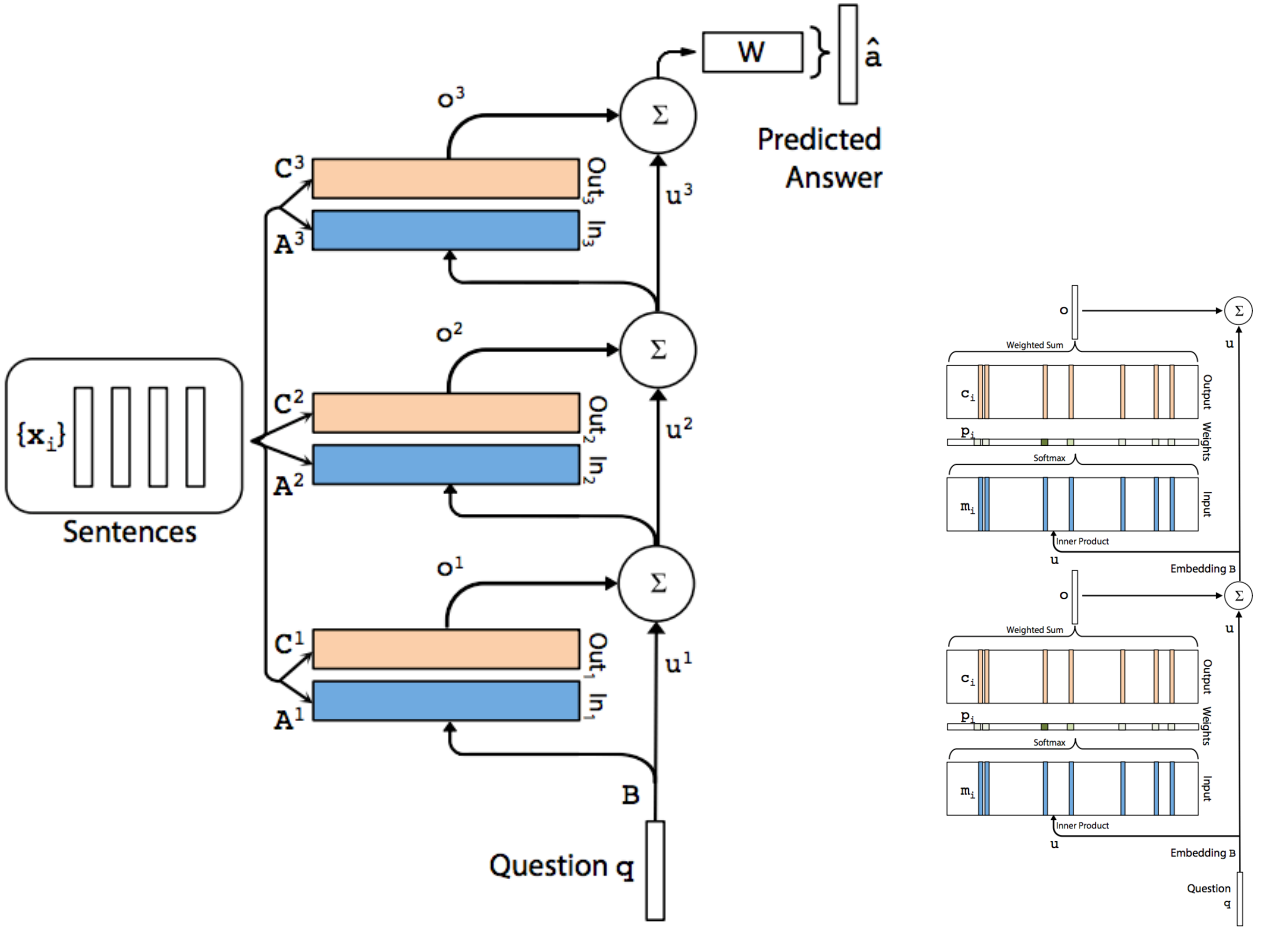


# Recurrent attention model with external memory

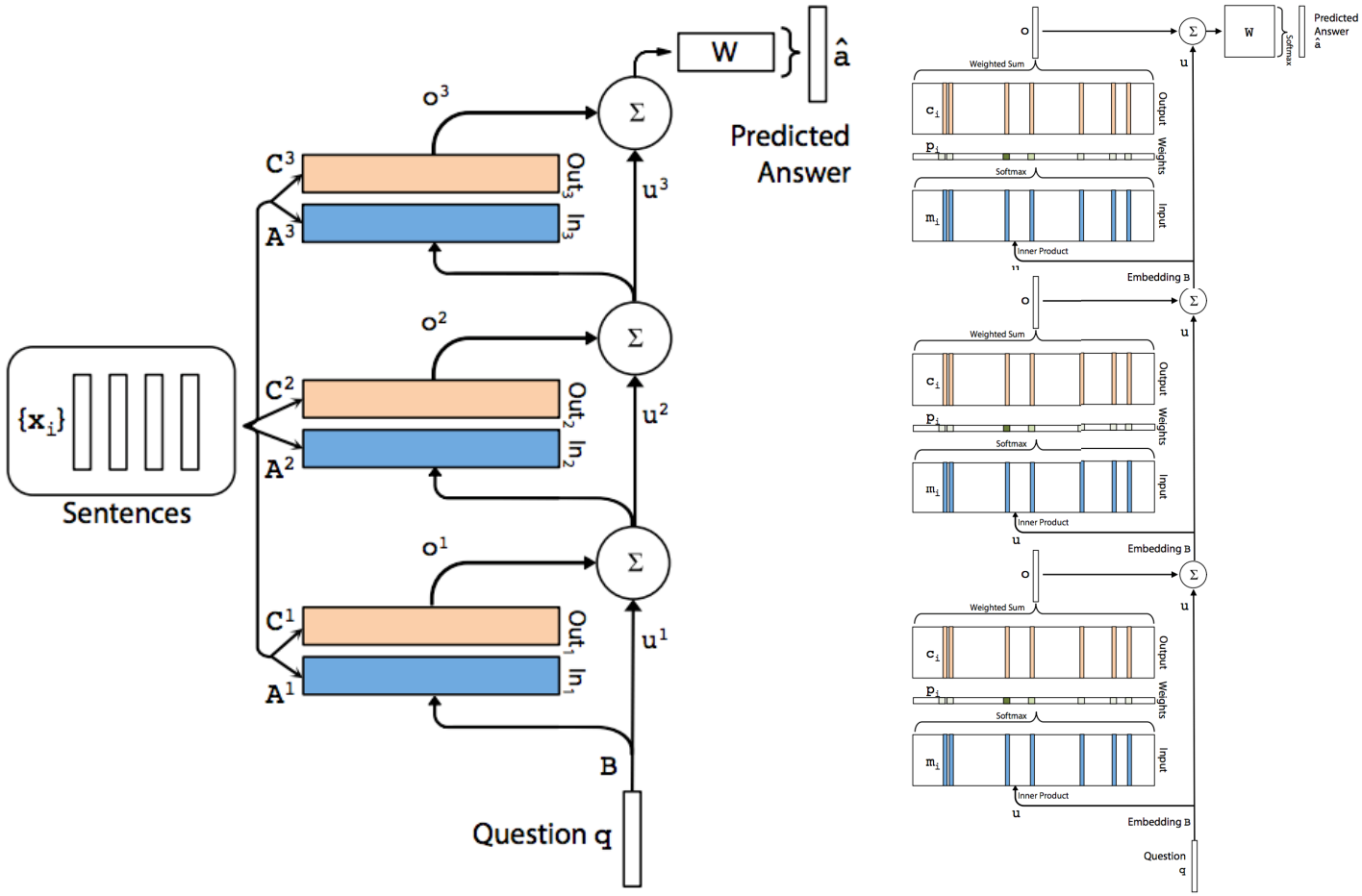


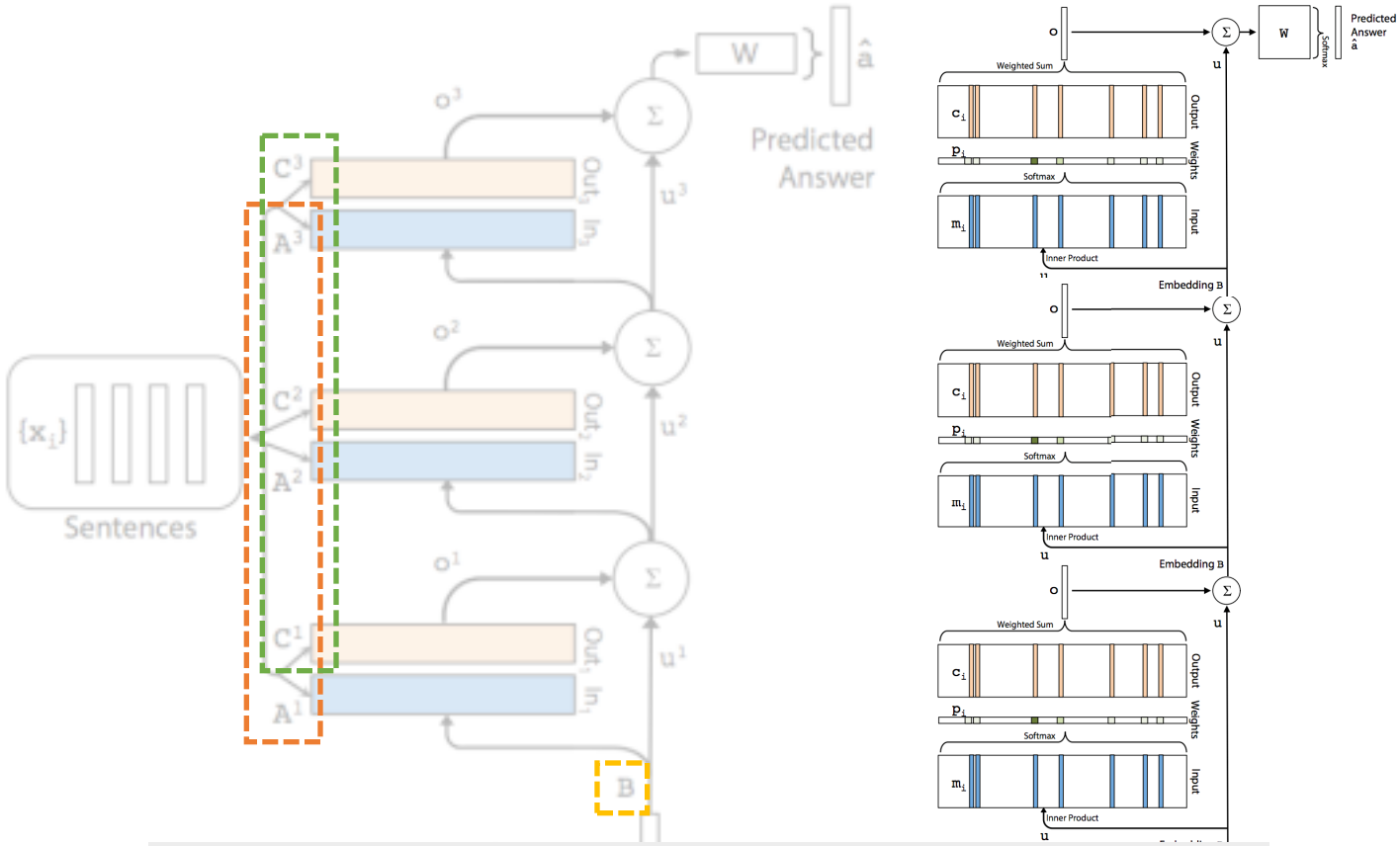
# Recurrent attention model with external memory



# Recurrent attention model with external memory



# Recurrent attention model with external memory




RNN 처럼 weight를 share할 수도 있고 안 할 수도 있다

```
import tensorflow as tf
```

# 참고

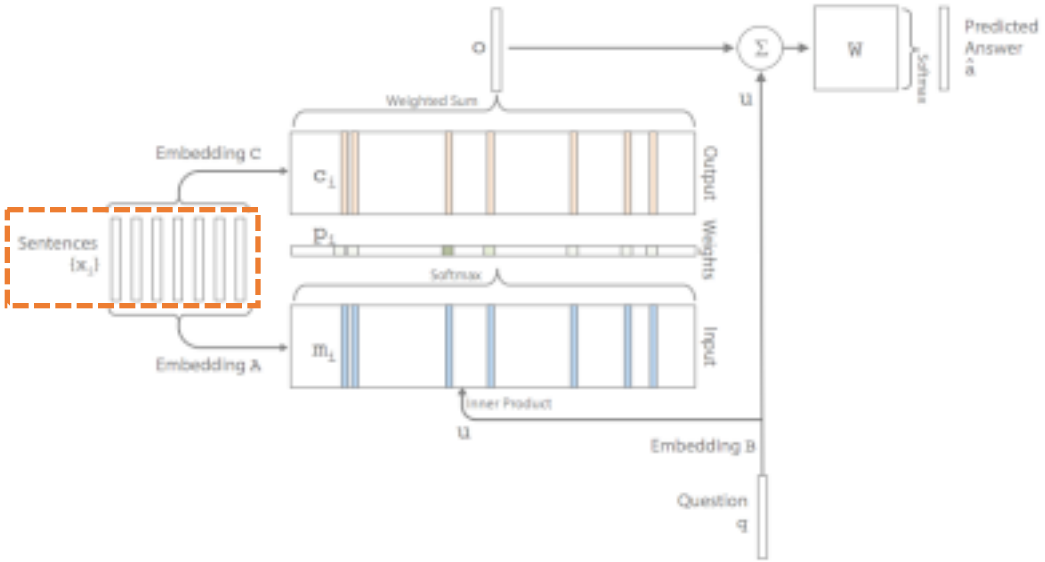
이 슬라이드의 코드는

**가독성**을 위해 많은 코드를 생략했습니다

(실제 코드와는 다릅니다)

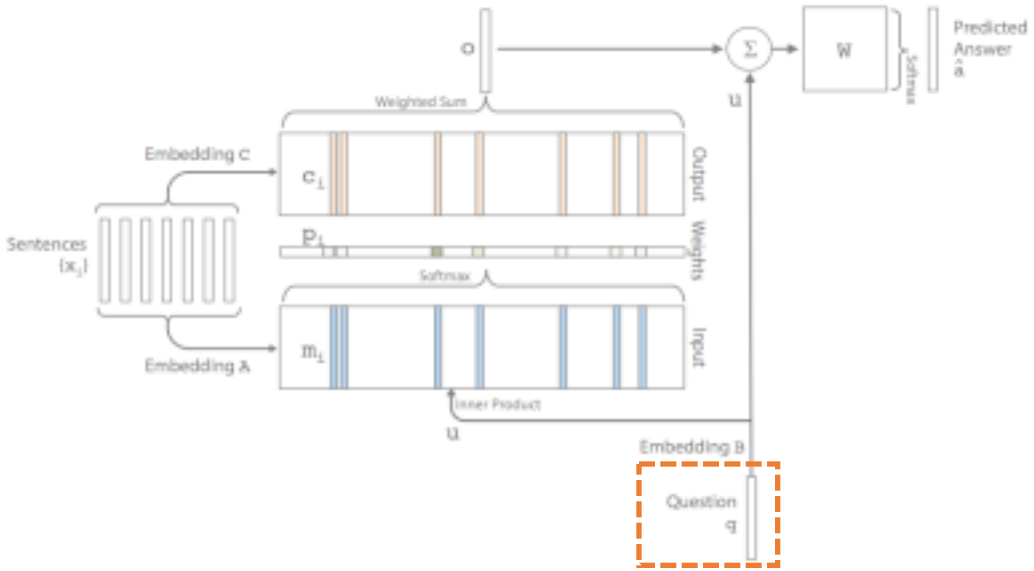
# Input: sentences $x_i$를 정의하고

```
sentences = tf.placeholder(tf.int32, [batch_size, mem_size])
```



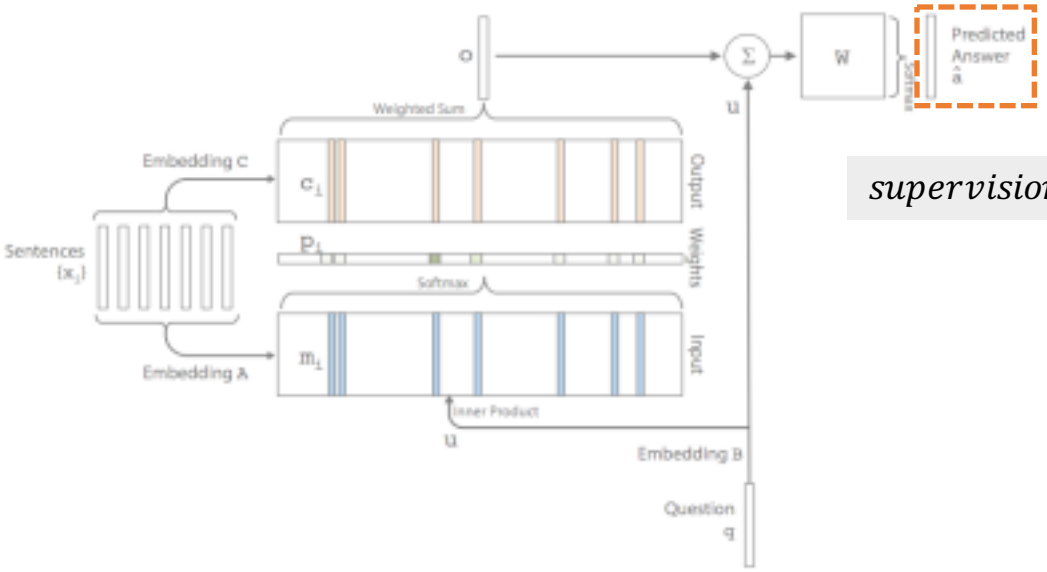
# Input: question $q$를 정의하고

```
sentences = tf.placeholder(tf.int32, [batch_size, mem_size])
question = tf.placeholder(tf.float32, [None, edim])
```



# Input: answer $\hat{a}$를 정의하고

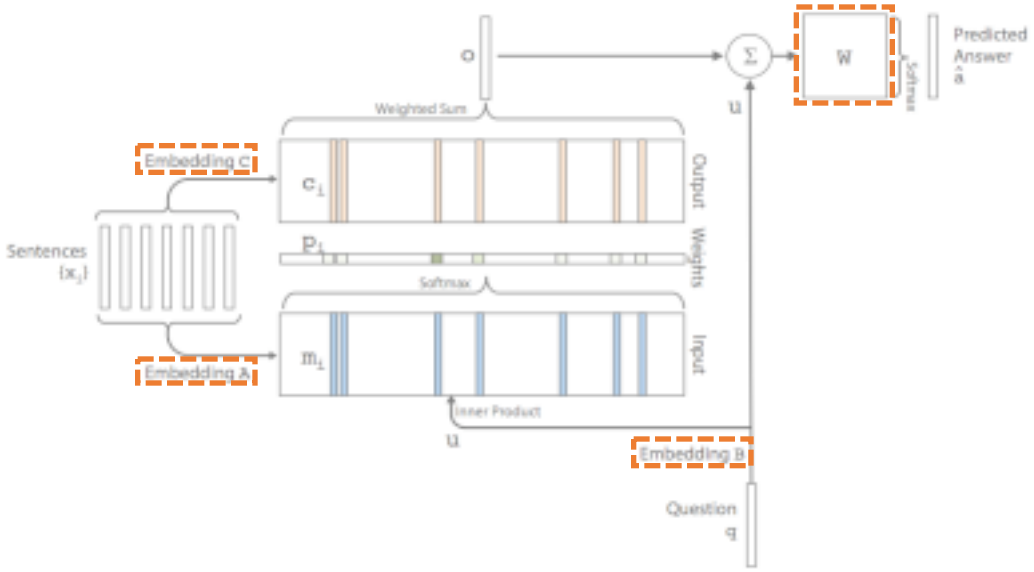
```
sentences = tf.placeholder(tf.int32, [batch_size, mem_size])
question = tf.placeholder(tf.float32, [None, edim])
answer = tf.placeholder(tf.float32, [batch_size, nwords])
```




*supervision*을 주기 위함

# Variable: 학습될 $A, B, C, W$를 정의한다

```
A = tf.Variable(tf.random_normal([nwords, edim], stddev=init_std))
B = tf.Variable(tf.random_normal([nwords, edim], stddev=init_std))
C = tf.Variable(tf.random_normal([edim, edim], stddev=init_std))
W = tf.Variable(tf.random_normal([edim, nwords], stddev=init_std))
```

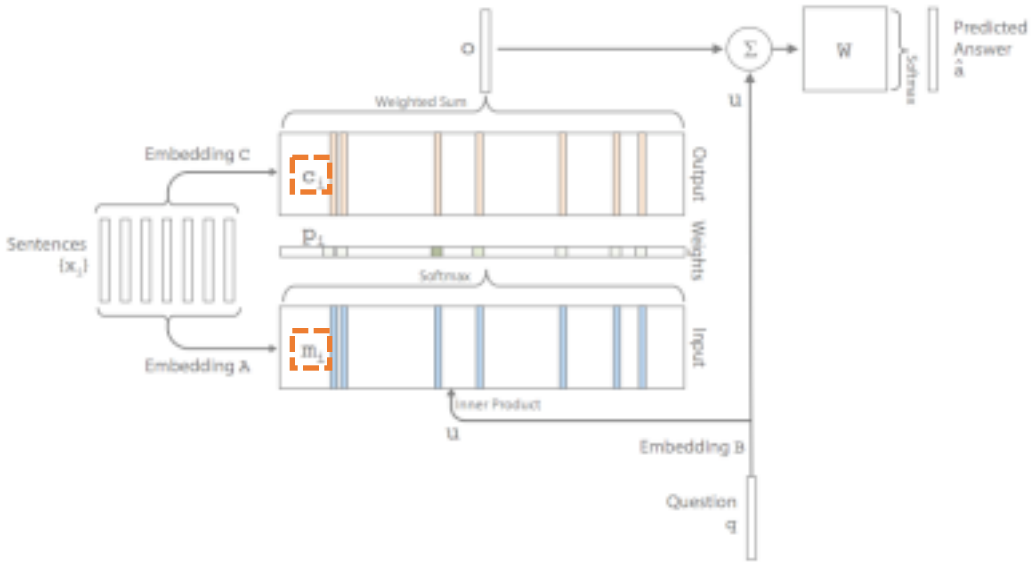


# Memory: $m_i$와 $c_i$를 계산하고

```
m_i = tf.nn.embedding_lookup(A, sentences)
c_i = tf.nn.embedding_lookup(C, sentences)
```

$$m_i = \sum_j Ax_{ij}$$

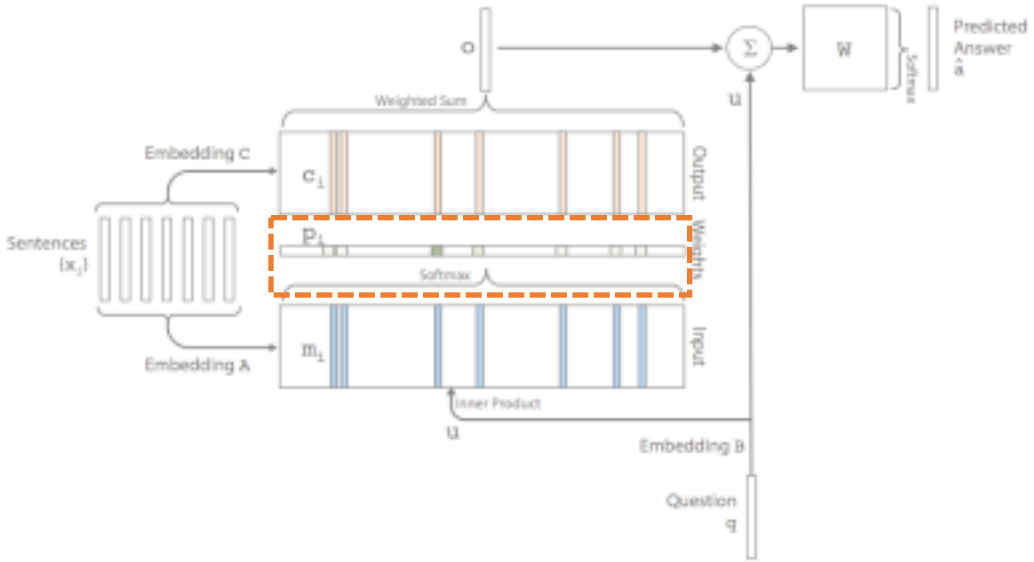
$$c_i = \sum_j Cx_{ij}$$



# Attention: $p_i$를 계산한다

```
Aout = tf.batch_matmul(u, m_i, adj_y=True)
p_i = tf.nn.softmax(Aout)
```

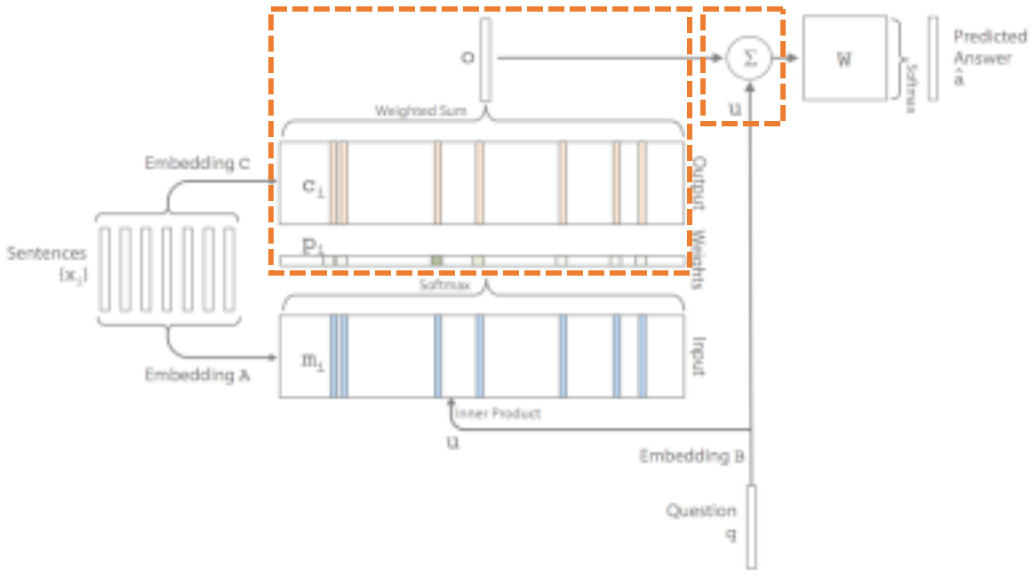
$$p_i = Softmax(u^T m_t)$$



# Output: $o$와 $o + u$를 계산하고

```
o = tf.batch_matmul(p_i, c_i) # weighted sum
Dout = o + u
```

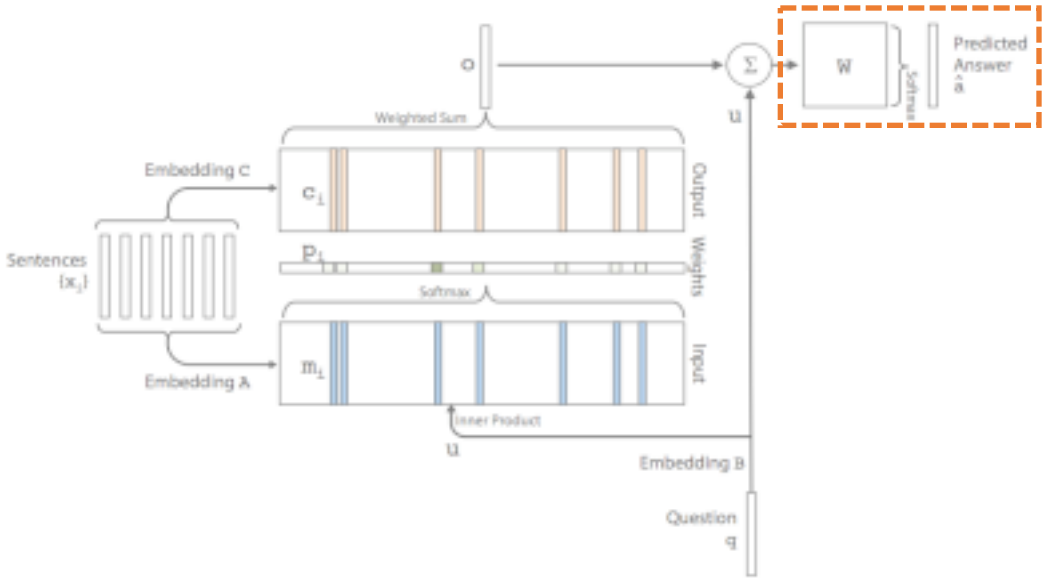
$$o = \sum_j p_i c_i$$



# Output: $\hat{a}$를 계산한다

```
z = tf.matmul(Dout, W)
a = tf.nn.softmax(z)
```

$$\hat{a}=\text{Softmax}(W(o+u))$$



# Optimization:
## *loss*와 *optimizer*를 정의한다

```
loss = tf.nn.softmax_cross_entropy_with_logits(z, answer)
lr = tf.Variable(current_lr, trainable=False)
opt = tf.train.GradientDescentOptimizer(lr)
```

sequence. A cross-entropy loss is used to train model by backpropagating the error through multiple memory layers, in the same manner as the QA tasks. To aid training, we apply ReLU operations to

# Tip: TensorFlow 에서 custom gradient를 정의하는 방법?

```
# optimizer를 정의한다
opt = tf.GradientDescentOptimizer(learning_rate=0.1)
```

# Tip: TensorFlow 에서 custom gradient를 정의하는 방법?

```
# optimizer를 정의한다
opt = tf.GradientDescentOptimizer(learning_rate=0.1)


# 주어진 variable의 loss에 대한 gradients를 계산한다.
grads_and_vars = opt.compute_gradients(loss, <list of variables>)
```

# Tip: TensorFlow 에서 custom gradient를 정의하는 방법?

```
# optimizer를 정의한다
opt = tf.GradientDescentOptimizer(learning_rate=0.1)

# 주어진 variable의 loss에 대한 gradients를 계산한다.
grads_and_vars = opt.compute_gradients(loss, <list of variables>)

# grads_and_vars은 tuple(gradient, variable)의 리스트다.
# 이제 gradient를 마음대로 바꾸고 optimizer를 적용(apply_gradient)하면 된다.
capped_grads_and_vars =
                    [(MyCapper(gv[0]), gv[1]) for gv in grads_and_vars]
optim = opt.apply_gradients(capped_grads_and_vars)
```

# Optimization:
# *graident clipping*을 한다

```
params = [A, B, C, W]
grads_and_vars = opt.compute_gradients(loss, params)
clipped_grads_and_vars = [(tf.clip_by_norm(gv[0], 50), gv[1])
                                                    for gv in grads_and_vars]
optim = opt.apply_gradients(clipped_grads_and_vars)
```

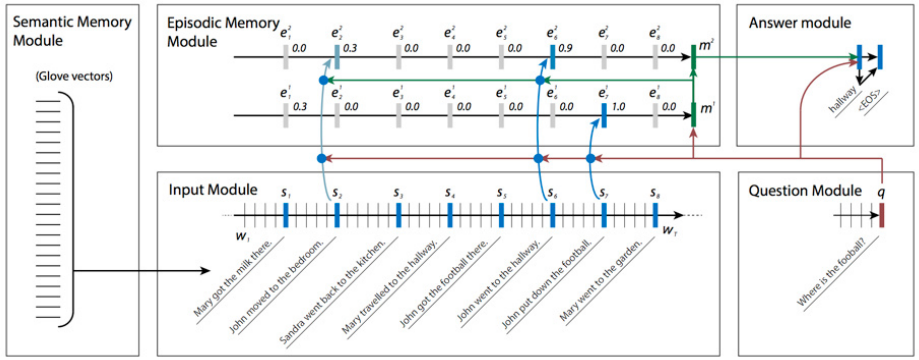
The training procedure we use is the same as the QA tasks, except for the following. For each mini-batch update, the $\ell_2$ norm of the whole gradient of all parameters is measured[2] and if larger than $L = 50$, then it is scaled down to have norm $L$. This was crucial for good performance. We

# Train: 학습 시키면 끝

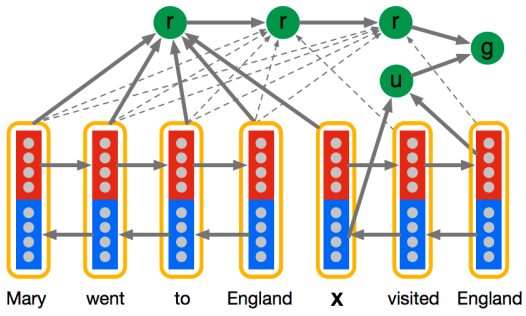
```
for idx in xrange(N):
  sess.run(optim, feed_dict={input: x,
                             target: target,
                             context: context})
```

# Resources

- Paper “End-To-End Memory Networks”
- 저자의 코드 : https://github.com/facebook/MemNN
- Question Answering에 더 관심이 있다면
  - Paper “Ask Me Anything: Dynamic Memory Networks for Natural Language Processing”
  - Paper “Teaching machines to read and comprehend”




Dynamic Memory Networks




Attentive Impatient Reader

# QA dataset: The bAbI tasks

> AI가 문장들에서 **정보를 추론**할 수 있을까?

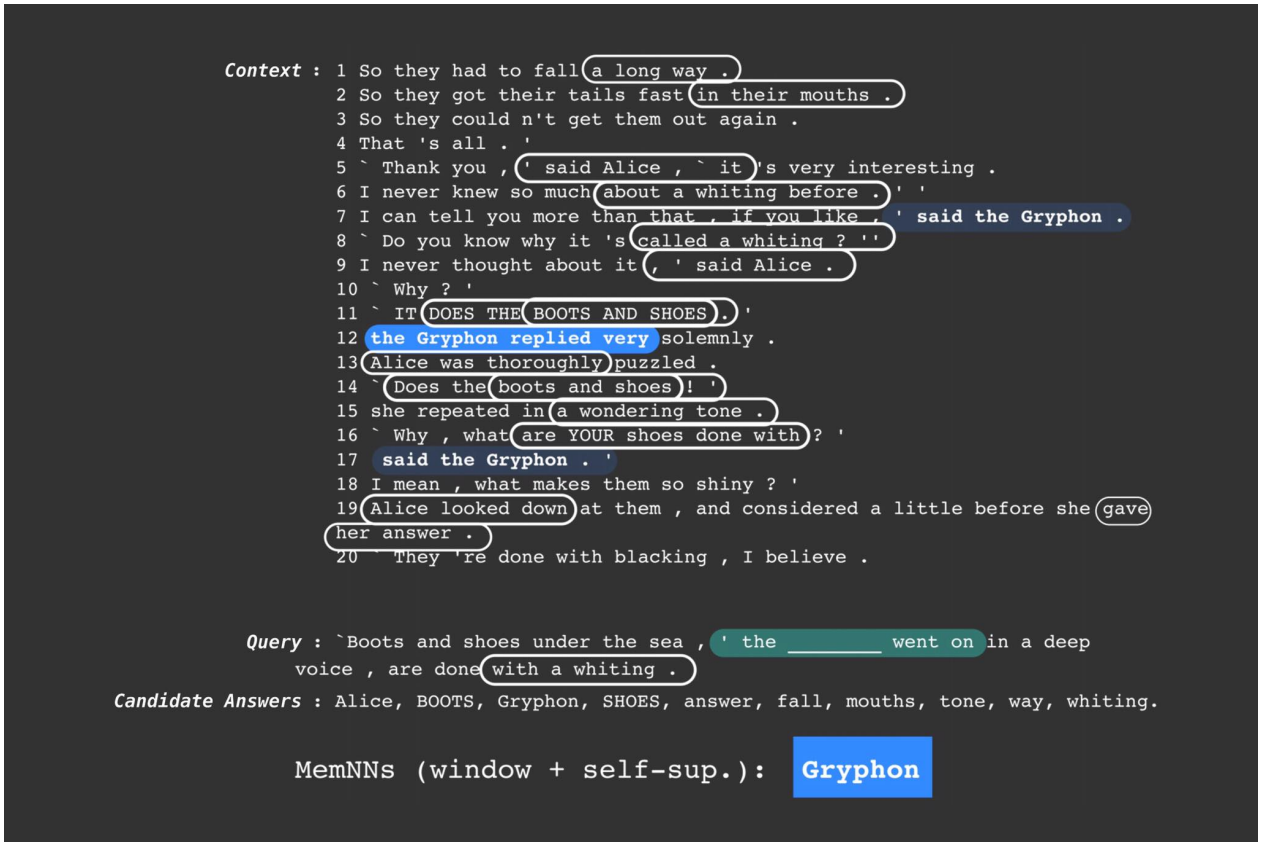
**Question:** Where was Mary before the Bedroom?
**Answer:** Cinema.

| Facts | Episode 1 | Episode 2 | Episode 3 |
|---|---|---|---|
| Yesterday Julie traveled to the school. | | | |
| Yesterday Marie went to the cinema. | | ■ | |
| This morning Julie traveled to the kitchen. | | | |
| Bill went back to the cinema yesterday. | | | |
| Mary went to the bedroom this morning. | ■ | | |
| Julie went back to the bedroom this afternoon. | | | |
| [done reading] | | | ■ |

https://research.facebook.com/research/babi

# QA dataset: The Children's Book Test

> AI가 "이상한 나라의 앨리스"를 이해할 수 있을까?




https://research.facebook.com/research/babi/#cbt

# Asynchronous Methods for Deep Reinforcement Learning

핵심: Threading, Custom gradients, `partial_run()`

# Asynchronous Methods for Deep Reinforcement Learning (A3C)

**Algorithm 3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

*// Assume global shared parameter vectors* $\theta$ *and* $\theta_v$ *and global shared counter* $T = 0$
*// Assume thread-specific parameter vectors* $\theta'$ *and* $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
**repeat**
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state $s_t$
  **repeat**
    Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
    Receive reward $r_t$ and new state $s_{t+1}$
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  **until** terminal $s_t$ **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  **for** $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial\,(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  **end for**
  Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
**until** $T > T_{max}$

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Kavukcuoglu, K. (ICML 2016)

https://github.com/carpedm20/deep-rl-tensorflow

# 해결한 문제

Reinforcement Learning

# 해결한 문제

## Reinforcement Learning

1. Atari 2600 Games


Atari

# 해결한 문제

## Reinforcement Learning

1. Atari 2600 Games
2. TORCS Car Racing, MuJoCo


Atari

TORCS

# 해결한 문제

## Reinforcement Learning

1. Atari 2600 Games
2. TORCS Car Racing, MuJoCo
3. Labyrinth


Atari

TORCS

Labyrinth

# 한 문장 정리

Asynchronous

n-step actor-critic

# 한 문장 정리

Asynchronous

n-step actor-critic

1. Asynchronous one-step Q-learning
2. Asynchronous one-step Sarsa
3. Asynchronous n-step Q-learning
4. **Asynchronous advantage actor-critic**

# 필요한 사전 지식

Asynchronous

Asynchronous q-learning

n-step actor-critic

Actor-critic methods

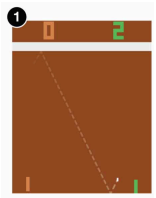
n-step TD Prediction

# WARNING

이 논문은 **많은 사전 지식**을 필요로 하기때문에

전체적인 흐름 파악에만 **집중**해 주세요

# 정의: State, Value

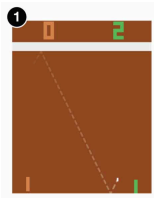


State: game의 **상태**

화면, 공의 위치, 상대방의 위치 …

Demystifying Deep Reinforcement Learning

# 정의: State, Value



State: game의 **상태** 화면, 공의 위치, 상대방의 위치 …

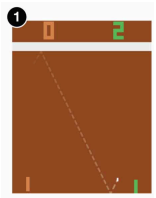
Value: state $s_t$에서 행동 $a_t$의 **가치** 보상 (return)

$a_t$를 하고 난 후의 점수 $r_{t+1}$뿐만 아니라

$s_t, s_{t+1}, s_{t+2}, \ldots$에서 받게되는 미래의 모든 점수, **즉** $r_{t+1} + r_{t+2} + r_{t+3} + \cdots$의 기대값

Demystifying Deep Reinforcement Learning

# 정의: State, Value



State: game의 상태 화면, 공의 위치, 상대방의 위치 …

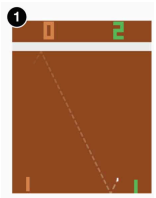
Value: state $s_t$에서 행동 $a_t$의 가치 보상 (return)

Value가 중요한 이유는 Value를 통해 어떠한 행동을 할지 결정할 수 있기 때문

epsilon greedy policy

Demystifying Deep Reinforcement Learning

# 정의: State, Value



State: game의 상태 화면, 공의 위치, 상대방의 위치 …

Value: state $s_t$에서 행동 $a_t$의 가치 보상 (return)

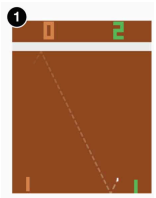
| State | $Q(s_t, \text{위})$ | $Q(s_t, \text{아래})$ |
|---|---|---|
| $s_1$ | 1.2 | -1 |
| $s_2$ | -1 | 0 |
| $s_3$ | 2.3 | 1.2 |
| $s_4$ | 0 | 0 |
| . | . | . |
| . | . | . |
| . | . | . |
| $s_N$ | 0.2 | -0.1 |

가장 쉬운 방법은 표로 기록하고 값을 업데이트 하는것!

Dynamic programming, Value iteration …

Demystifying Deep Reinforcement Learning

# 정의: State, Value



State: game의 상태 화면, 공의 위치, 상대방의 위치 …

Value: state $s_t$에서 행동 $a_t$의 가치 보상 (return)

| State | $Q(s_t, 위)$ | $Q(s_t, 아래)$ |
|---|---|---|
| $s_1$ | 1.2 | -1 |
| $s_2$ | -1 | 0 |
| $s_3$ | 2.3 | 1.2 |
| $s_4$ | 0 | 0 |
| . | . | . |
| . | . | . |
| . | . | . |
| $s_N$ | 0.2 | -0.1 |

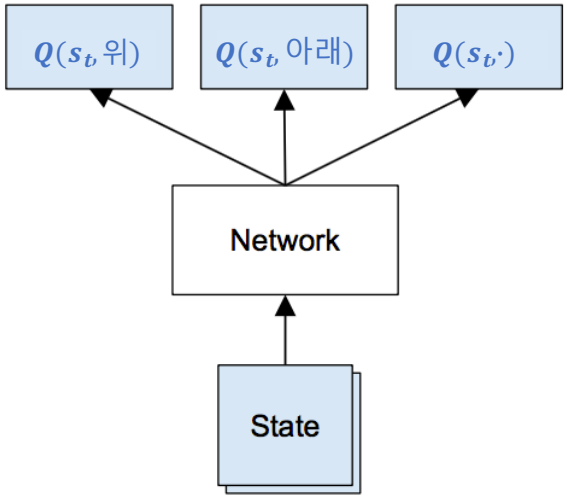
하지만 state가 너무 많을 경우 각각을 하나하나 계산하는게 불가능

Demystifying Deep Reinforcement Learning

# Model: Deep Q (quality) -Network

$Q(s_t)$를 함수로 approximate 하는것 :

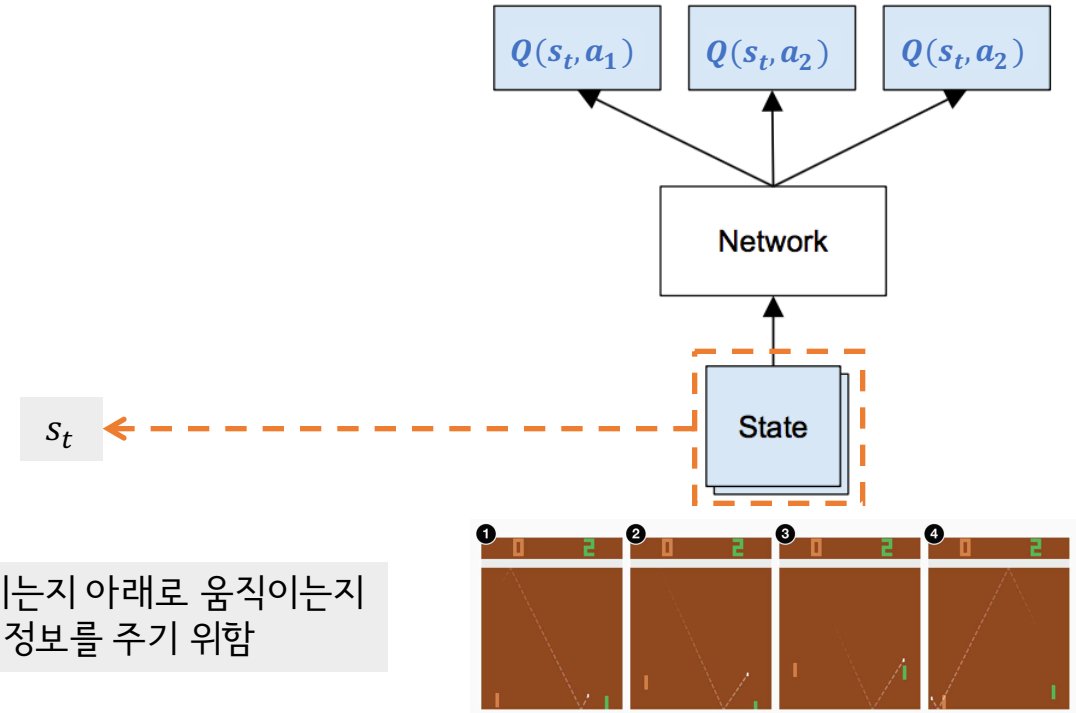
Value function approximation




딥러닝 모델로 한번 배워보자!

Demystifying Deep Reinforcement Learning

# Input: 4개의 연속된 프레임

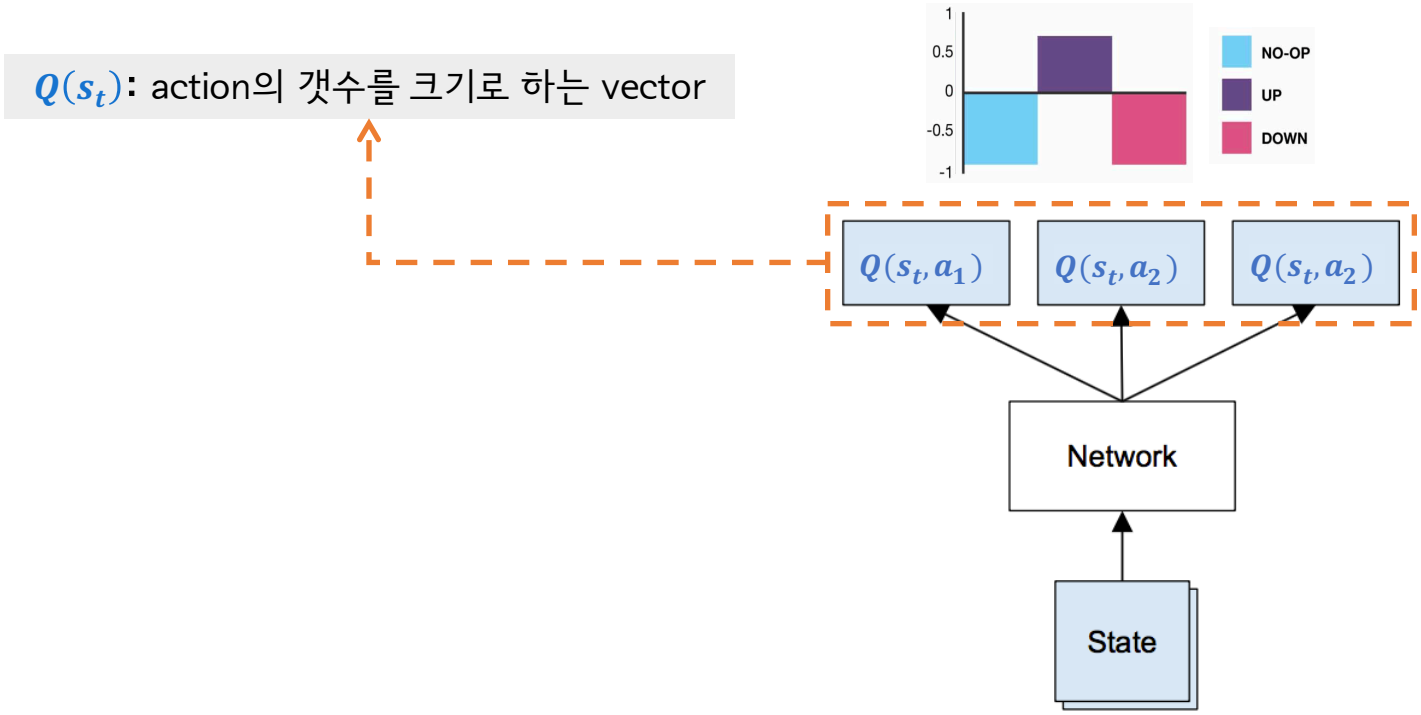



공이 위로 움직이는지 아래로 움직이는지
context 정보를 주기 위함

Human-level control through deep reinforcement learning

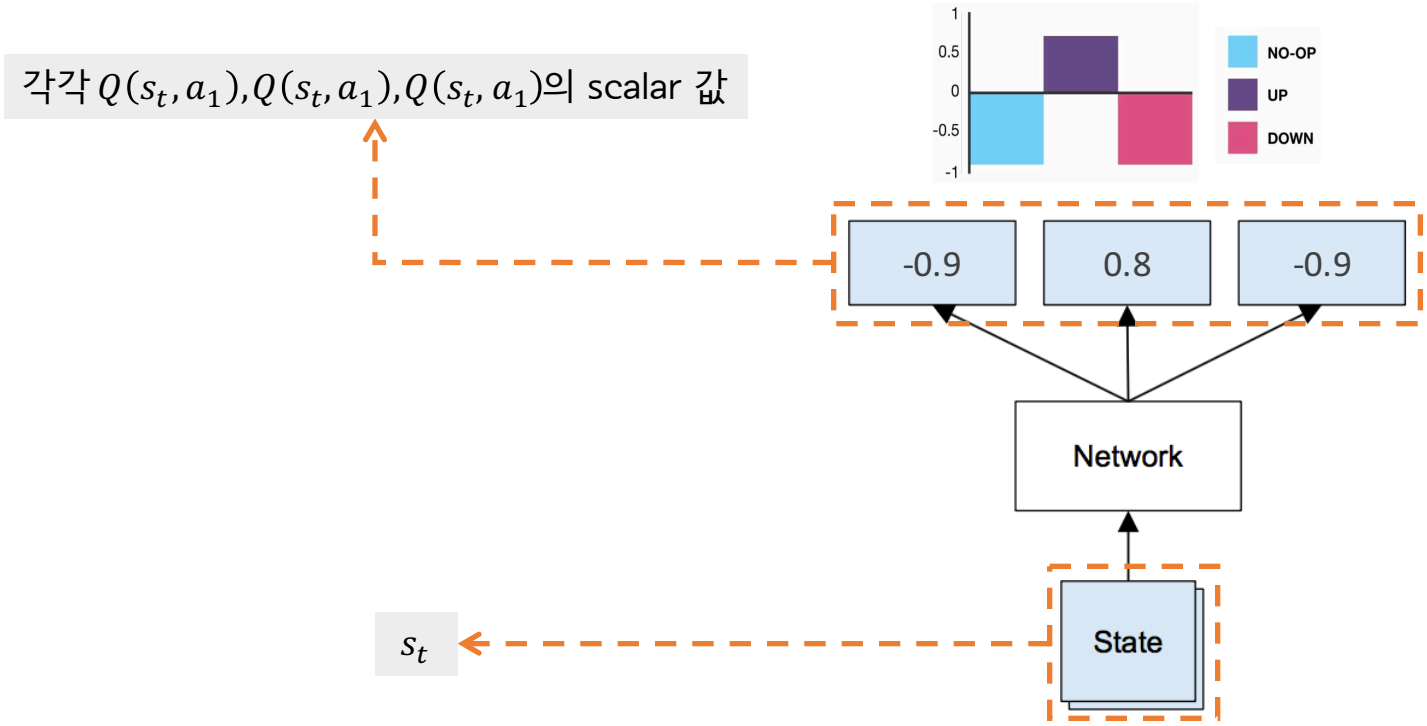
# Output: 행동의 가치(q-value)

$Q(s_t)$: action의 갯수를 크기로 하는 vector



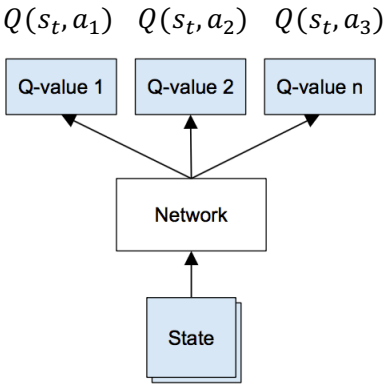

Human-level control through deep reinforcement learning

# Output: 행동의 가치(q-value)




Human-level control through deep reinforcement learning

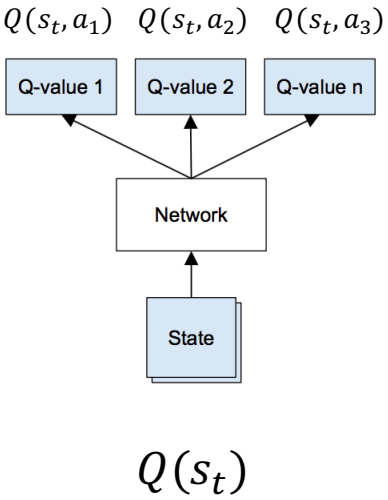
# Training: Q-learning




$Q(s_t)$

See David Silver's course & Sutton's book for more information

# Training: Q-learning




$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

See David Silver's course & Sutton's book for more information

# $s_t$가 있을때

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$



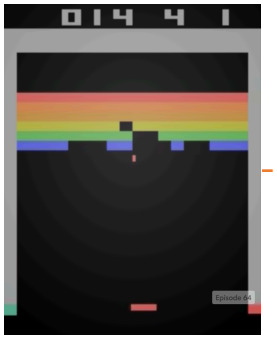

$Q(s_t)$


$s_t$

# 행동 $a$를 해서 reward $r_{t+1}$을 받으면

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$




$s_t$

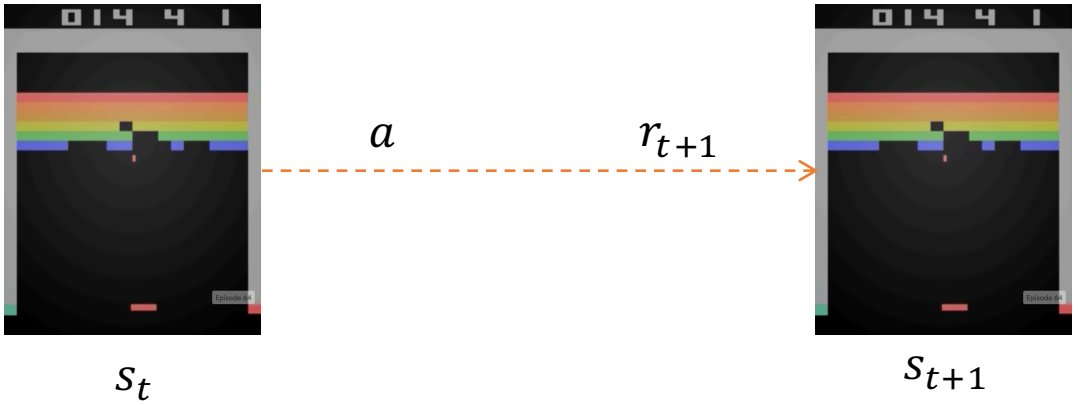
$a$

$r_{t+1}$

# 새로운 state $s_{t+1}$가 생긴다

$$Q(s_t, a) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$



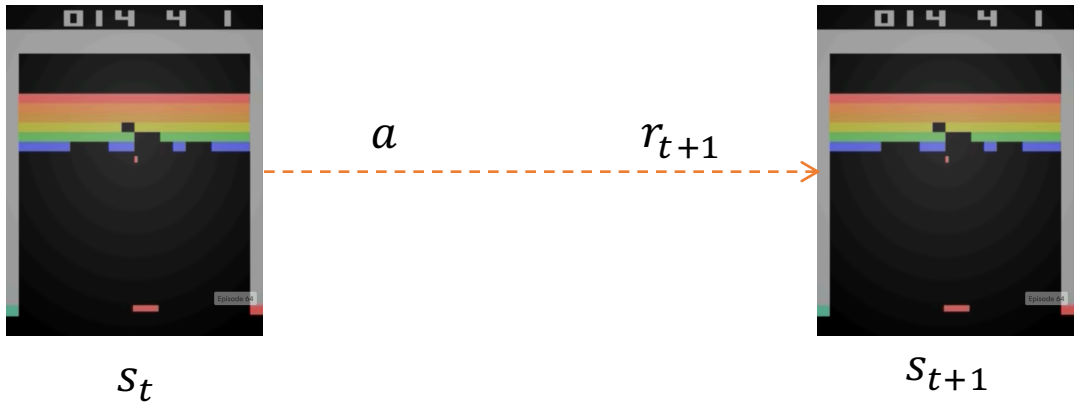
# Q-network는

return, value

## 각 행동이 그 state에서 창출할 **가치**를 예상하며

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$




예상하는 가치 : $Q(s_t)$ $Q(s_{t+1})$

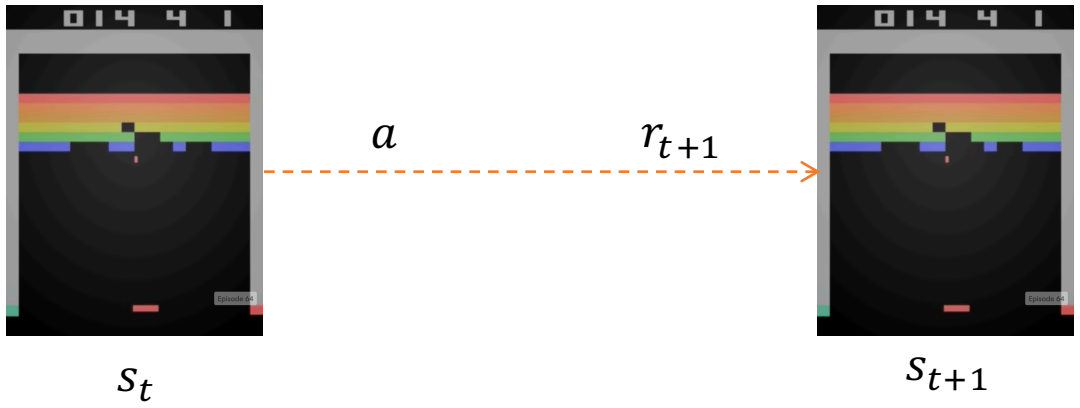
# Return은

expectation

## 미래에 받을 reward 합의 **기대값**이다

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$




예상하는 가치 : $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$

$Q(s_t)$

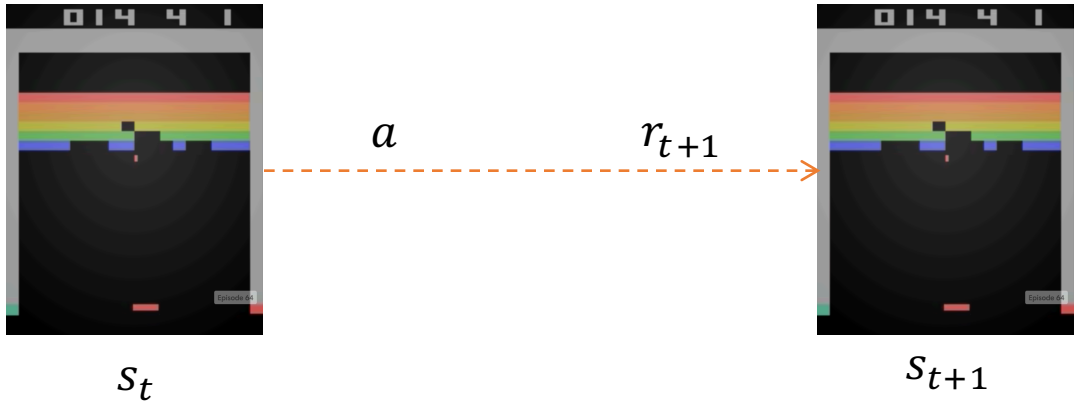
$r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots$

$Q(s_{t+1})$

# $Q(s_t) = r_{t+1} + \gamma Q(s_{t+1})$가 되어야 하며

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$



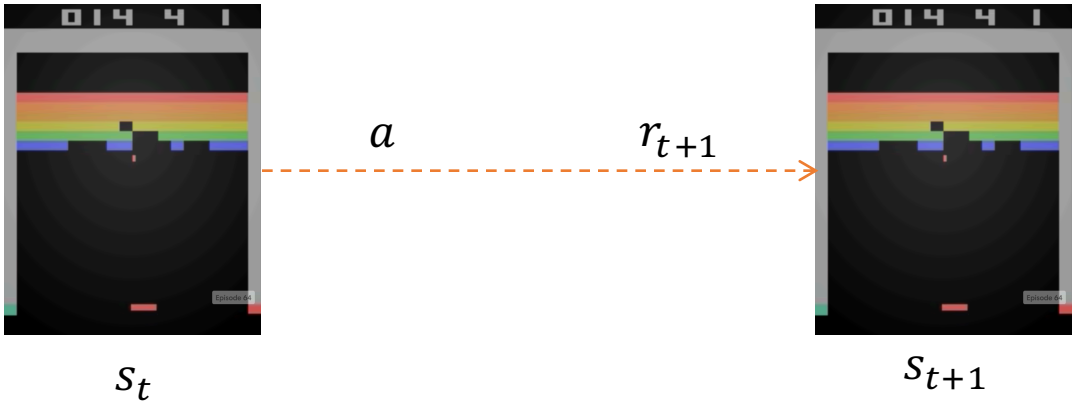

예상하는 가치 : $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$ $r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots$

$Q(s_t)$ $\boxed{Q(s_t) = r_{t+1} + \gamma Q(s_{t+1})}$ $Q(s_{t+1})$

# $loss$는 둘의 **차이**가 된다

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$




예상하는 가치 : $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$ $r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots$

$Q(s_t)$ $\boxed{Q(s_t) = r_{t+1} + \gamma Q(s_{t+1})}$ $Q(s_{t+1})$

$$\boxed{loss = \{r_{t+1} + \gamma Q(s_{t+1})\} - Q(s_t)}$$

# 즉, loss만큼 $Q(s_t)$를 업데이트 한다

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$




$Q(s_t)$



$s_t$

$a$

$r_{t+1}$



$s_{t+1}$

예상하는 가치 : $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$

$r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots$

$$loss = \{r_{t+1} + \gamma Q(s_{t+1})\} - Q(s_t)$$

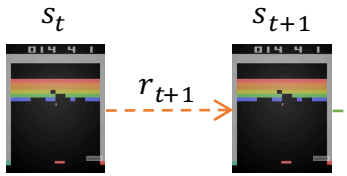
# n-step Q-learning




$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

업데이트

1-step




TD (1-step)

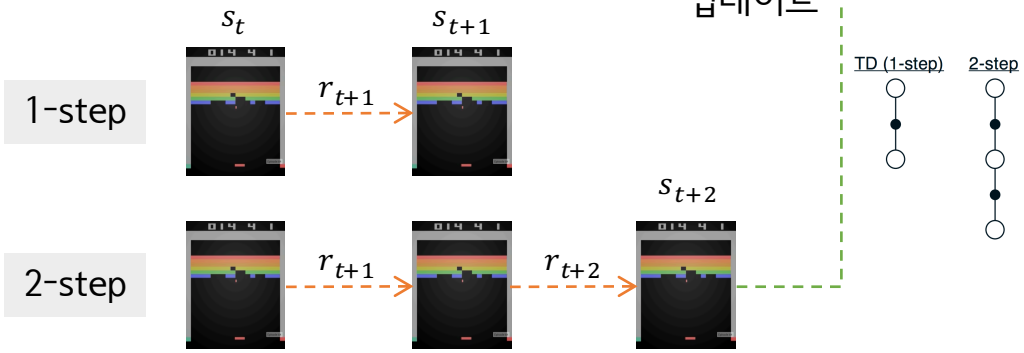
See David Silver's course & Sutton's book for more information

# n-step Q-learning




$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

업데이트



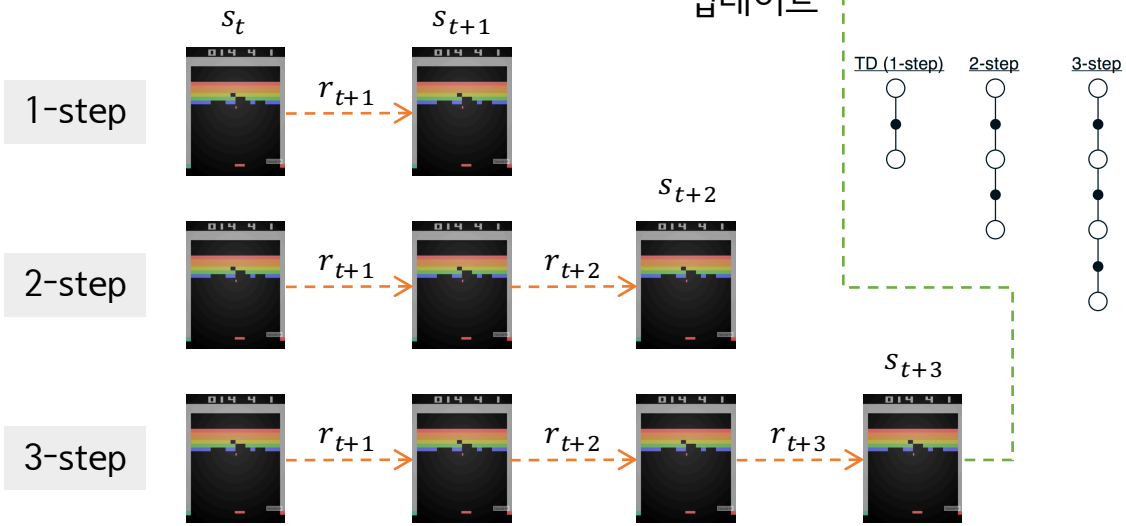

See David Silver's course & Sutton's book for more information

# n-step Q-learning




$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

업데이트




See David Silver's course & Sutton's book for more information

# n-step Q-learning




$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

업데이트

1-step $\{r_{t+1} + \gamma Q(s_{t+1})\} - Q(s_t)$

TD (1-step) 2-step 3-step *n*-step

2-step $s_{t+2}$ $r_{t+1}$ $r_{t+2}$

3-step $s_{t+3}$ $r_{t+1}$ $r_{t+2}$ $r_{t+3}$

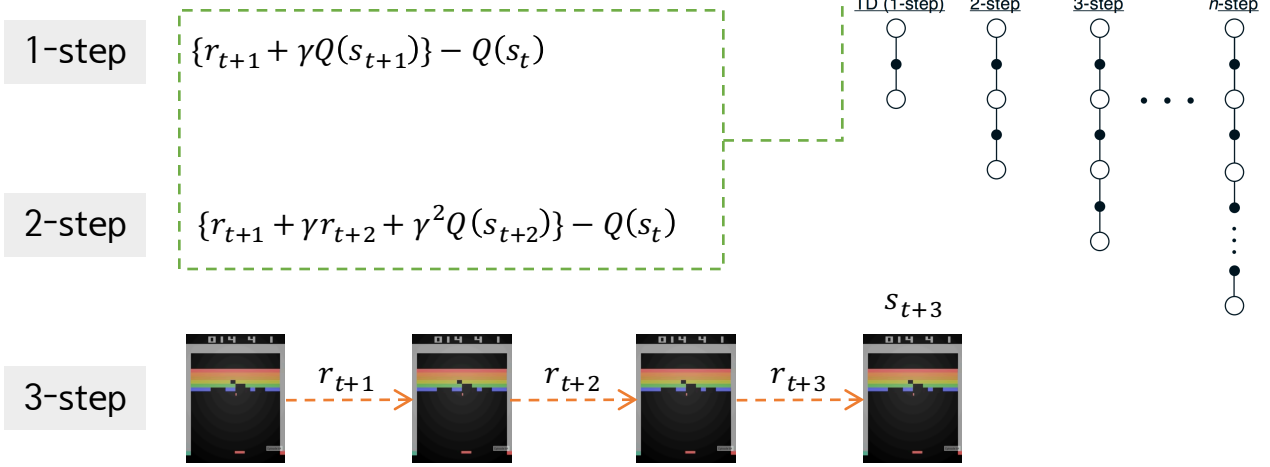
See David Silver's course & Sutton's book for more information

# n-step Q-learning

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

업데이트




See David Silver's course & Sutton's book for more information

# n-step Q-learning



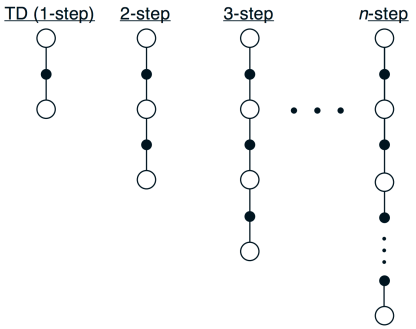

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

업데이트

1-step $\{r_{t+1} + \gamma Q(s_{t+1})\} - Q(s_t)$

2-step $\{r_{t+1} + \gamma r_{t+2} + \gamma^2 Q(s_{t+2})\} - Q(s_t)$

3-step $\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 Q(s_{t+3})\} - Q(s_t)$




Higher variance, lower bias

See David Silver's course & Sutton's book for more information

# n-step Q-learning

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big( \overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)$$

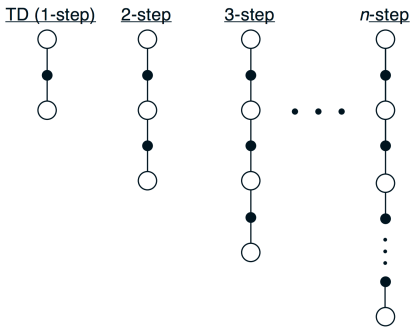
업데이트




1-step $\{r_{t+1} + \gamma Q`(s_{t+3})\} - Q(s_{t+2})$

2-step $\{r_{t+1} + \gamma r_{t+2} + \gamma^2 Q`(s_{t+3})\} - Q(s_{t+1})$

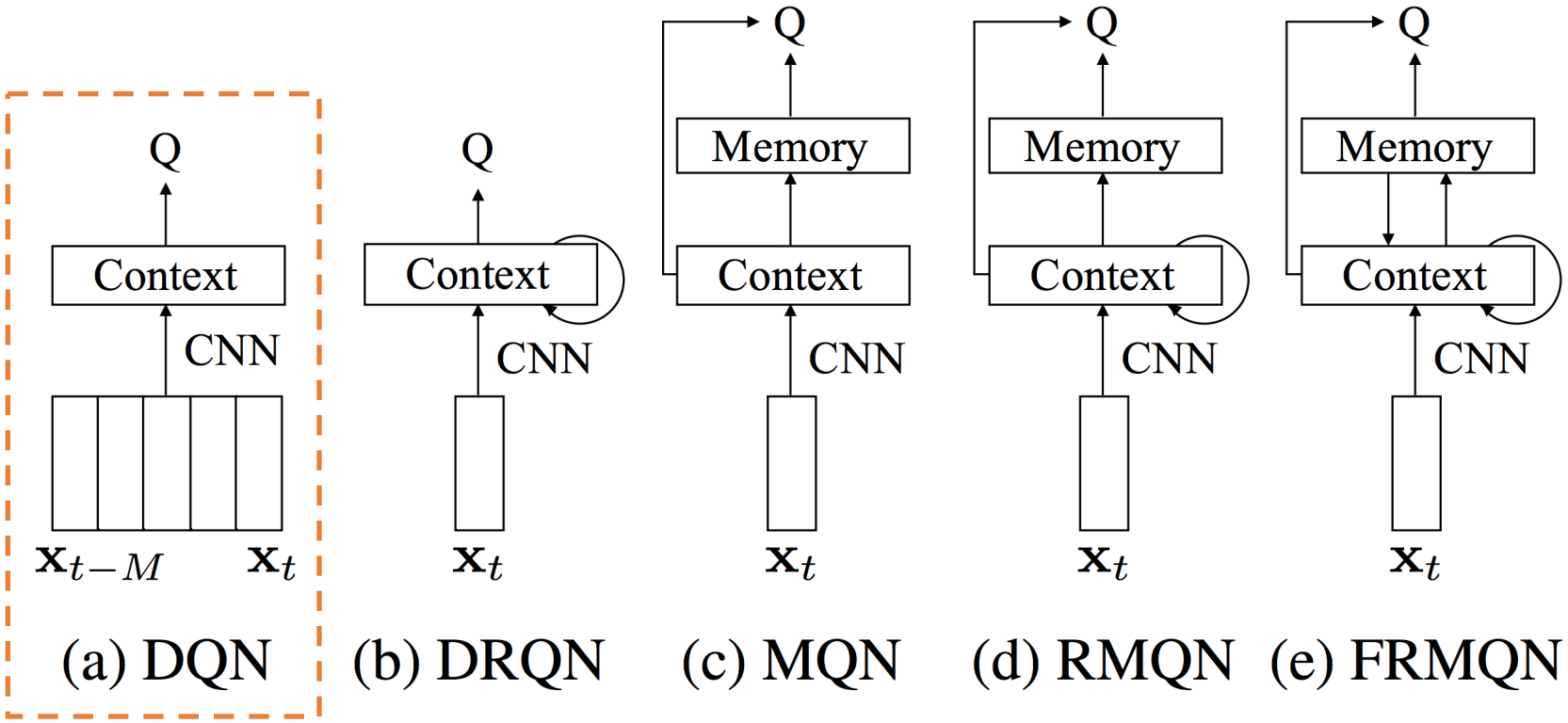
3-step $\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 Q`(s_{t+3})\} - Q(s_t)$




$Q`$ : target network, $Q$ : prediction network

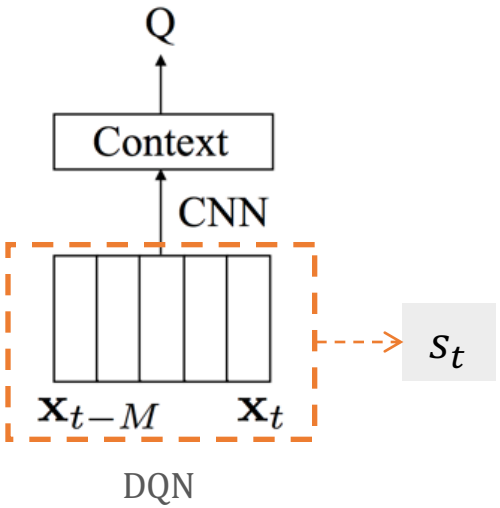
Read the paper

# 다양한 Q-network




Control of Memory, Active Perception, and Action in Minecraft
Oh, J., Chockalingam, V., Singh, S., & Lee, H. (2016).

# Deep Q-Network (DQN)

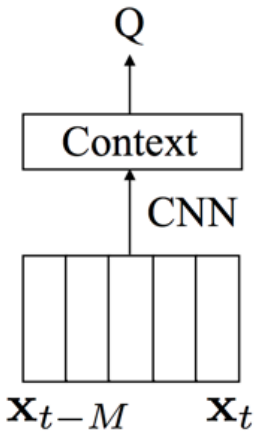



NIPS 2013

# Asynchronous n-step actor-critic

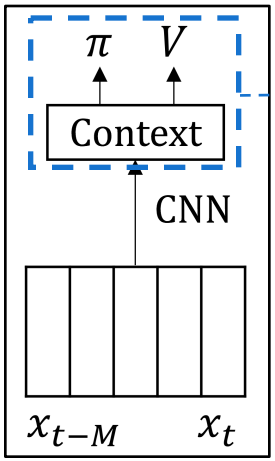
행동의 확률 : $\pi$ : actor

state의 가치 : V : critic




DQN

NIPS 2013



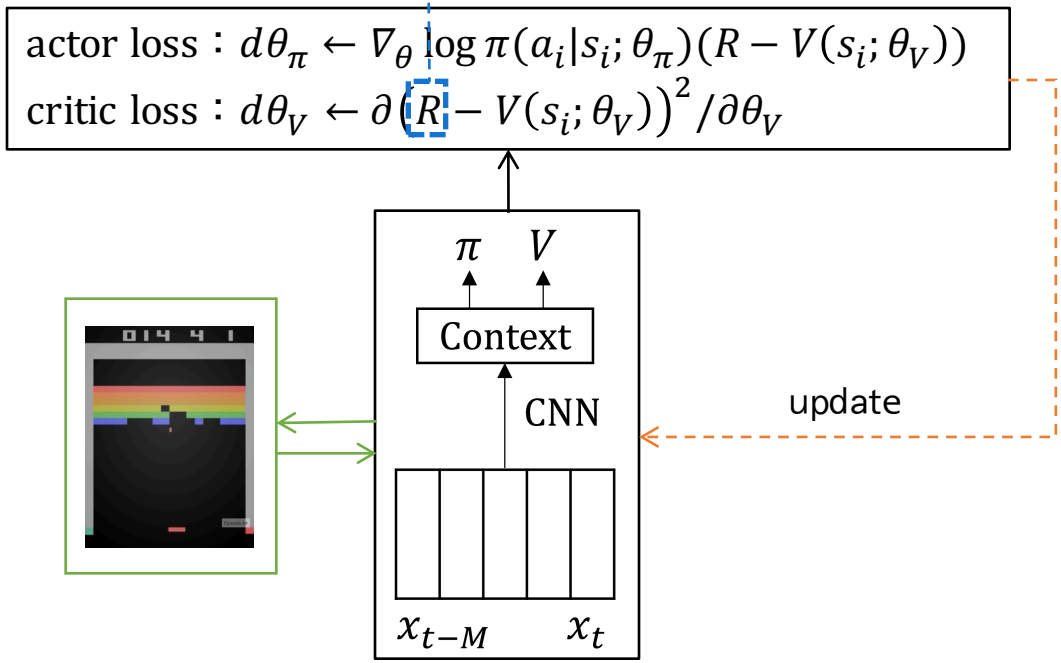

두개의 FC layers

actor-critic (ac)

ICML 2016

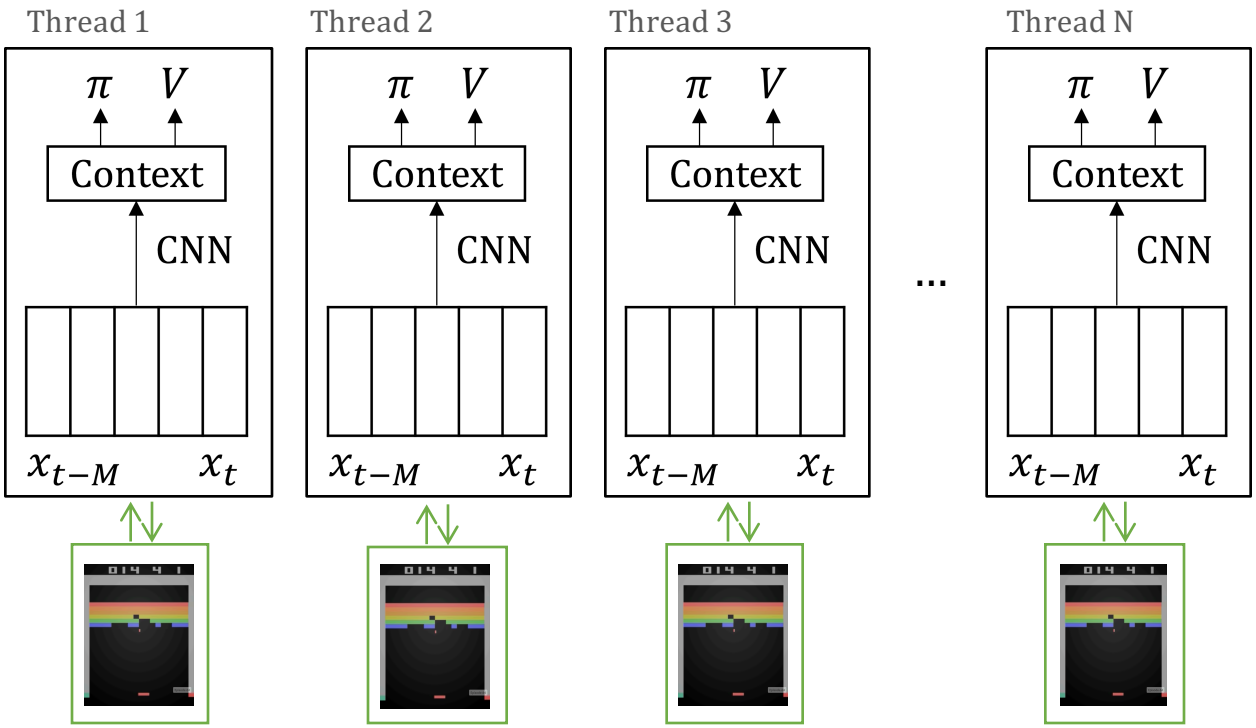
# Single thread actor-critic worker

$\{r_{t+1} + \gamma Q(s_{t+1})\} - Q(s_t)$ $\{r_{t+1} + \gamma r_{t+2} + \gamma^2 Q(s_{t+1})\} - Q(s_t)$ $\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 Q(s_{t+1})\} - Q(s_t)$




하나의 actor-critic network으로 모델을 학습

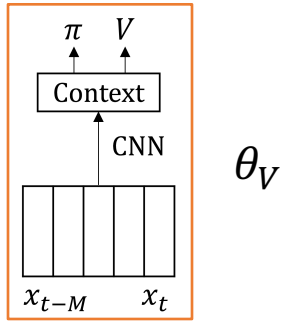
# Multi-thread actor-critic workers



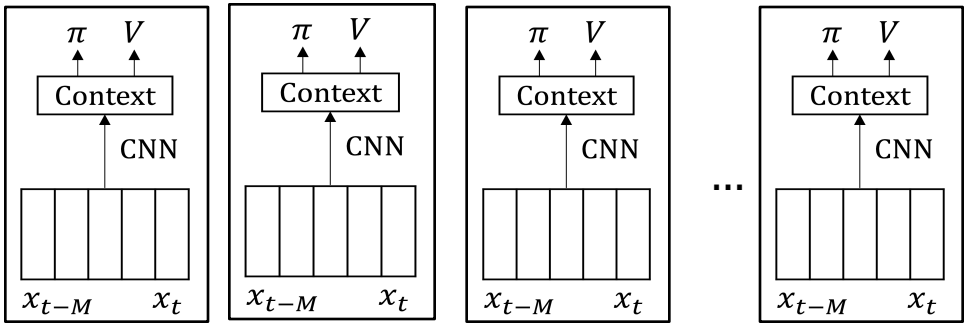

각 thread 별로 다른 게임을 하며 학습

Bach로 학습한다고 이해하면 쉽다

# Global actor-critic Network



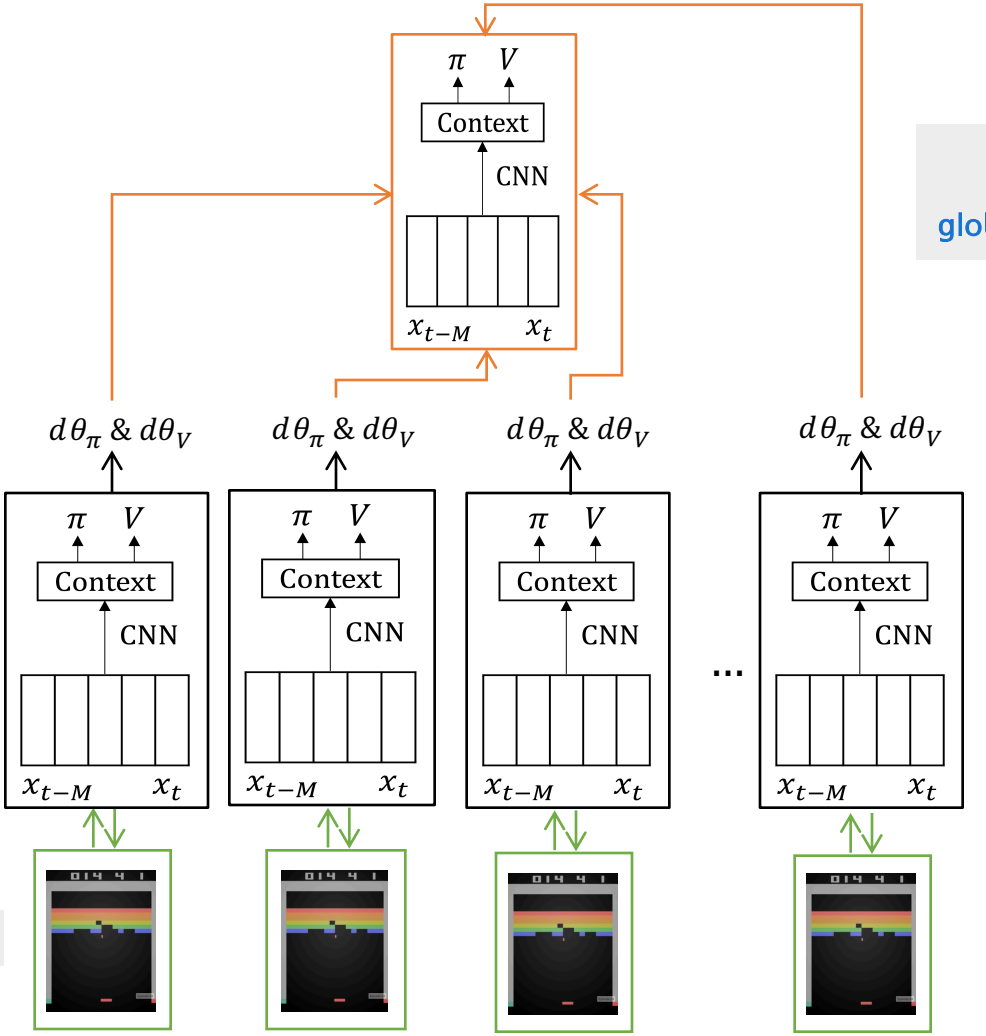

$\theta_V$




$\theta`_V$

global weight(parameter)를 관리하는 Global Q-Network

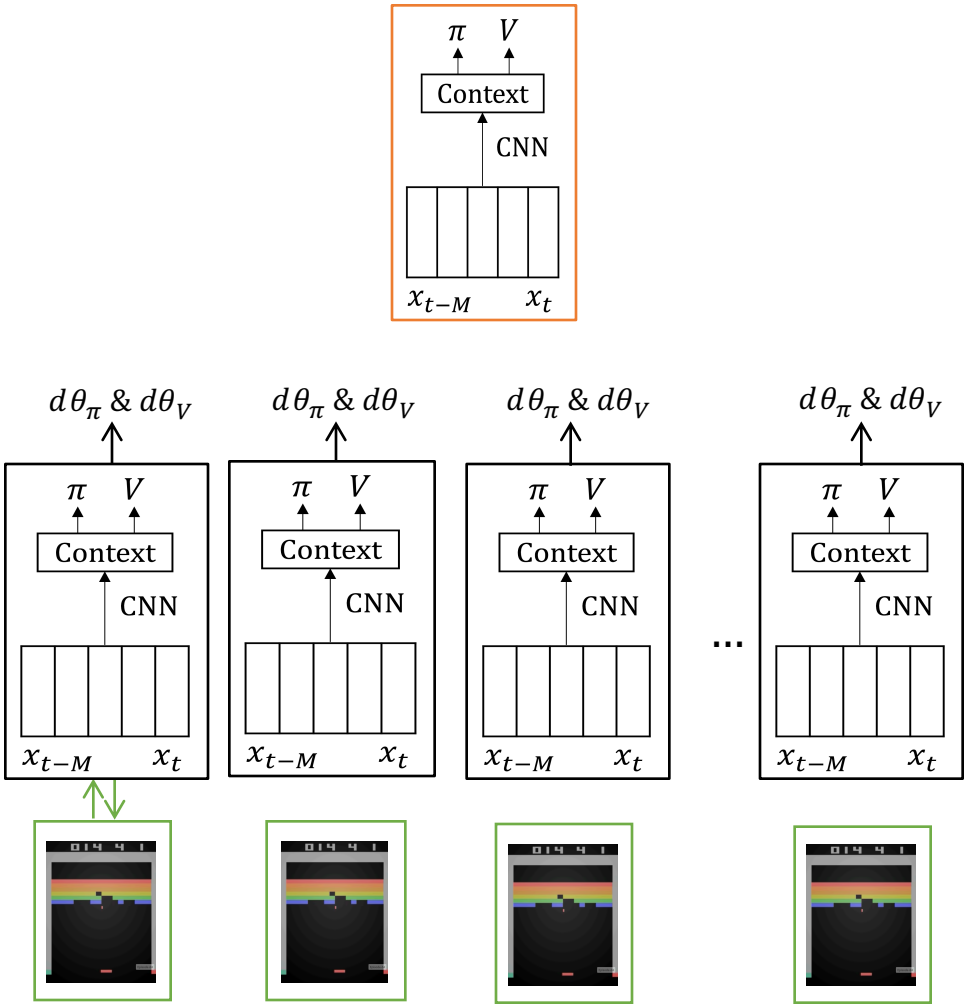
# Asynchronous actor-critic



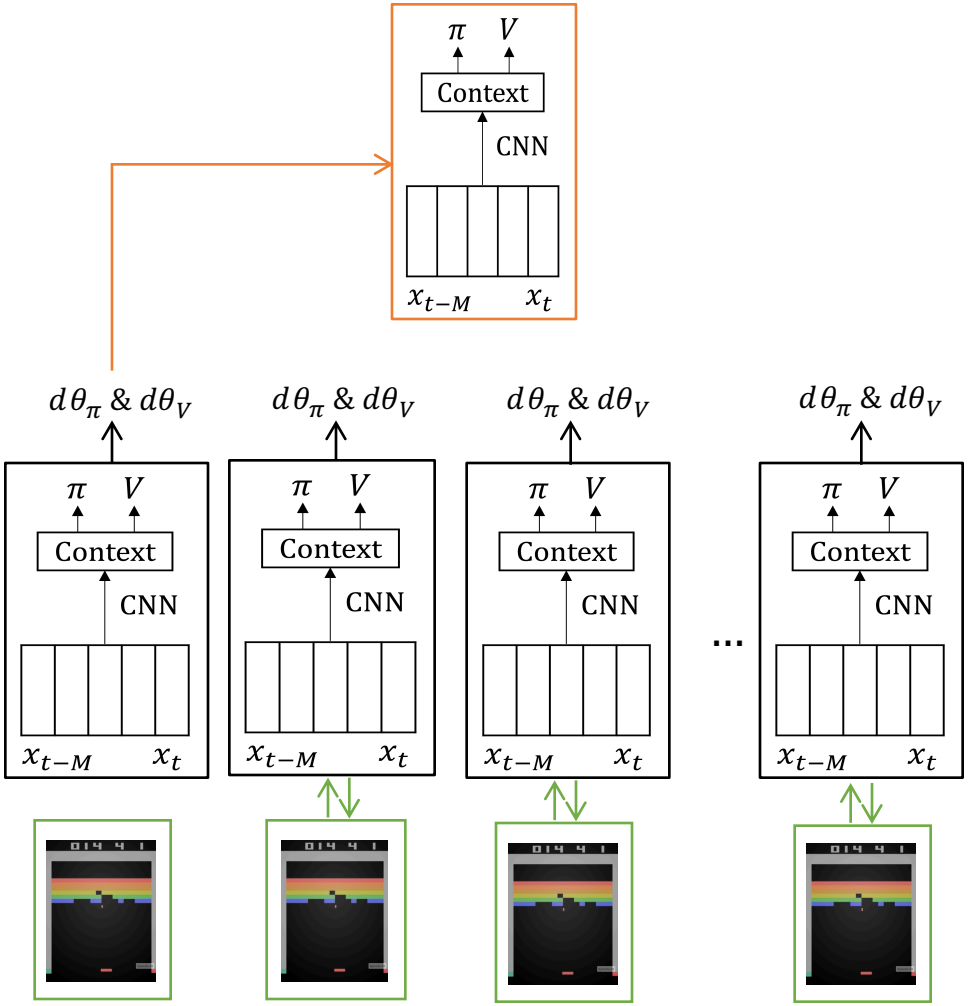

계산한 loss로
global network를 학습 시킴

각 thread에서 다른 게임을 하고

# Asynchronous actor-critic



# Asynchronous actor-critic

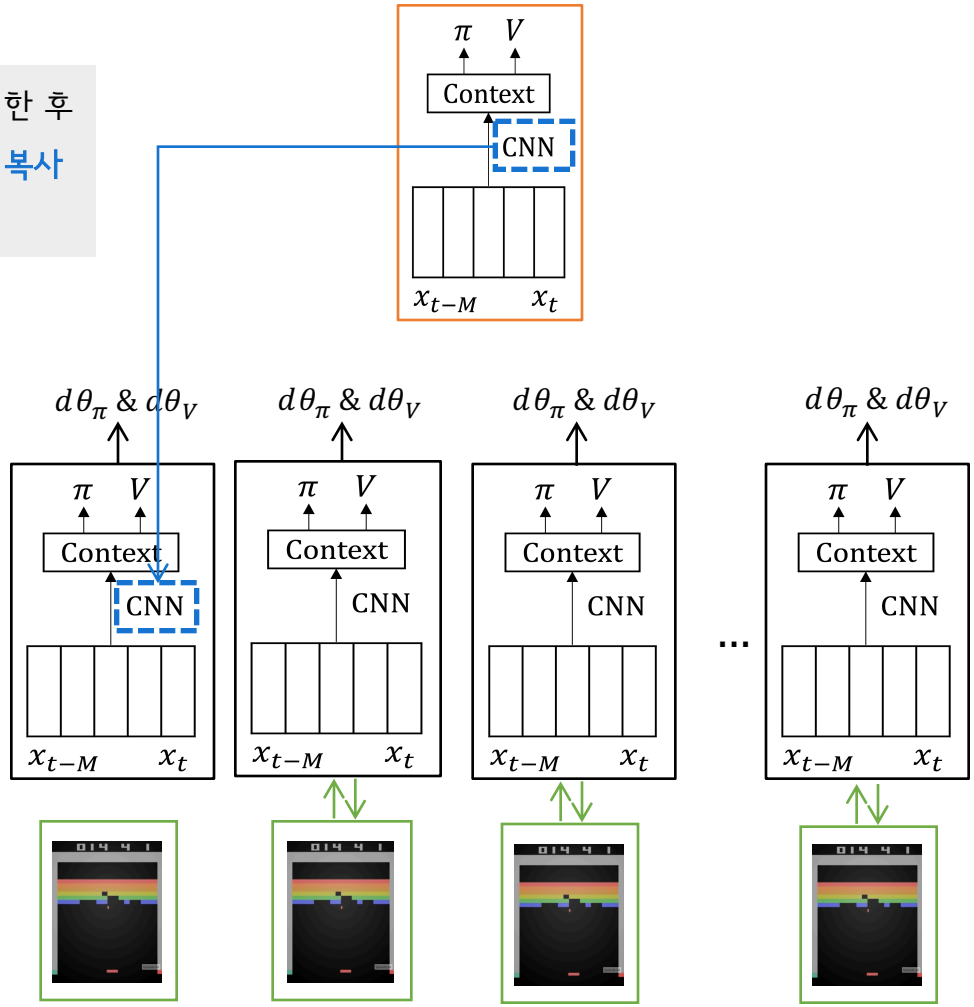


# Asynchronous actor-critic

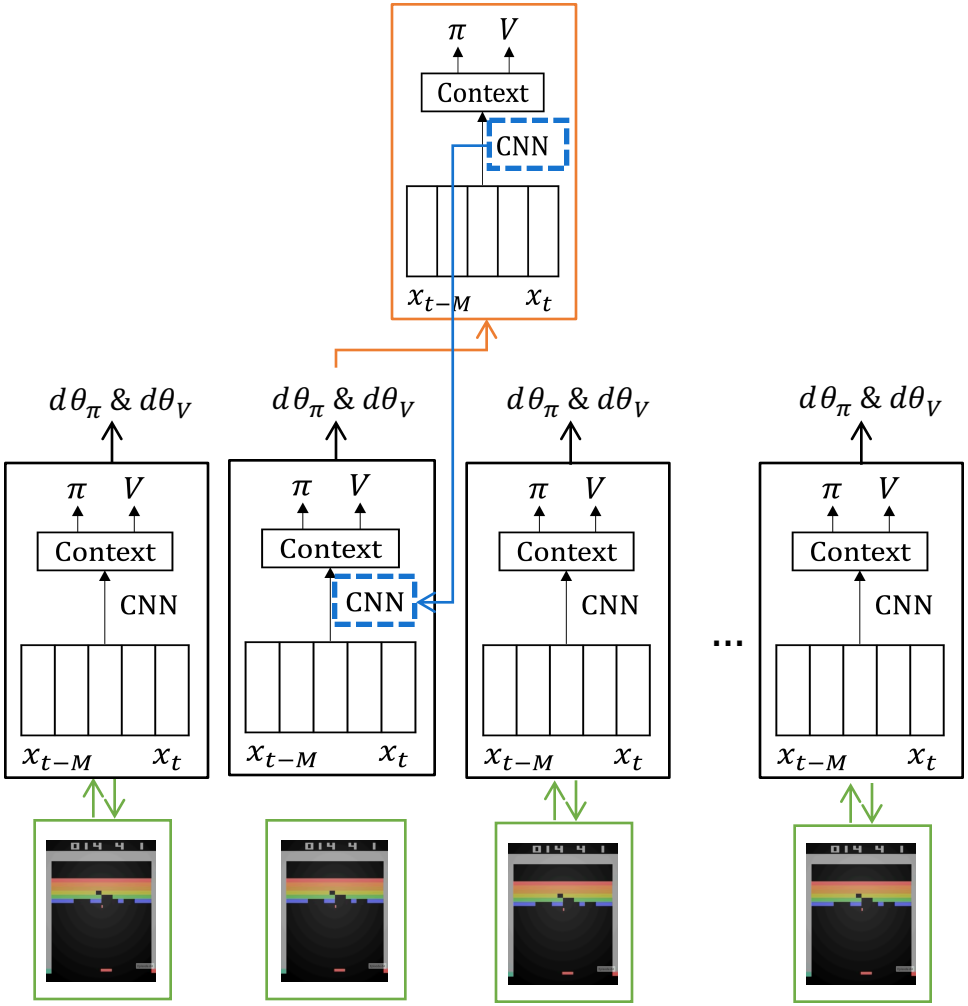
global network를 업데이트 한 후

local network로 weight를 복사

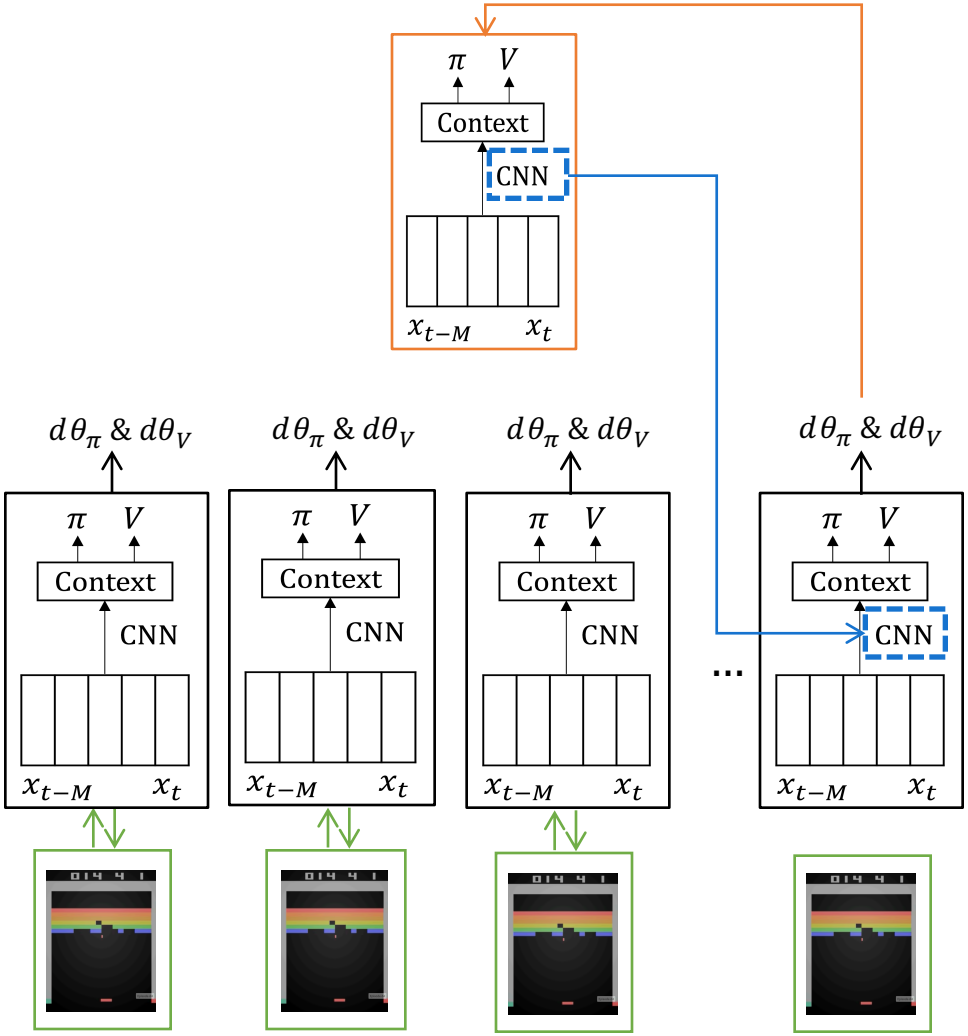
$\theta`_V \leftarrow \theta_V$



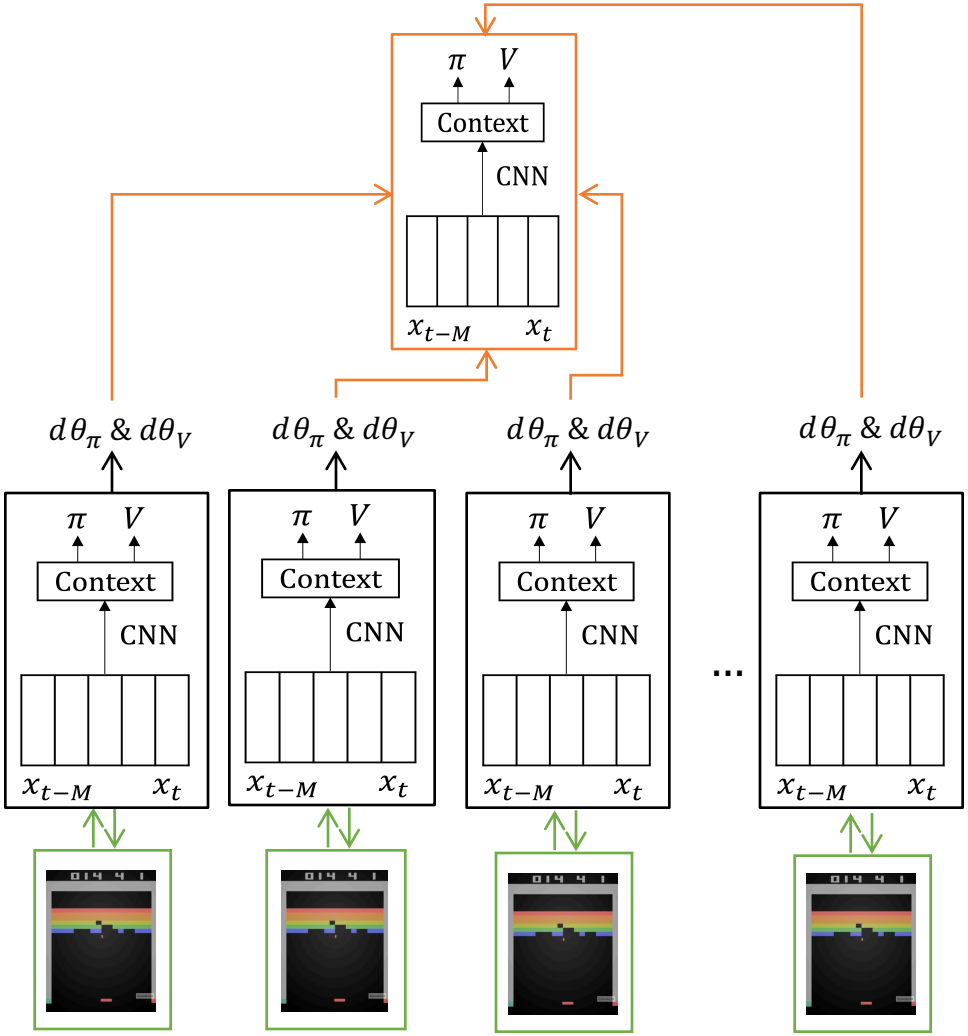
# Asynchronous actor-critic



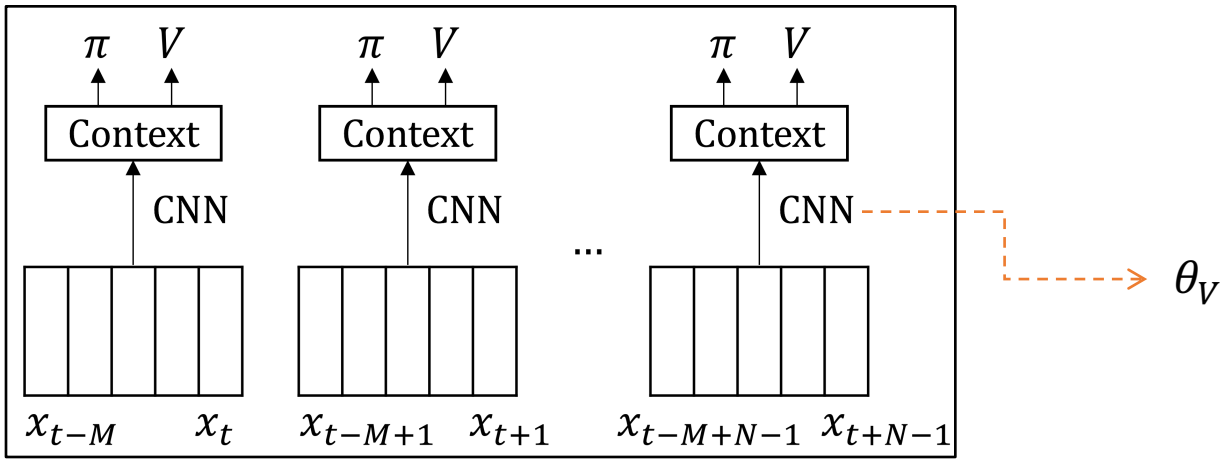
# Asynchronous actor-critic



# Asynchronous actor-critic

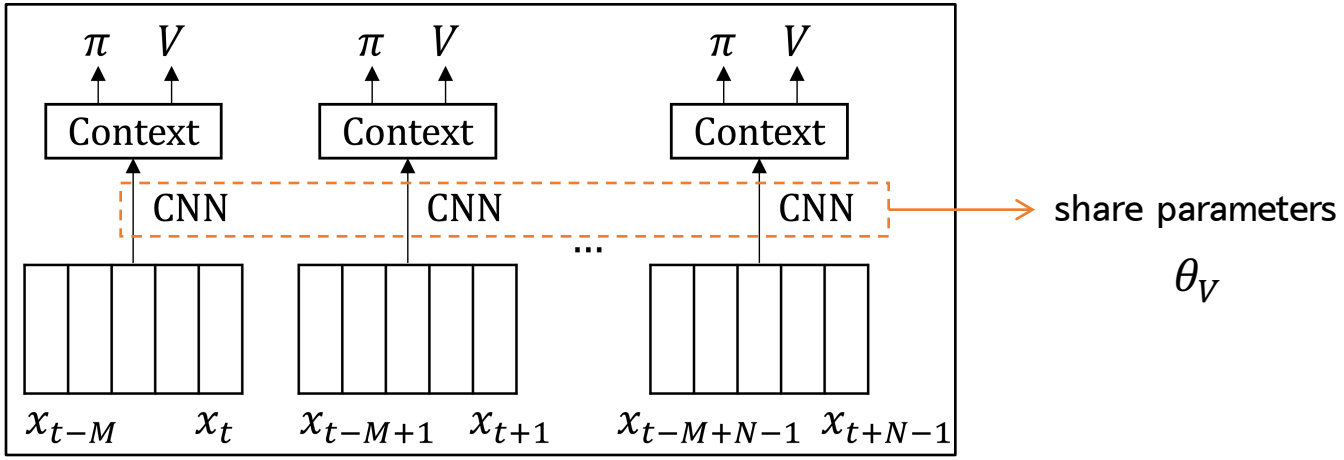


# Single thread
# n-step actor-critic worker



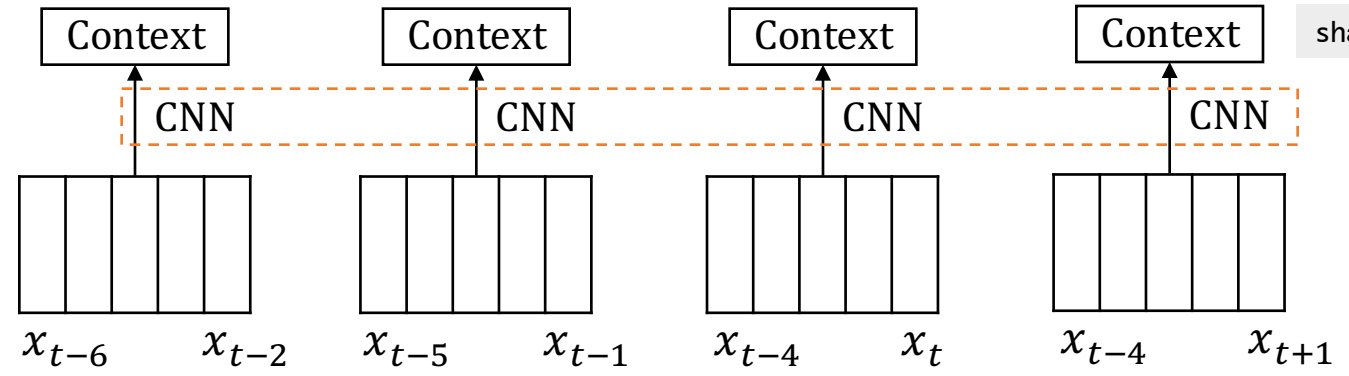

하나의 thread에 n 개의 actor-critic이 존재

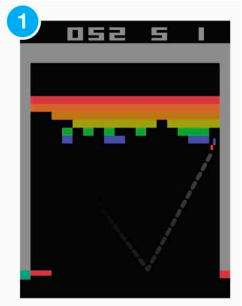
# Single thread
# n-step actor-critic worker




내부의 CNN은 weight를 share한다

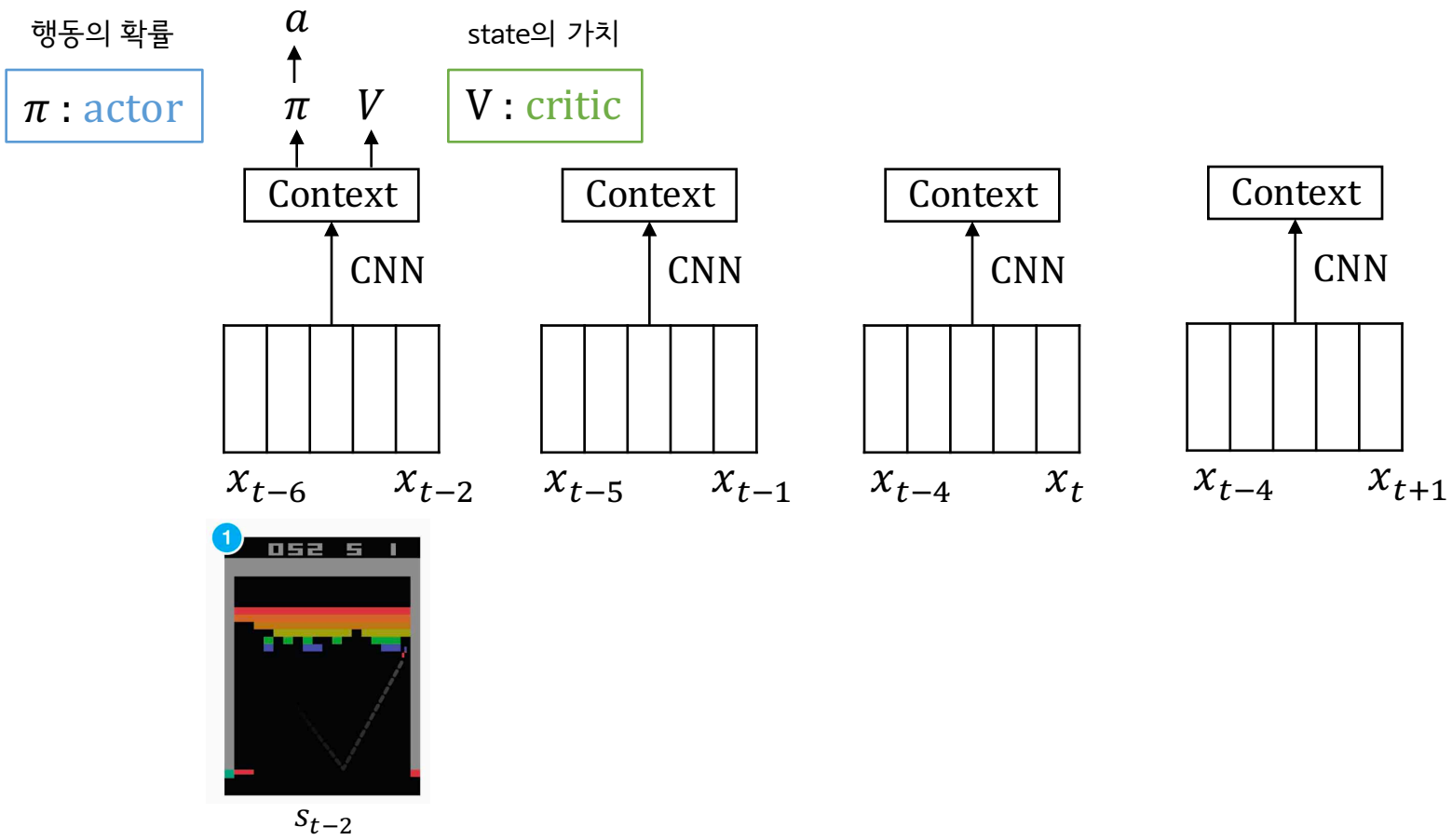
# 예제: 3-step actor-critic



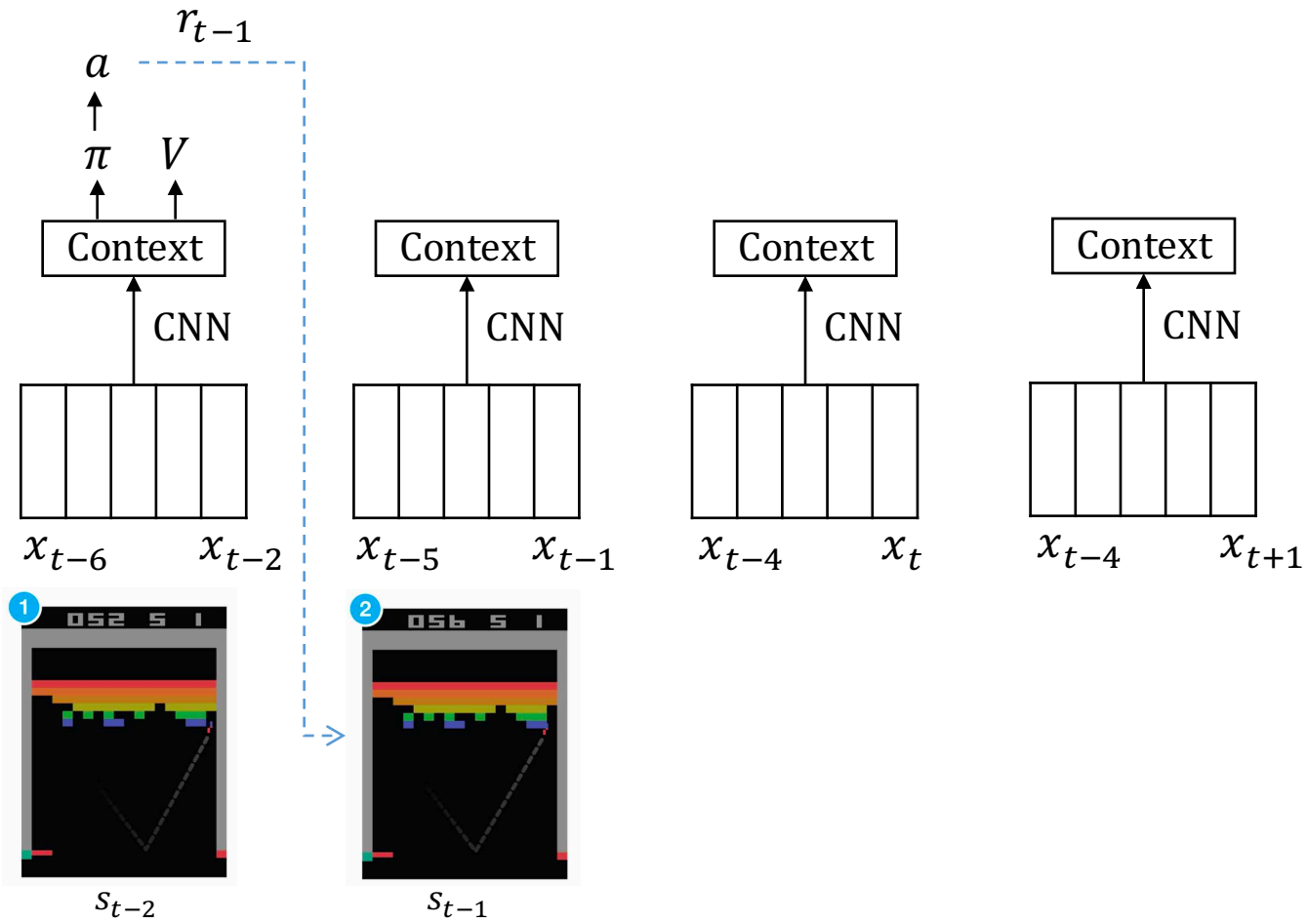




$s_{t-2}$

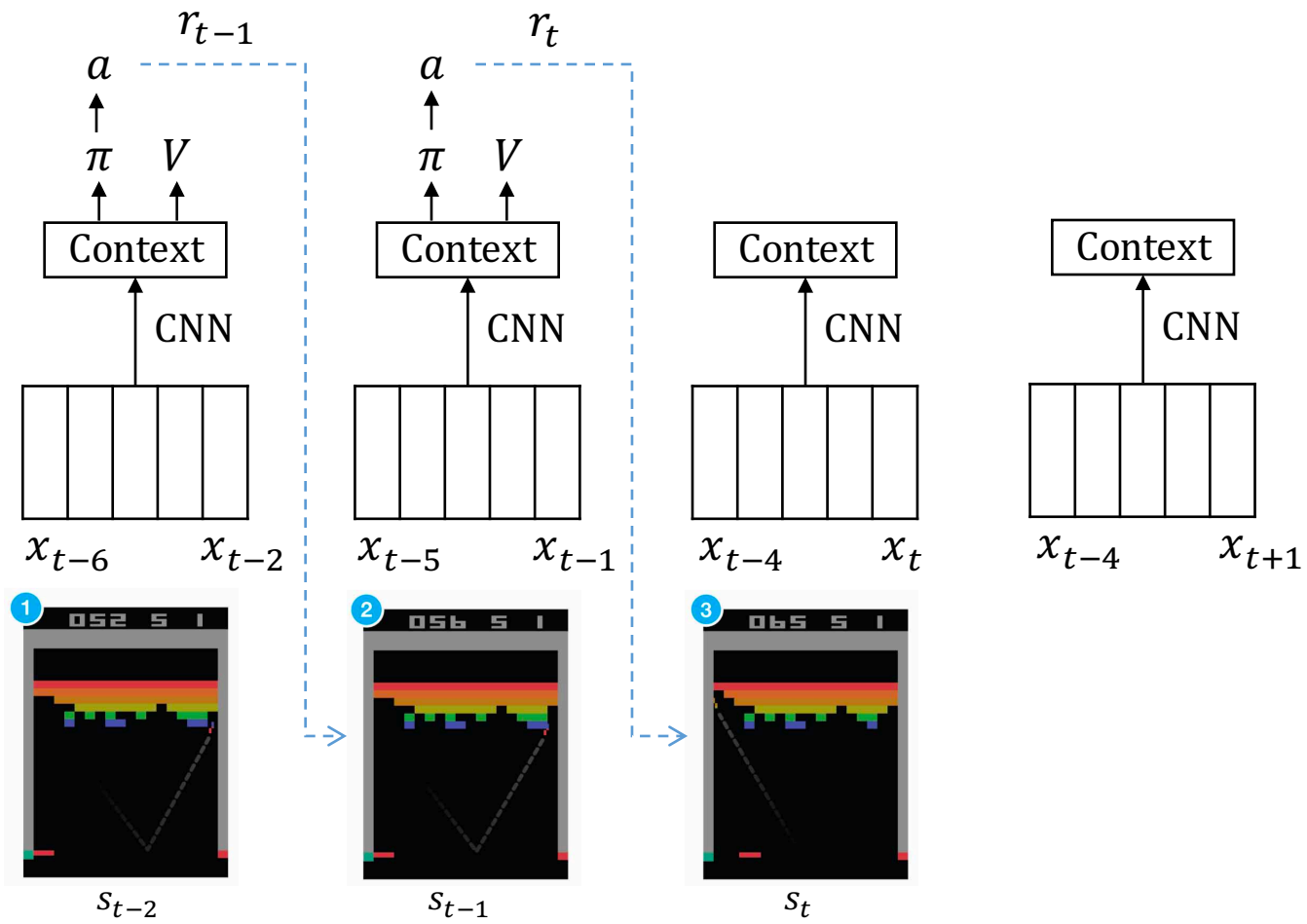
# $s_{t-2}$를 보고 $\pi$를 계산해 $a$를 구함



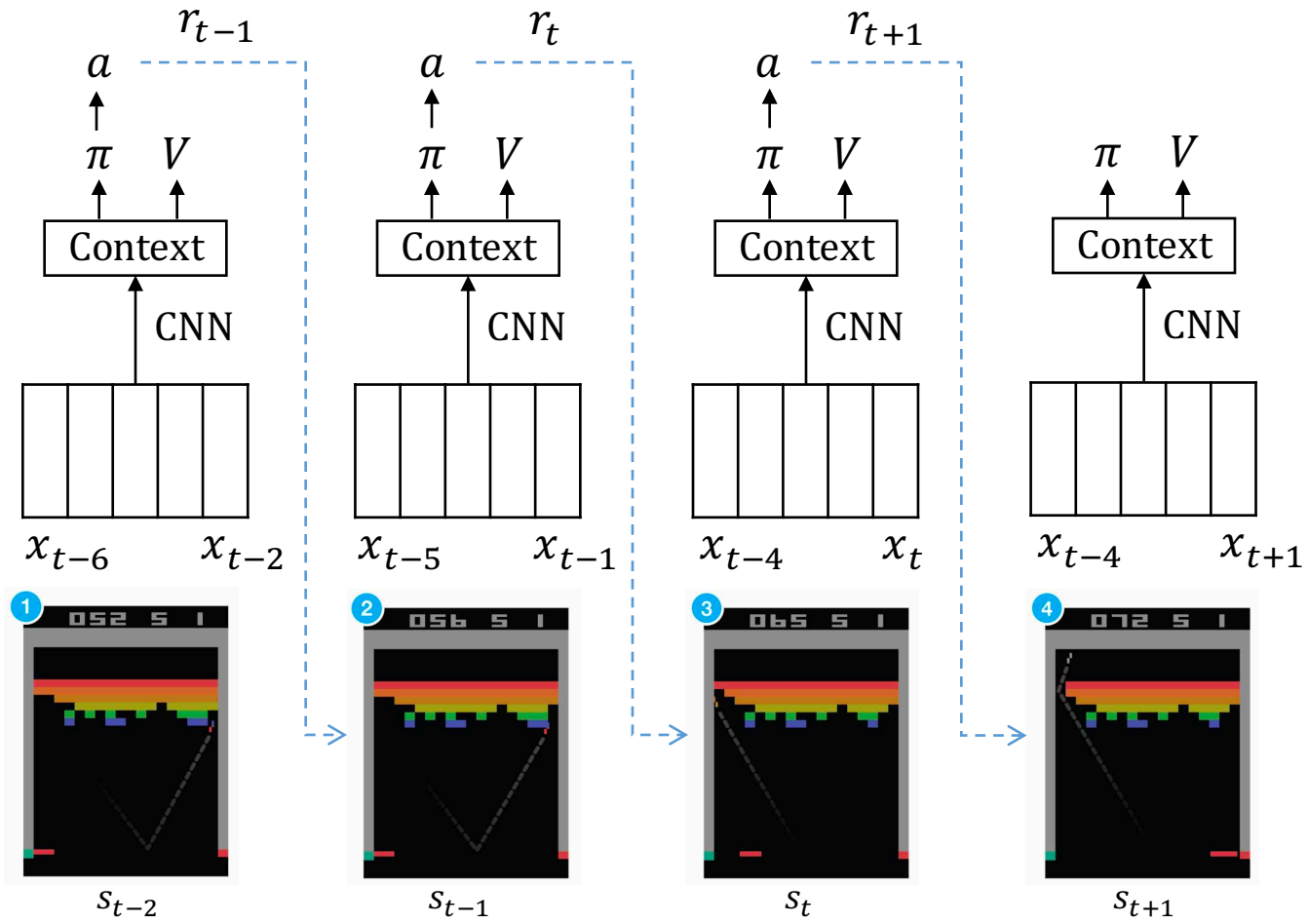
# agent가 $a$를 수행


$r_{t-1}$
$a$
$\pi$
$V$
Context
CNN
$x_{t-6}$
$x_{t-2}$
Context
CNN
$x_{t-5}$
$x_{t-1}$
Context
CNN
$x_{t-4}$
$x_t$
Context
CNN
$x_{t-4}$
$x_{t+1}$
1
052 5 1
2
056 5 1
$s_{t-2}$
$s_{t-1}$

# $s_{t-1}$를 보고 $\pi$를 계산해 $a$를 구함

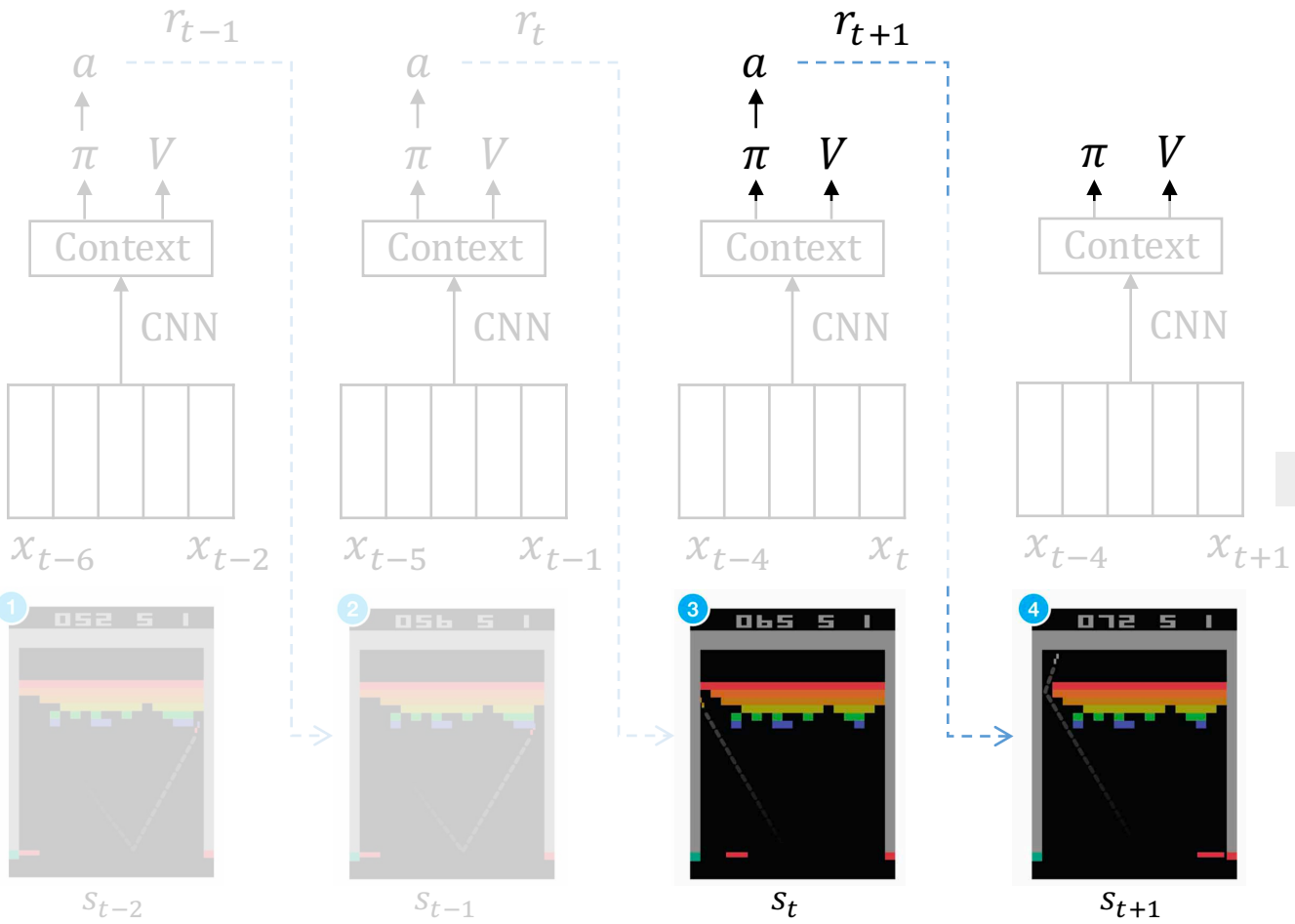


# $s_t$를 보고 $\pi$를 계산해 $a$를 구함



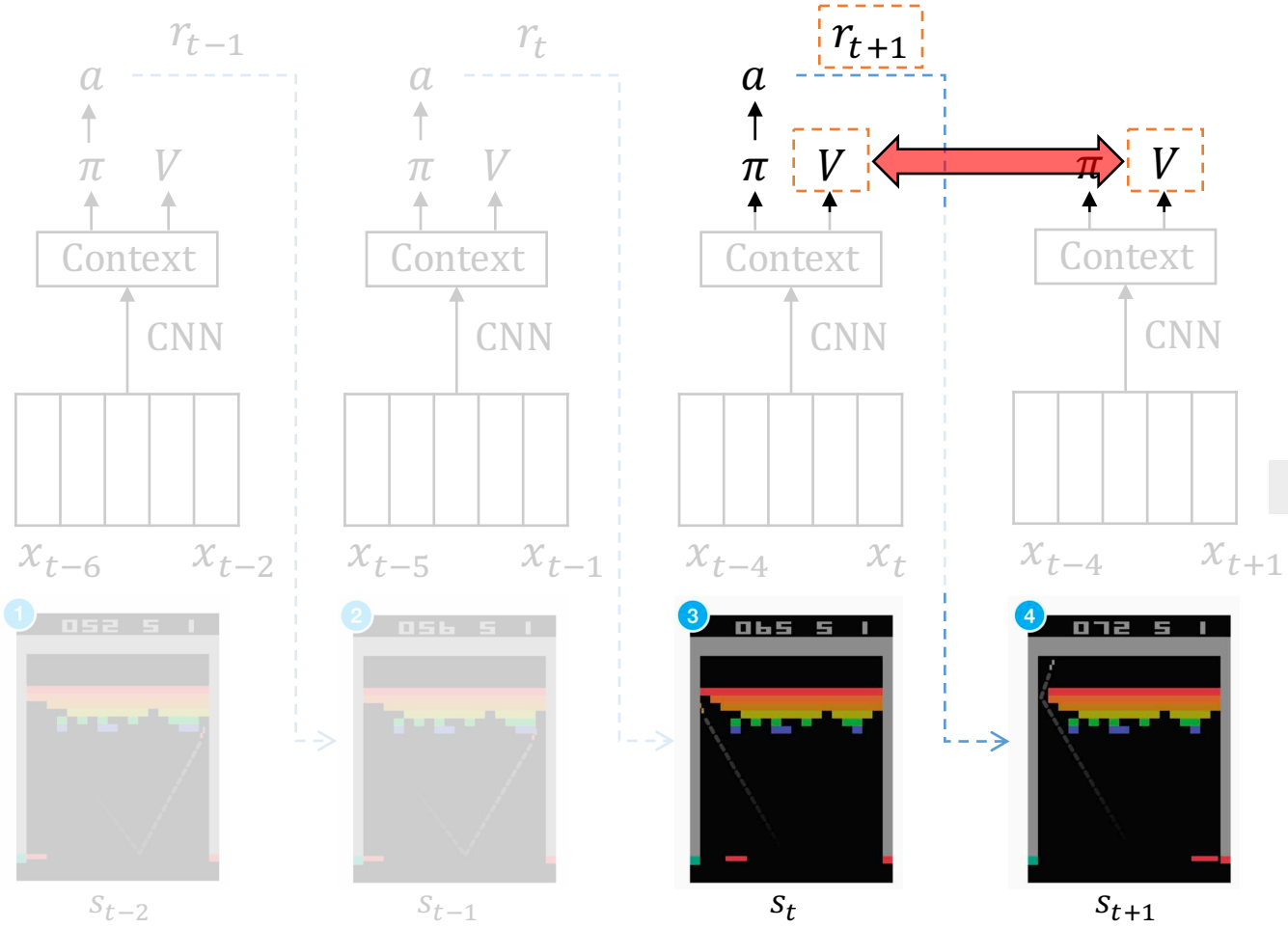
# gradient를 0으로 초기화하고

$$\partial\theta_V \leftarrow 0$$



# $r_{t+1}, V(s_t), V(s_{t+1})$를 사용해서

$\partial\theta_V \leftarrow 0$

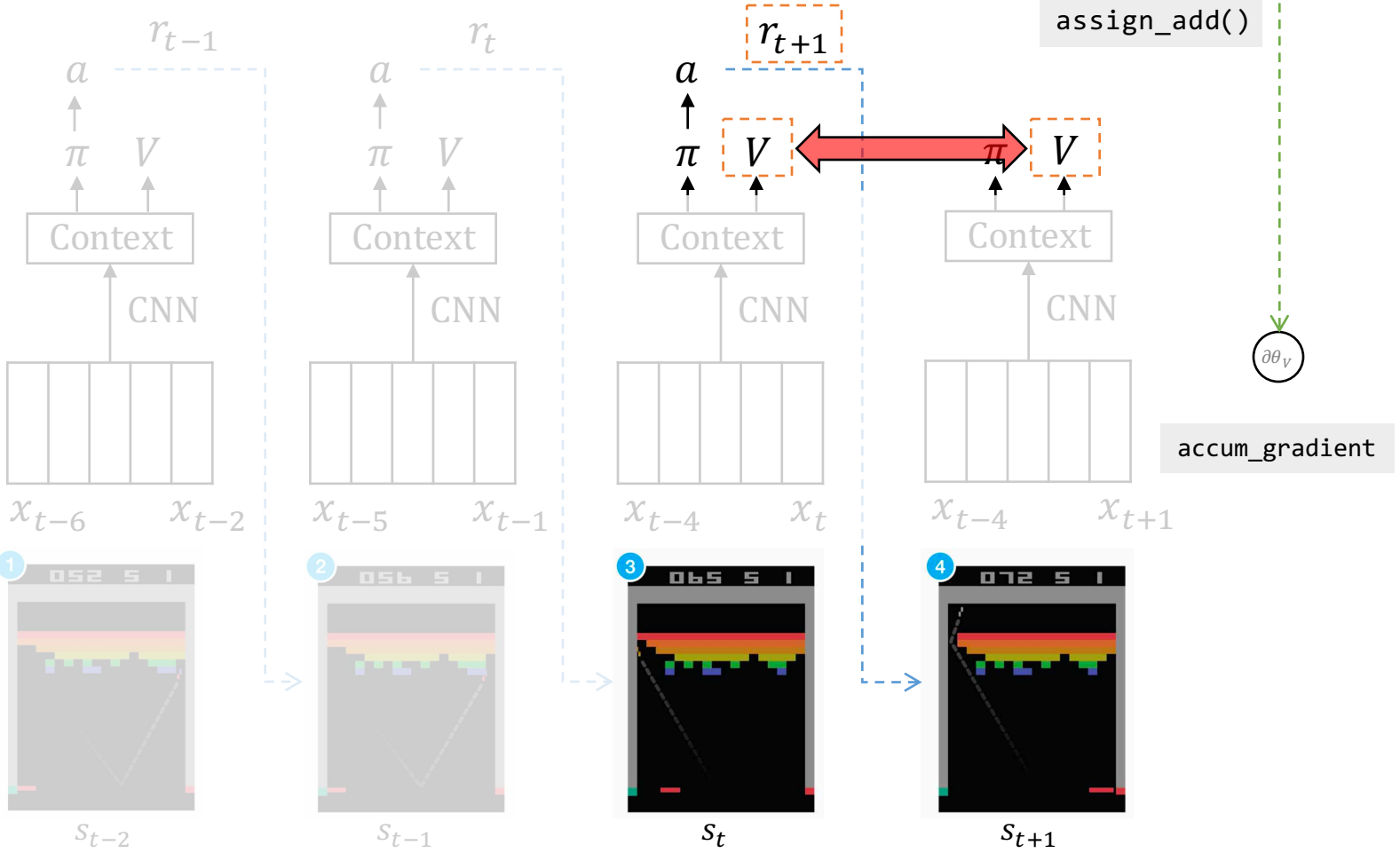


글로벌 네트워크의 Variable $\theta`_V$에 대한

# TD의 gradient를 더하고

Temporal Difference

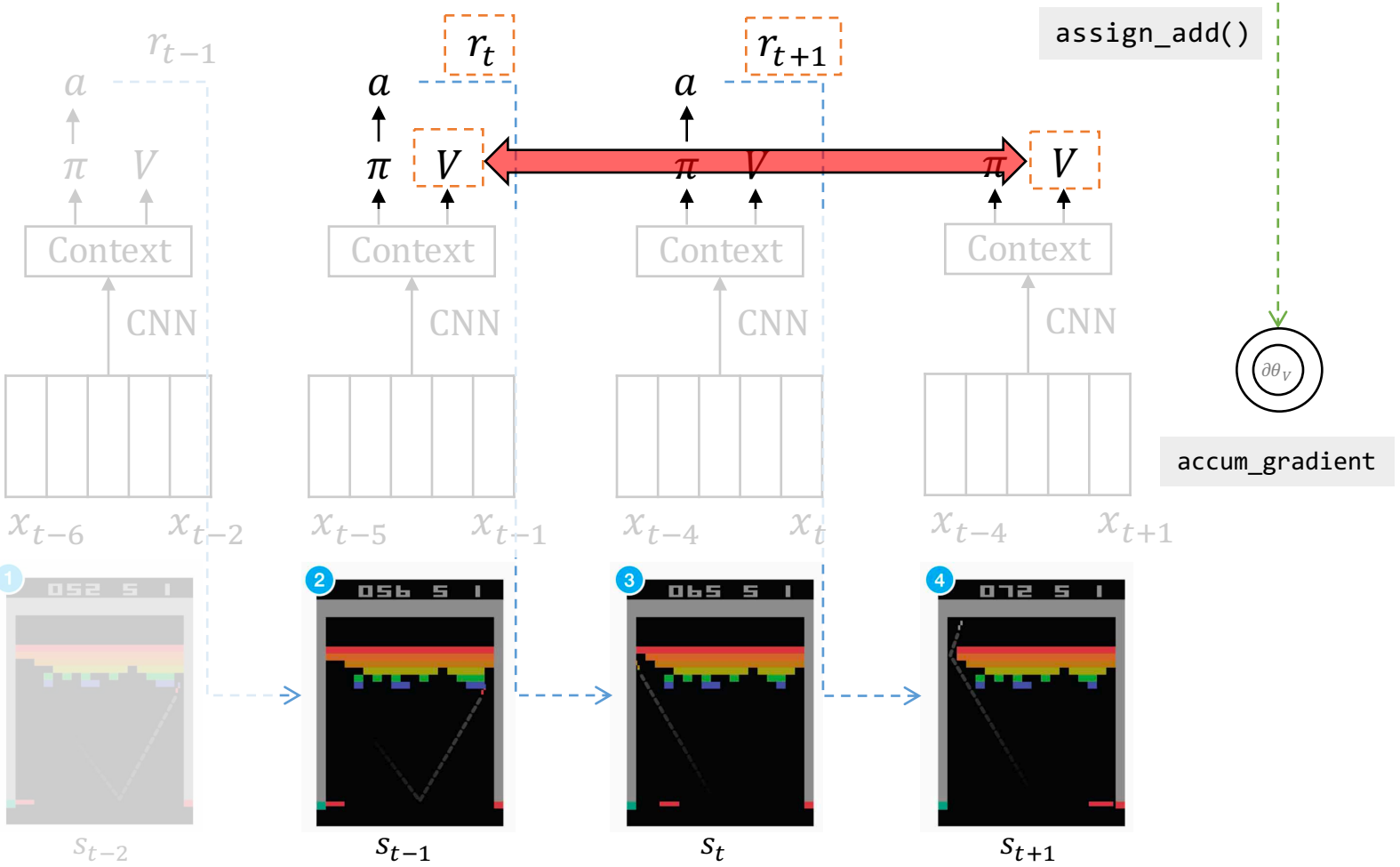
$$\partial\theta_V \leftarrow \partial\theta_V + \partial\left(r_{t+1} + \gamma V(s_{t+1}) - \mathrm{V}(\mathrm{s_t})\right)^2 / \partial\theta`_V$$



글로벌 네트워크의 Variable $\theta`_V$에 대한

# 또, TD의 gradient를 더하고

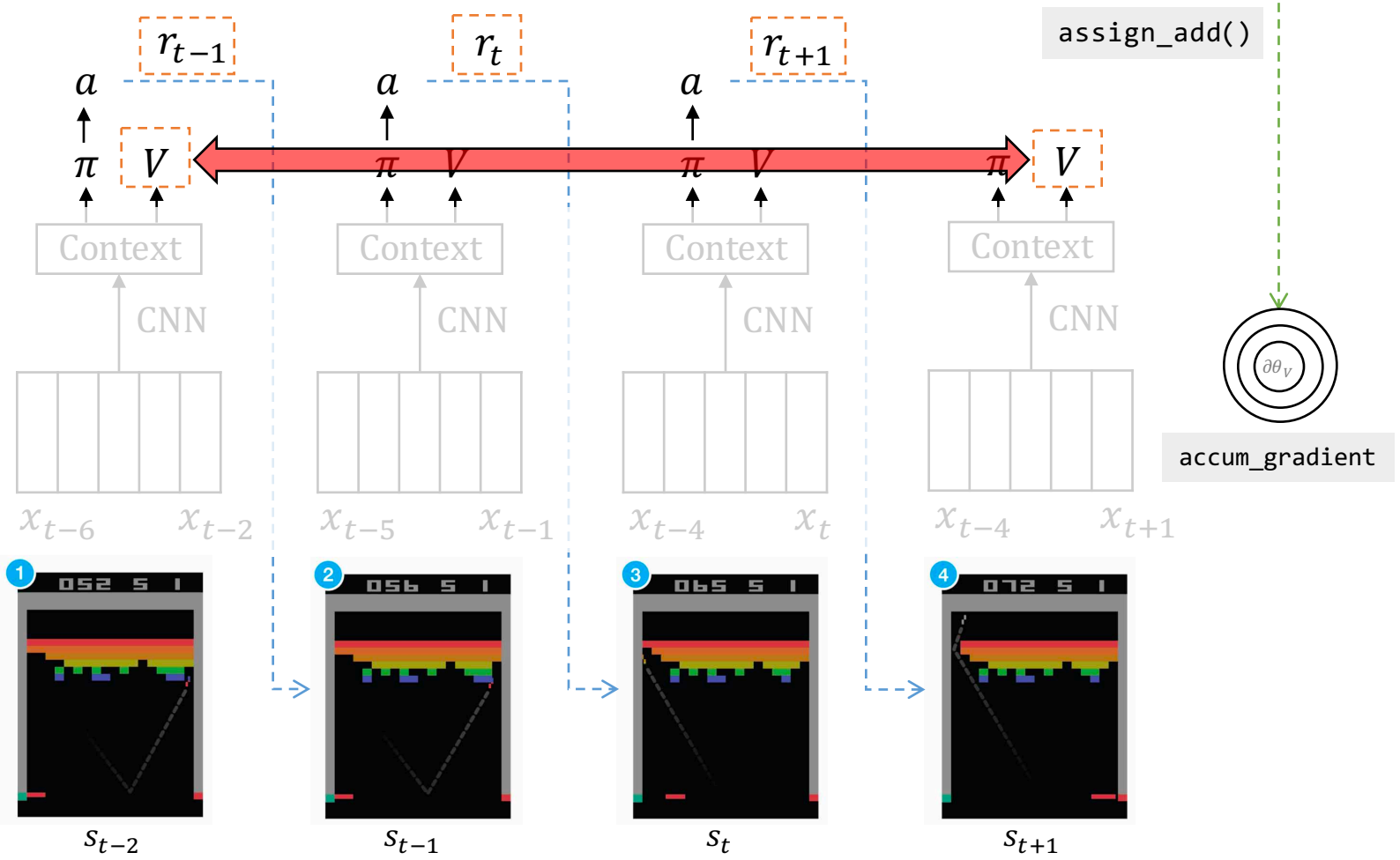
$$\partial\theta_V \leftarrow \partial\theta_V + \partial\left(r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+1}) - \mathrm{V}(\mathrm{s}_{t-1})\right)^2 / \partial\theta`_V$$



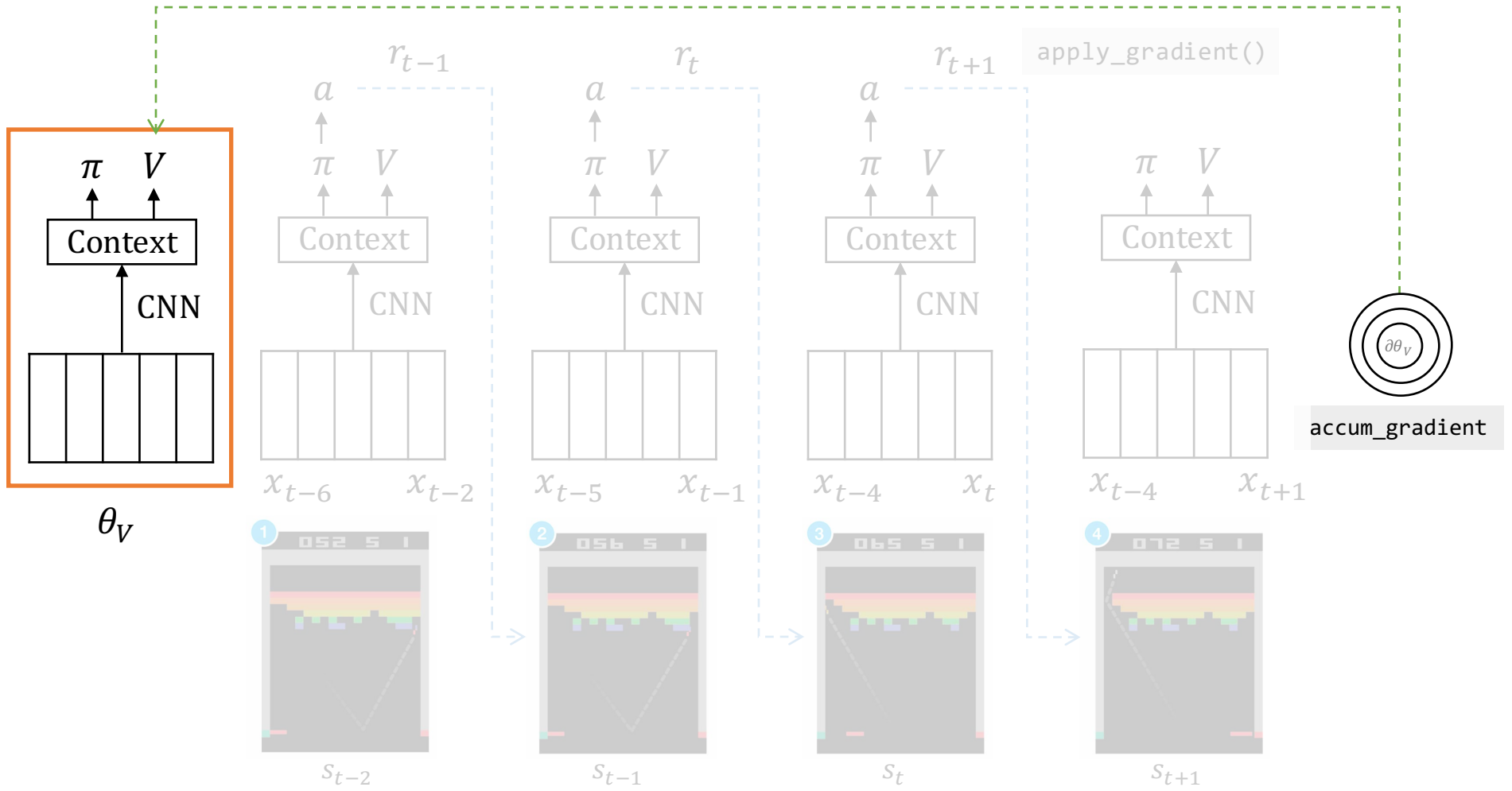
글로벌 네트워크의 Variable $\theta`_V$에 대한

# 또, TD의 gradient를 더해서

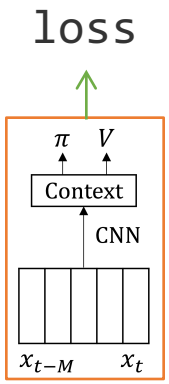
$$\partial\theta_V \leftarrow \partial\theta_V + \partial\left(r_{t-1} + \gamma r_t + \gamma^2 r_{t+1} + \gamma^3 V(s_{t+1}) - \mathrm{V}(\mathrm{s}_{t-2})\right)^2 / \partial\theta`_V$$



# 글로벌 네트워크의 Variable를 업데이트

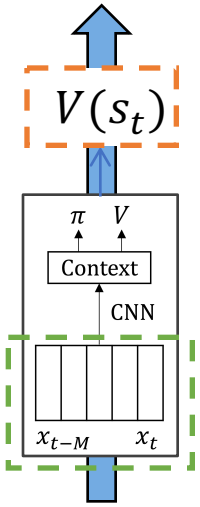


하지만 여기엔 문제가 있었습니다




```
optim = tf.train.GradientDescentOptimizer(0.5).minimize(loss)

sess.run(optim, feed_dict={inputs: s_t})
```

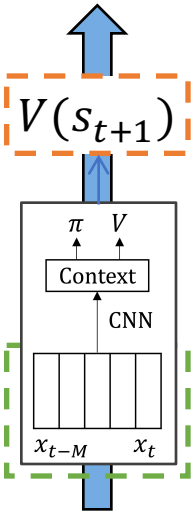
보통 optim을 정의하고 optim.minimize(loss)를 run()




```
v_s_0 = sess.run(V, feed_dict={inputs: s_t_0})
```

*n-th step*이 되기 전까지 action을 하기 위해 `run()`

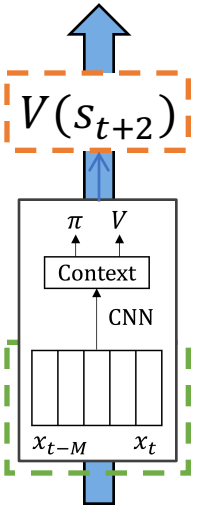
$V(s_t)$ $V(s_{t+1})$




```
v_s_1 = sess.run(V, feed_dict={inputs: s_t_1})
```

*n-th* step이 되기 전까지 action을 하기 위해 `run()`

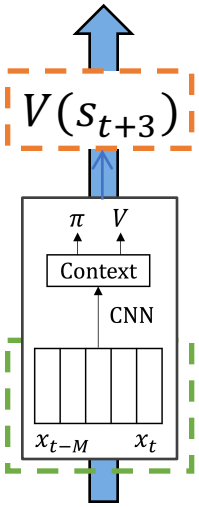
$V(s_t)$ $V(s_{t+1})$ $V(s_{t+2})$




```
v_s_2 = sess.run(V, feed_dict={inputs: s_t_2})
```

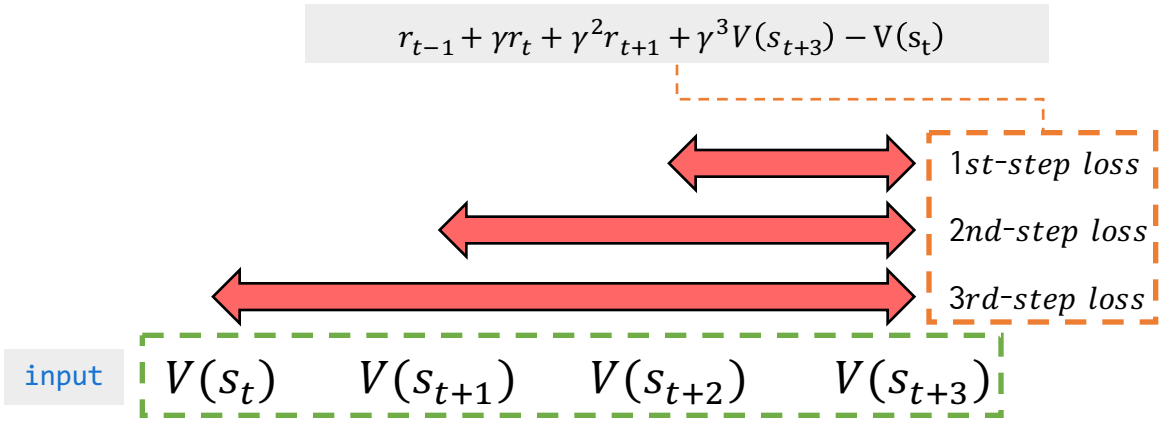
*n-th* step이 되기 전까지 action을 하기 위해 run()

$V(s_t)$ $V(s_{t+1})$ $V(s_{t+2})$ $V(s_{t+3})$



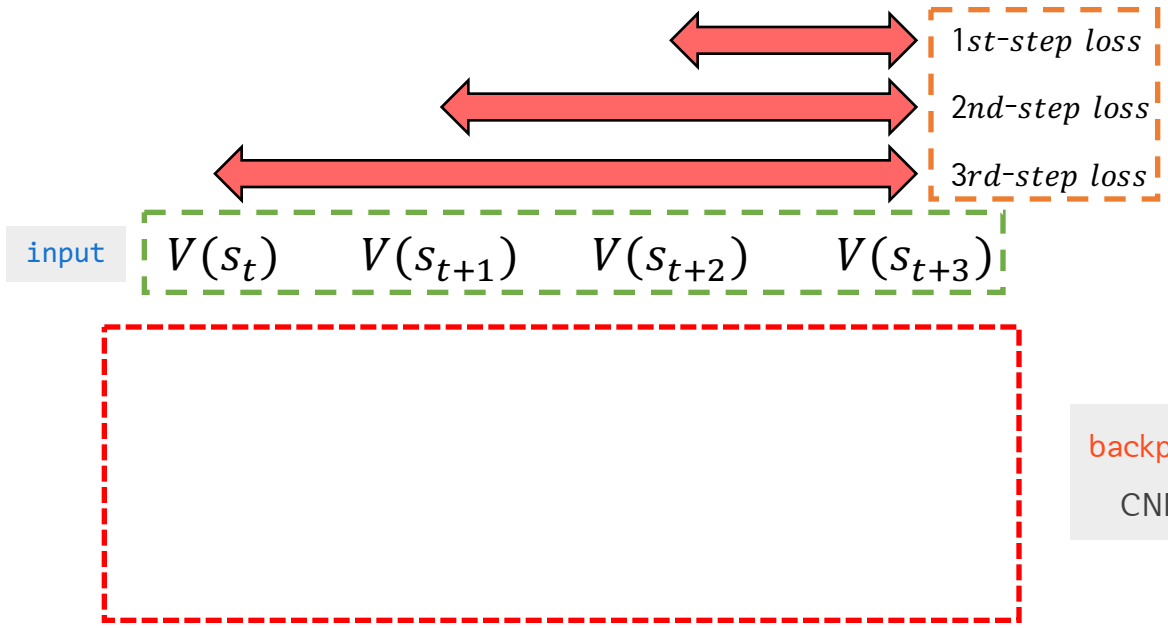

```
v_s_3 = sess.run(V, feed_dict={inputs: s_t_3})
```

*n-th* step이 되기 전까지 action을 하기 위해 `run()`




```
sess.run(optim, feed_dict=
    input {V_t_0: v_t_0, V_t_1: v_t_1, V_t_2: v_t_2, V_t_3: v_t_3})
```

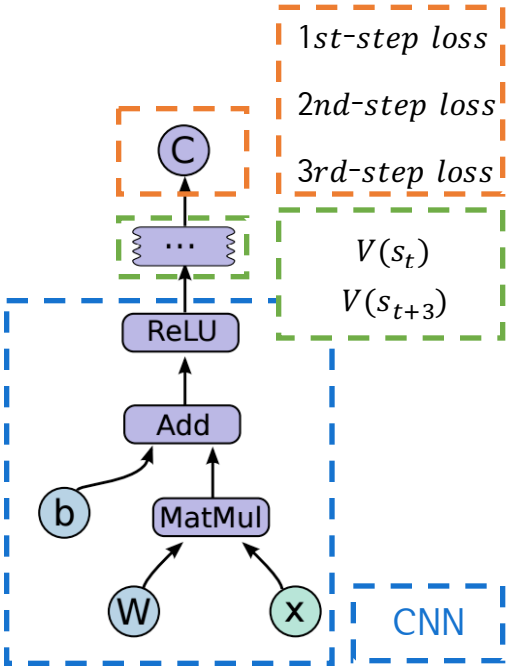
학습이 될까?




backpropagation하기 위한
CNN 내부에 값이 없음

```
sess.run(optim, feed_dict=
    input {V_t_0: v_t_0, V_t_1: v_t_1, V_t_2: v_t_2, V_t_3: v_t_3})
```

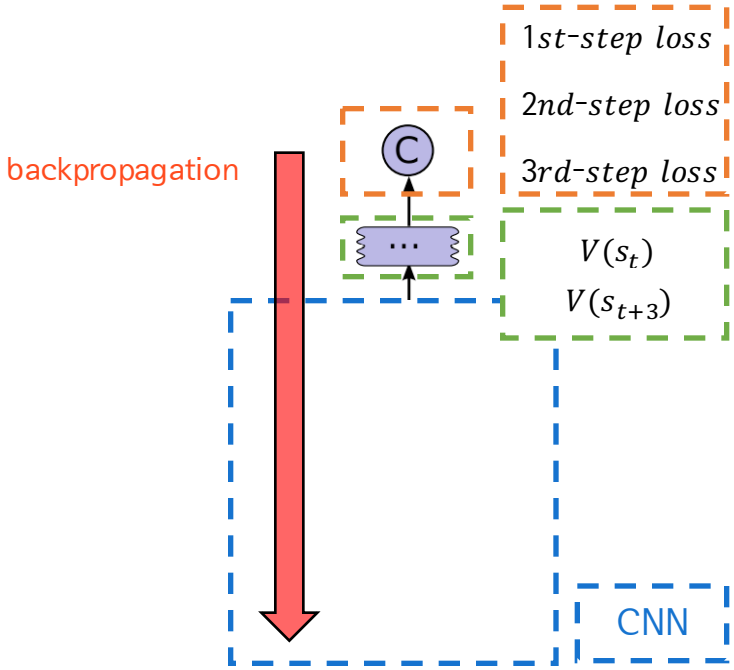
오류 발생!




```
sess.run(optim, feed_dict=
    input  {V_t_0: v_t_0, V_t_1: v_t_1, V_t_2: v_t_2, V_t_3: v_t_3})
```

이런 computation graph가 있을때

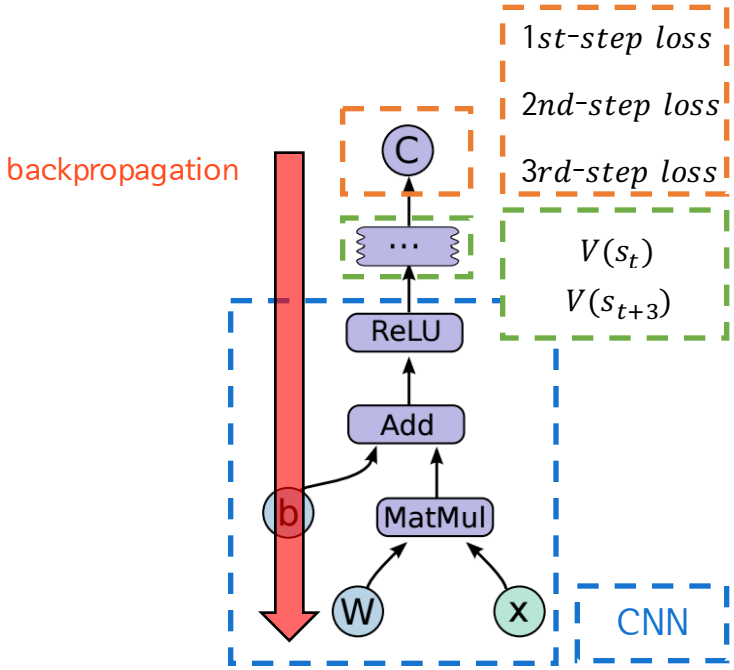
TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems




```
sess.run(optim, feed_dict=
    input {V_t_0: v_t_0, V_t_1: v_t_1, V_t_2: v_t_2, V_t_3: v_t_3})
```

loss를 계산하기 위해 사용된 값들을 모르면서 $\theta$를 업데이트 하려는 것

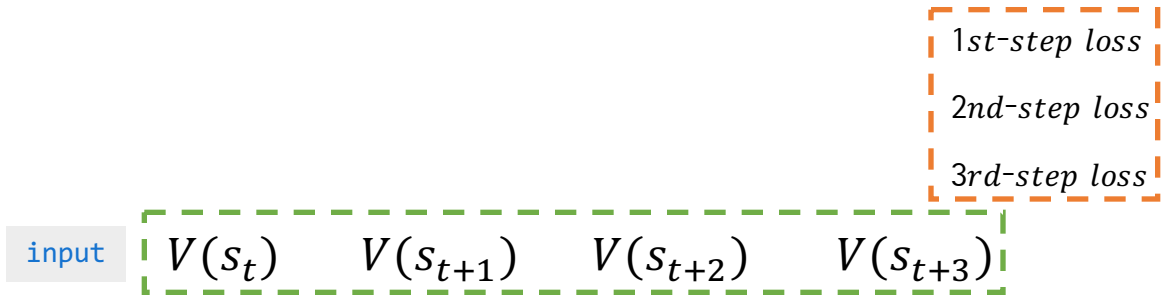
TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems




```
sess.run(optim, feed_dict=
    input  {V_t_0: v_t_0, V_t_1: v_t_1, V_t_2: v_t_2, V_t_3: v_t_3})
```

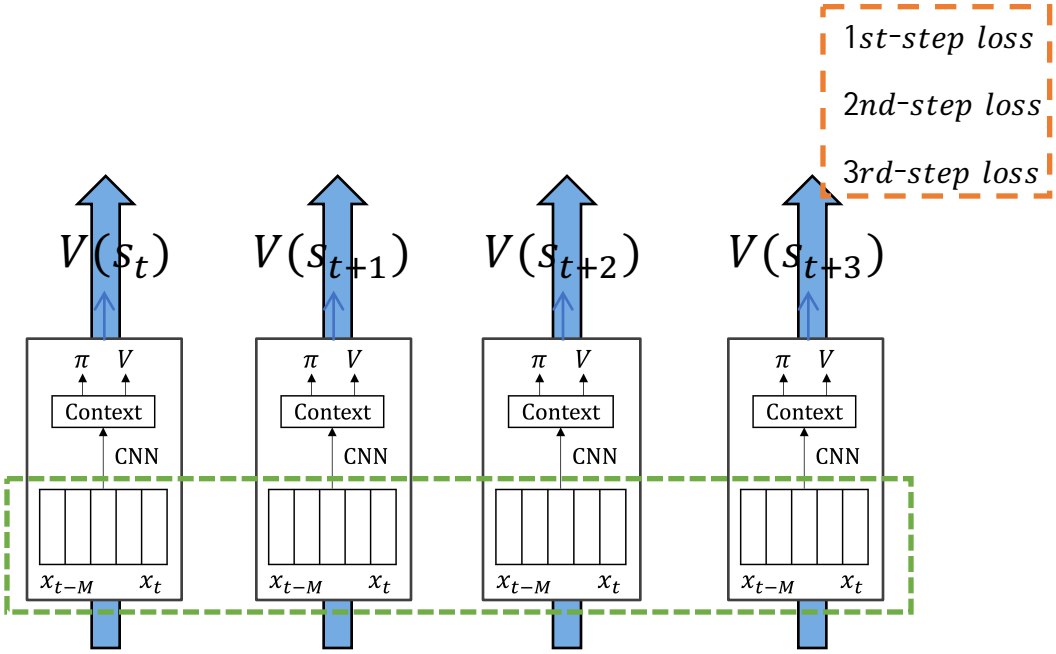
CNN 내부 계산값도 알아야 한다

TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems




```
sess.run(optim, feed_dict=
    input  {V_t_0: v_t_0, V_t_1: v_t_1, V_t_2: v_t_2, V_t_3: v_t_3})
```

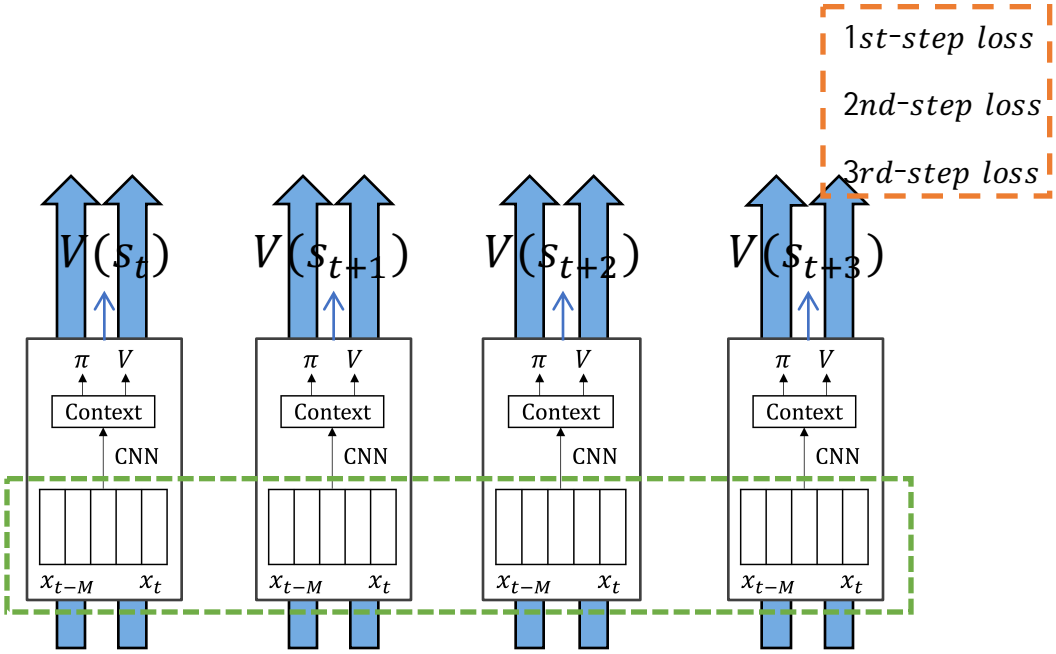
그래서 gradient를 적용하기 위해




```
sess.run(optim, feed_dict=
```

input `{inputs_0: s_t_0, inputs_0: s_t_1, inputs_0: s_t_2, inputs_0: s_t_3})`

input부터 다시 넣어 CNN을 다시 계산해야함



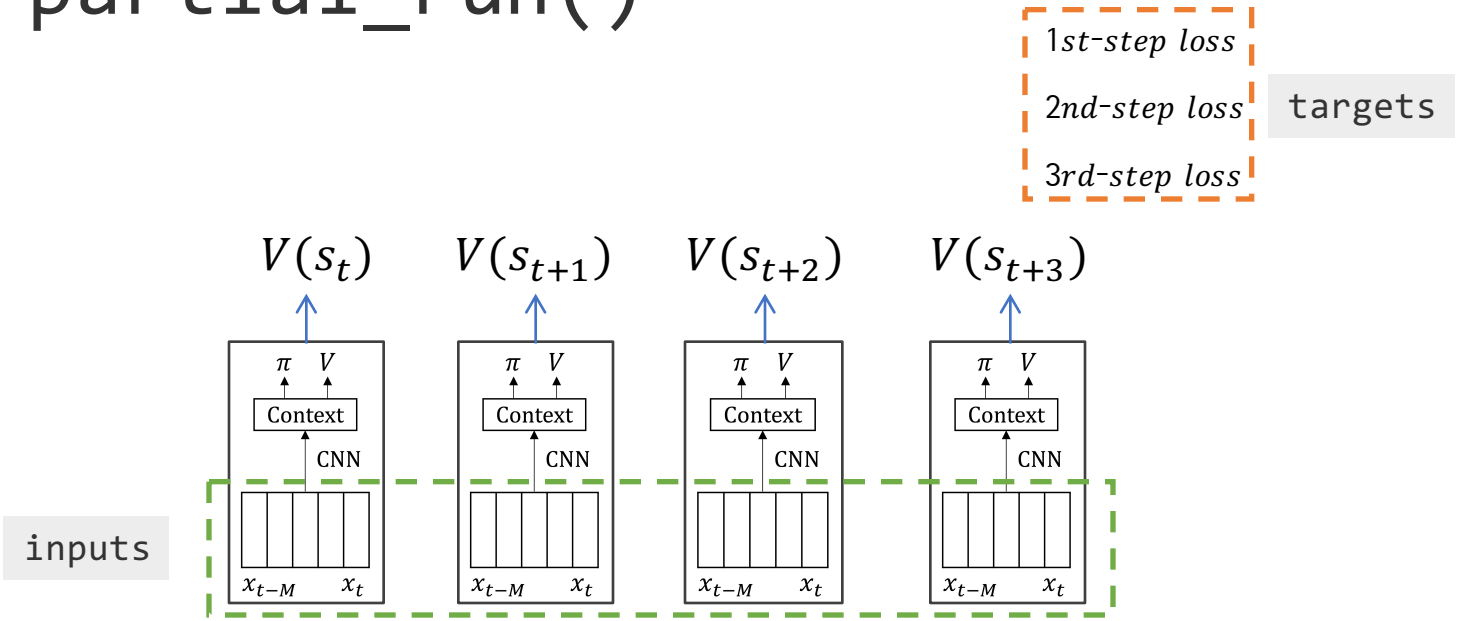

```
sess.run(optim, feed_dict=
```

input `{inputs_0: s_t_0, inputs_0: s_t_1, inputs_0: s_t_2, inputs_0: s_t_3})`

하지만 실제론

action을 얻기 위해 계산했던 CNN을 또 계산해야하는 비효율이 생김

# Tip: partial_run()




```
partial_graph = sess.partial_run_setup(targets, inputs)
```

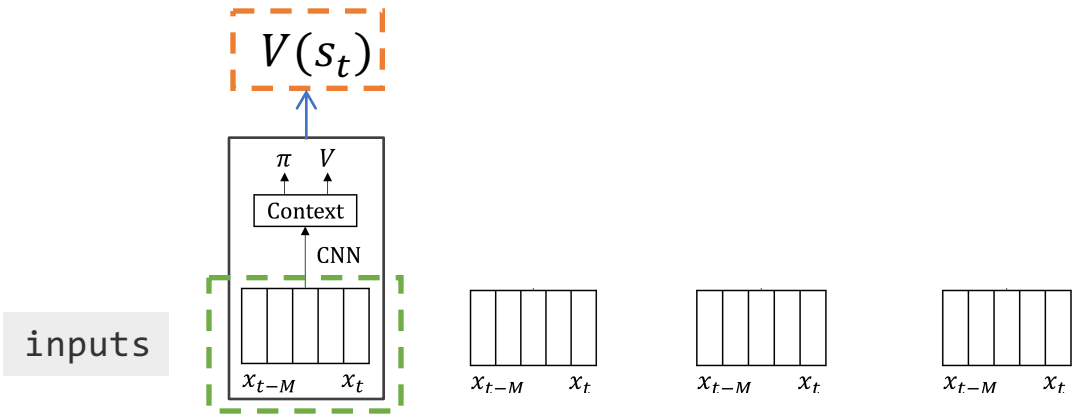
run한 값을 메모리에 저장하는 새로운 abstract network를 정의

# Tip: partial_run()

$1st\text{-}step\ loss$

$2nd\text{-}step\ loss$ `targets`

$3rd\text{-}step\ loss$




```
v_s_0 = sess.partial_run(partial_graph , V_s_0, feed_dict={inputs: s_t_0})
```

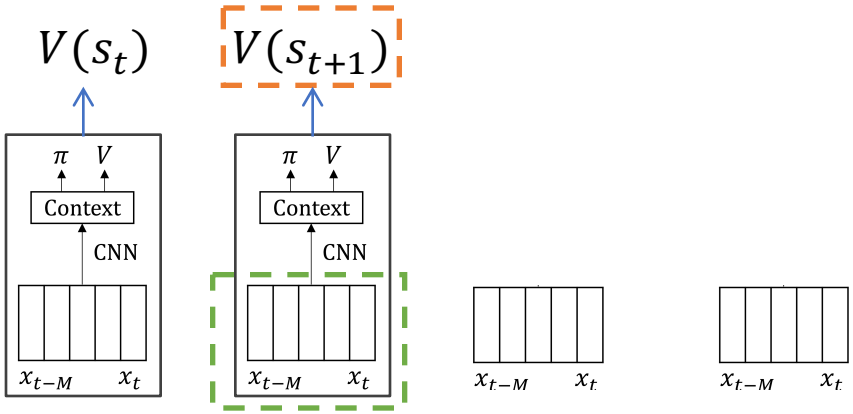
처음부터 하나씩 forward해도

# Tip: partial_run()

$1st\text{-}step\ loss$

$2nd\text{-}step\ loss$ `targets`

$3rd\text{-}step\ loss$




```
v_s_1 = sess.partial_run(partial_graph , V_s_1, feed_dict={inputs: s_t_1})
```

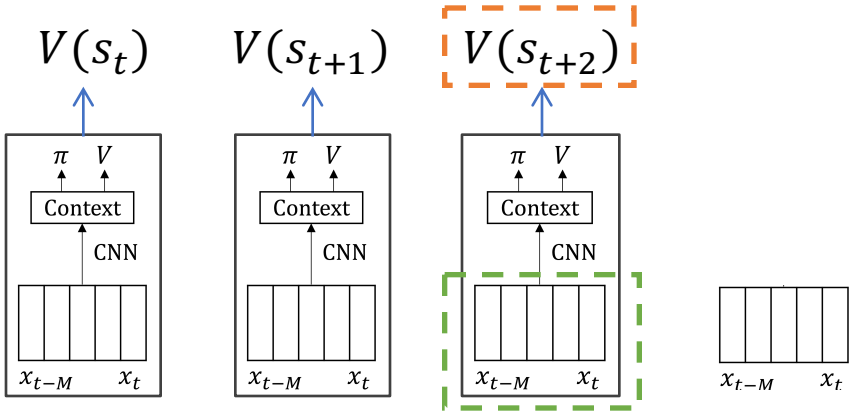
메모리에 중간 값이 저장되고

# Tip: partial_run()

$1st\text{-}step\ loss$

$2nd\text{-}step\ loss$

$3rd\text{-}step\ loss$




```
v_s_2 = sess.partial_run(partial_graph , V_s_2, feed_dict={inputs: s_t_2})
```

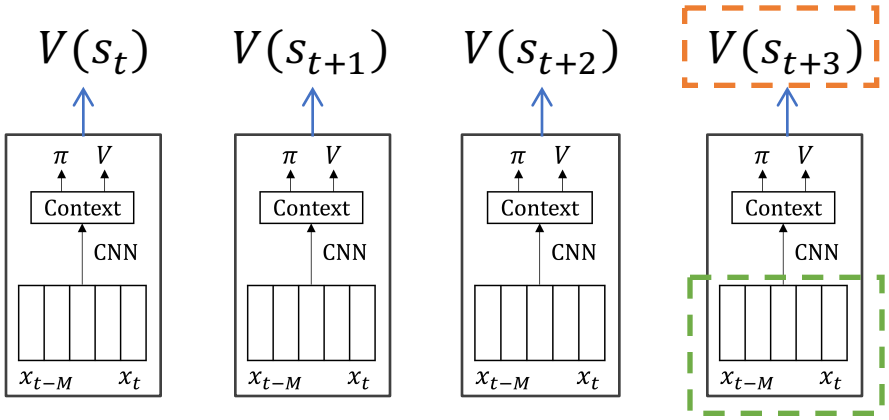
메모리에 중간 값이 계속 저장되고

# Tip: partial_run()

$1st\text{-}step\ loss$

$2nd\text{-}step\ loss$

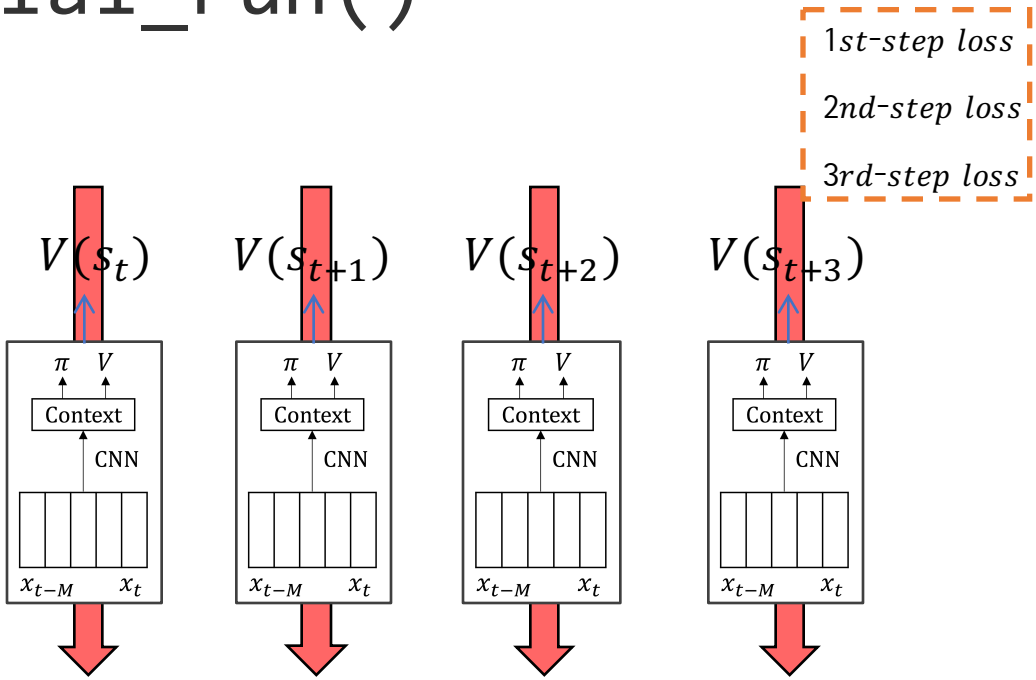
$3rd\text{-}step\ loss$




```
v_s_3 = sess.partial_run(partial_graph, V_s_3, feed_dict={inputs: s_t_3})
```

메모리에 중간 값이 계속 계속 저장되어

# Tip: partial_run()

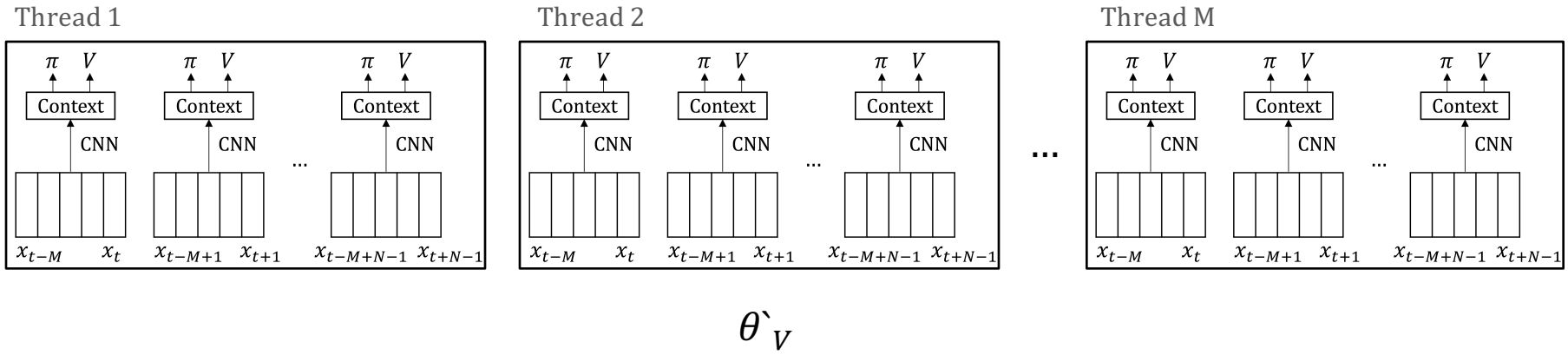



```
sess.partial_run(partial_graph, optim)
```

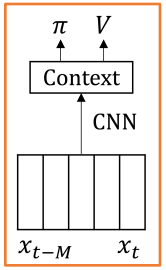
마지막 update때 input을 다시 넣어 forward 계산을 다시 하지 않아도 된다

# Multi thread

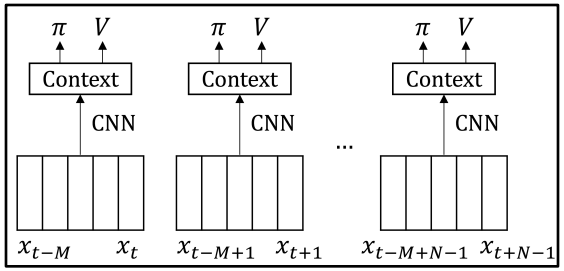
## n-step actor-critic

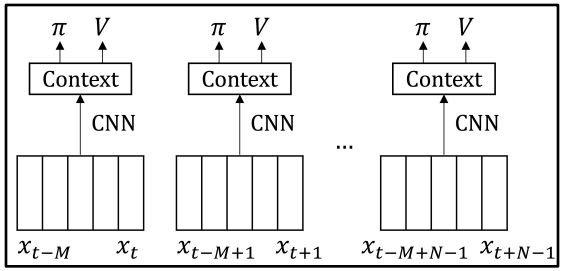


# Global actor-critic network

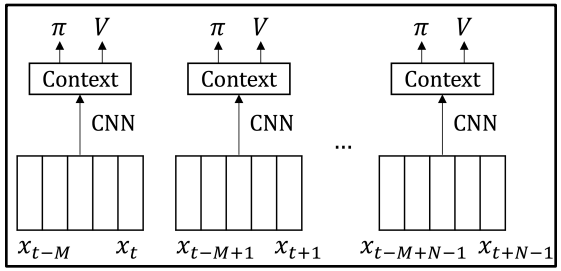



$\theta_V$ global weight



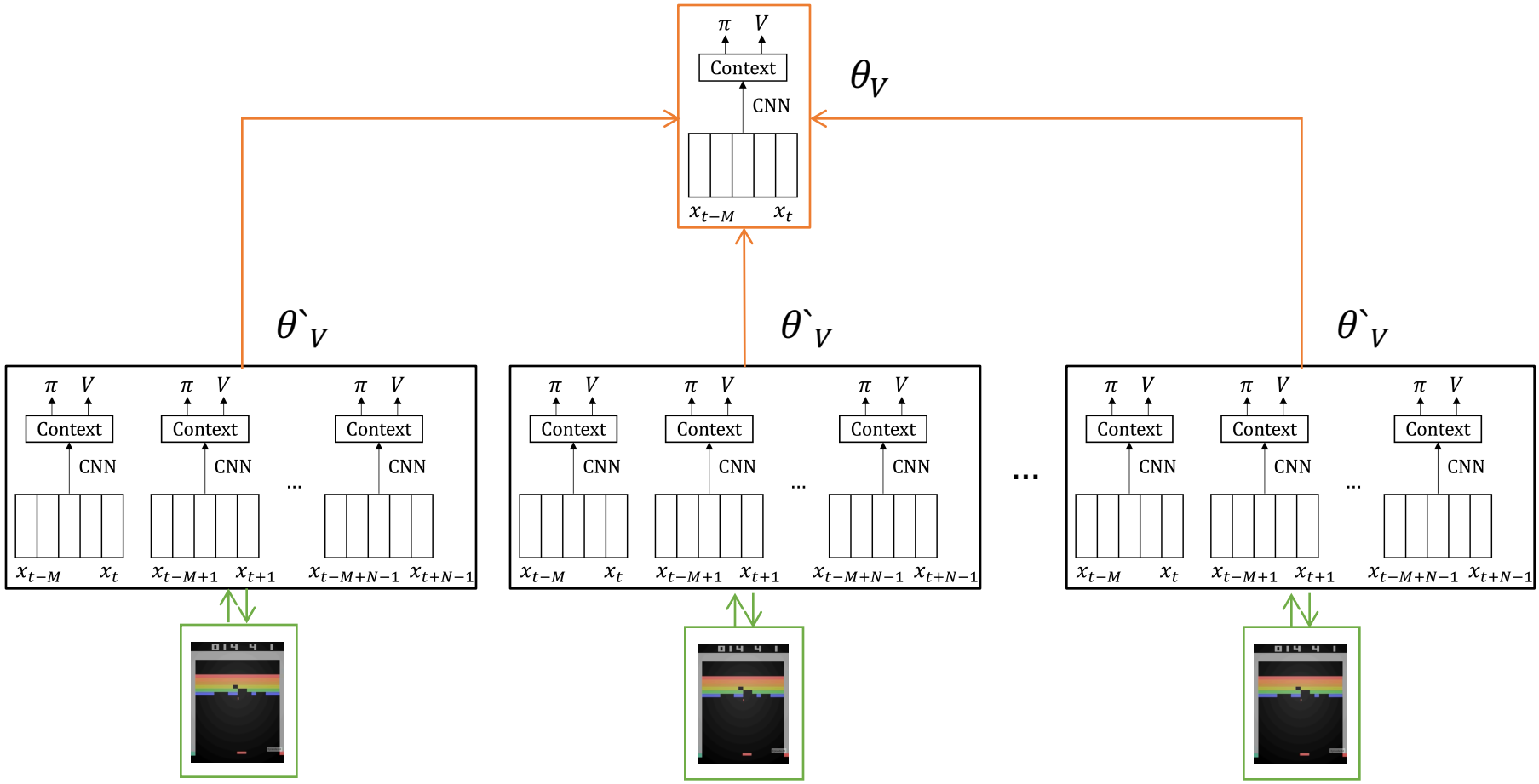

...



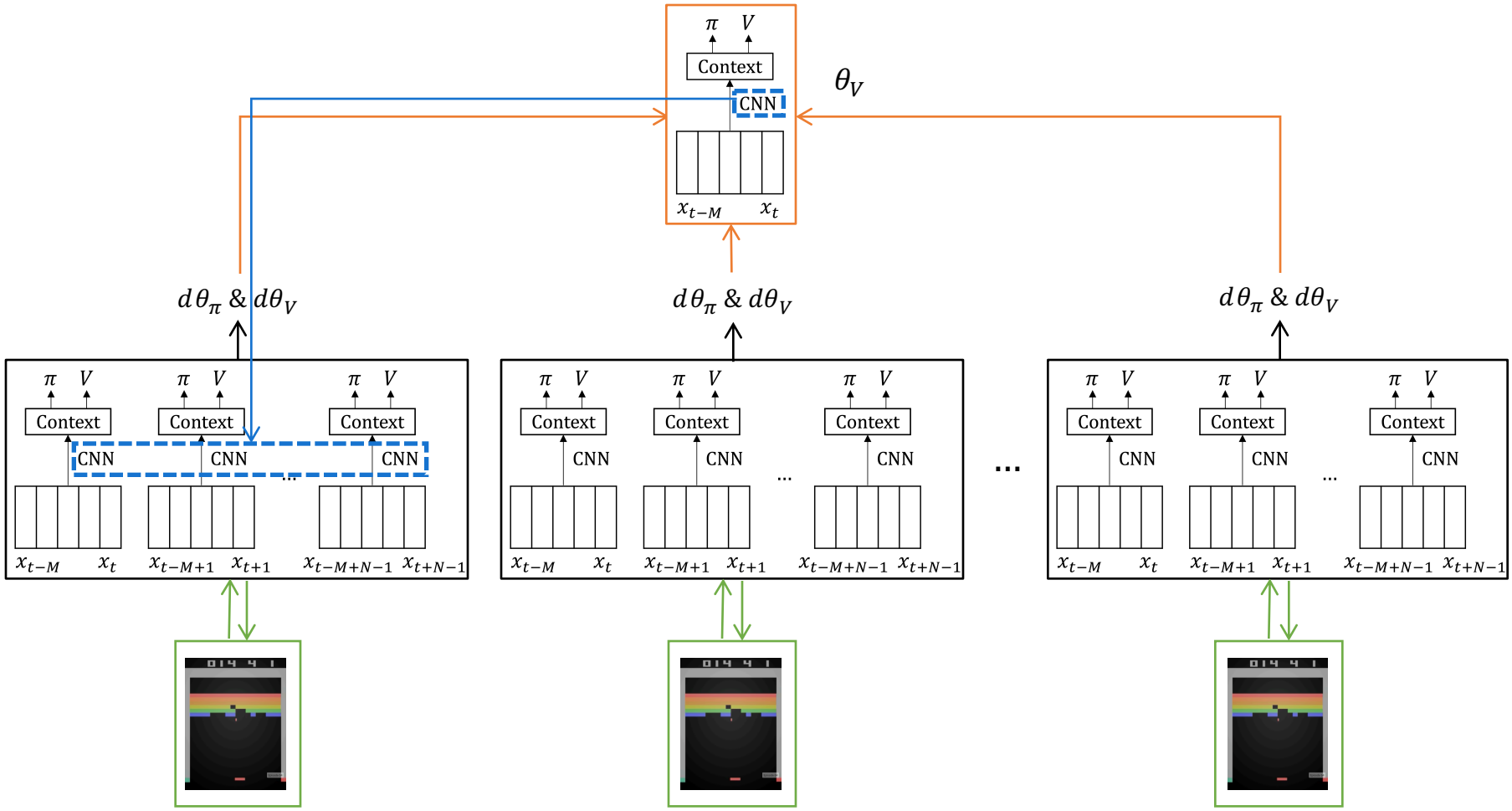

$\theta`_V$ local weight

global weight(parameter)를 관리하는 actor-critic network는 하나

# Asynchronous gradient update



# Asynchronous n-step actor-critic

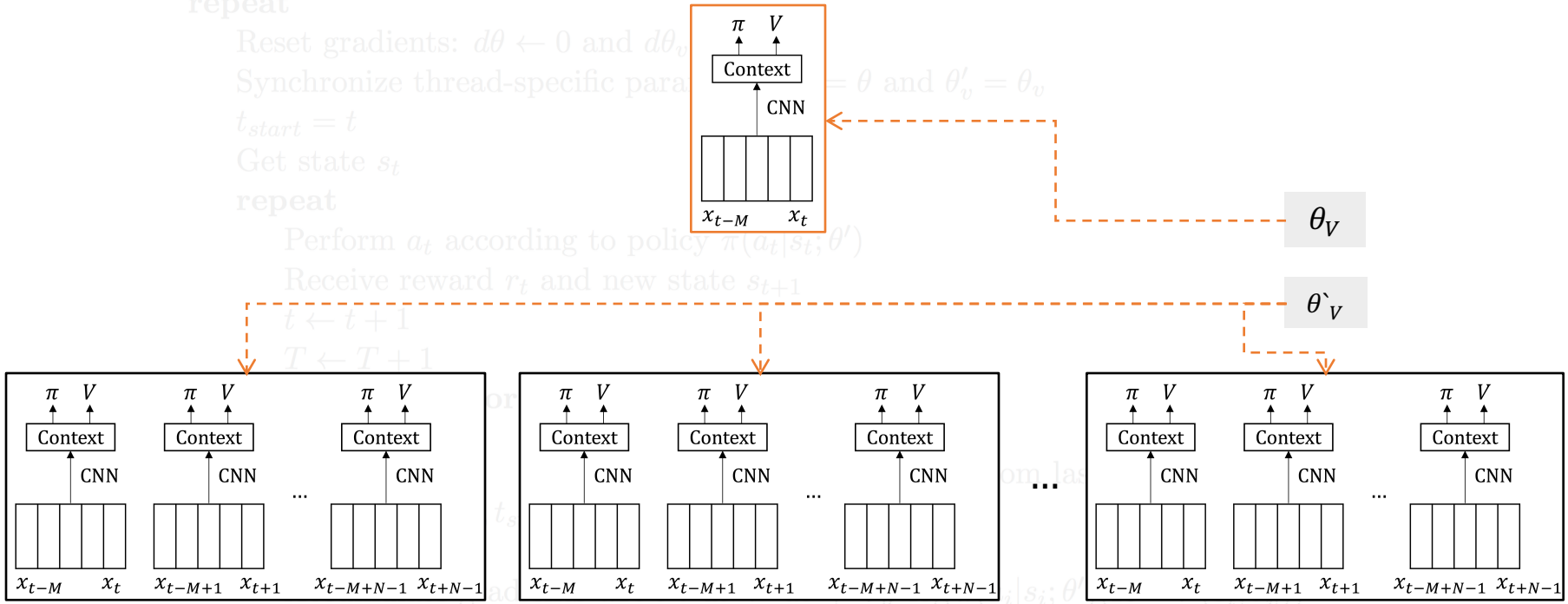


**Algorithm 3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

*// Assume global shared parameter vectors* $\theta$ *and* $\theta_v$ *and global shared counter* $T = 0$
*// Assume thread-specific parameter vectors* $\theta'$ *and* $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
**repeat**
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state $s_t$
  **repeat**
    Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
    Receive reward $r_t$ and new state $s_{t+1}$
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  **until** terminal $s_t$ **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  **for** $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  **end for**
  Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
**until** $T > T_{max}$

**Algorithm 3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

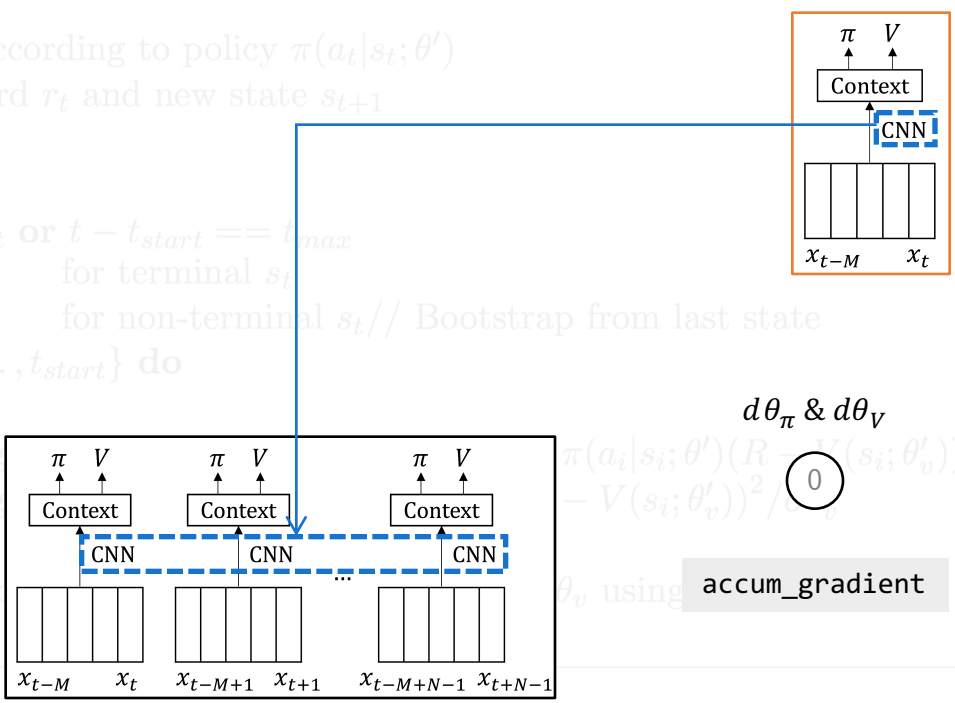
*// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$*
*// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$*

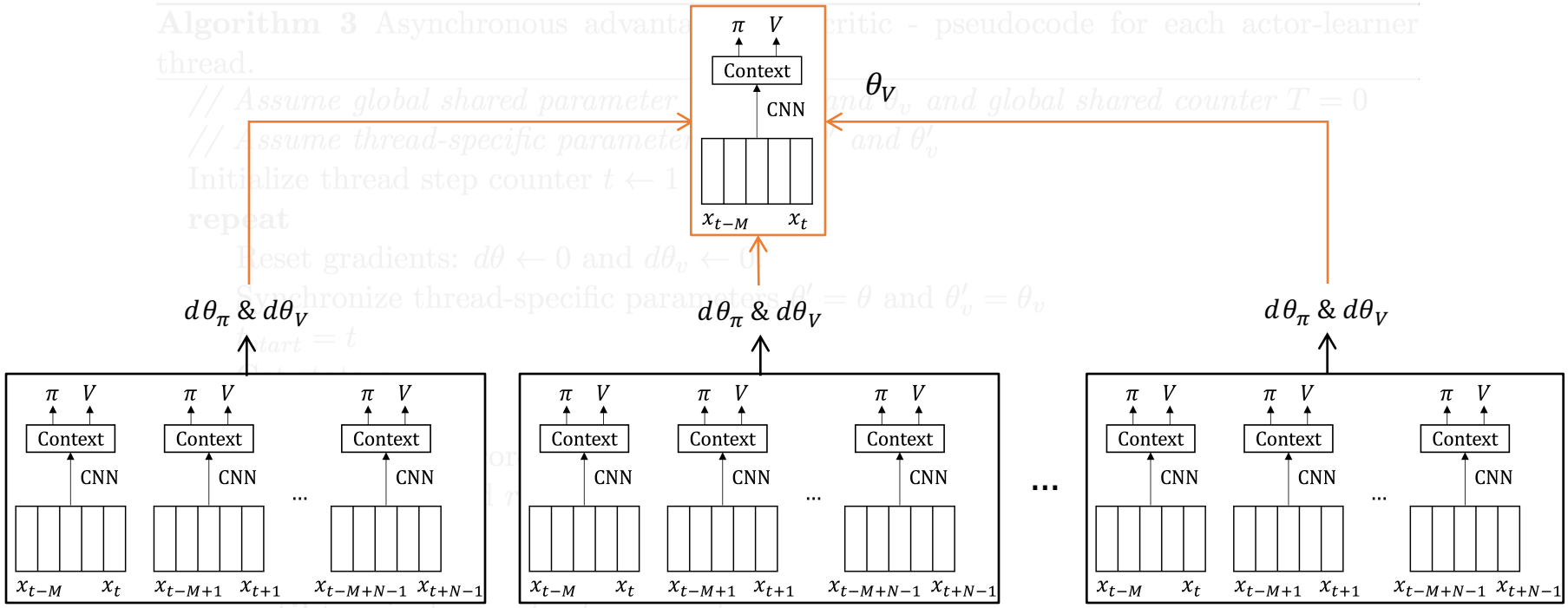


**repeat**

Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$



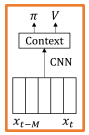



**for** $i \in \{t-1, \ldots, t_{start}\}$ **do**
  $R \leftarrow r_i + \gamma R$
  Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
  Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
**end for**
Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.

```
import tensorflow as tf
```

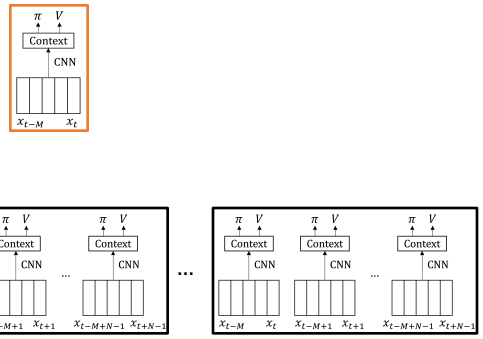
# Global network를 만들고

```
from threading import Thread

global_network = make_network(name='A3C_global’)
```



# Local network을 만들고

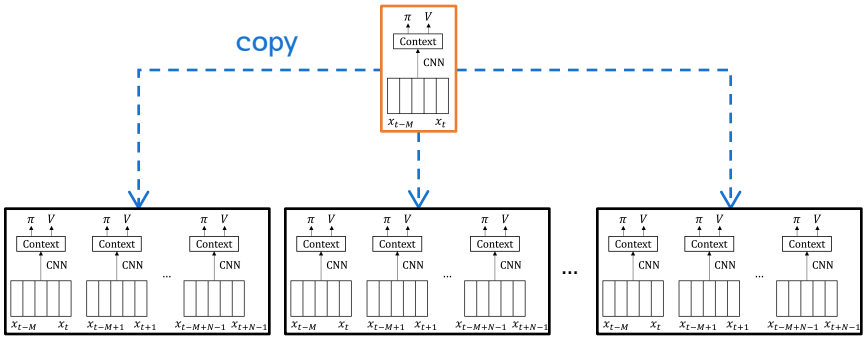
```
for worker_id in range(n_worker):
  with tf.variable_scope('thread_%d' % worker_id) as scope:
    for step in range(n_step):
      network = Network(global_network, name='A3C_%d' % (worker_id))
      networks.append(network)
      tf.get_variable_scope().reuse_variables()
  A3C_FFs[worker_id] = A3C_FF(worker_id, networks, global_network)
```

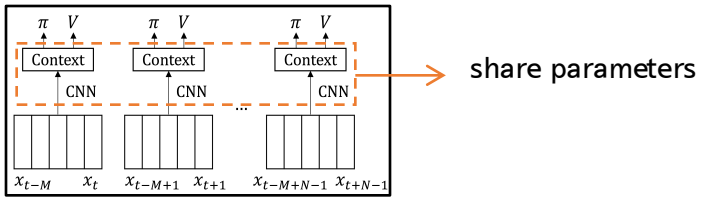


Global network의 $\theta_\pi, \theta_V$를

Local network로 **복사한다**

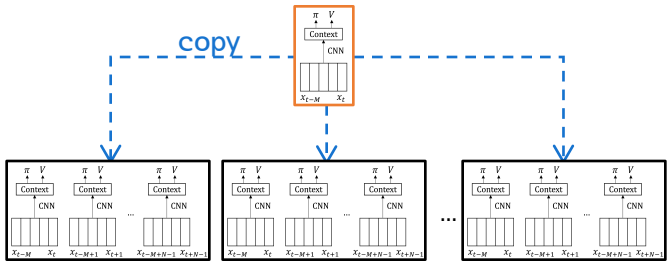
```
for worker_id in range(n_worker):
  A3C_FFs[worker_id].networks[0].copy_from_global()

# weight sharing하기 때문에 0번째 step에 해당하는 network만 하면 됨
```



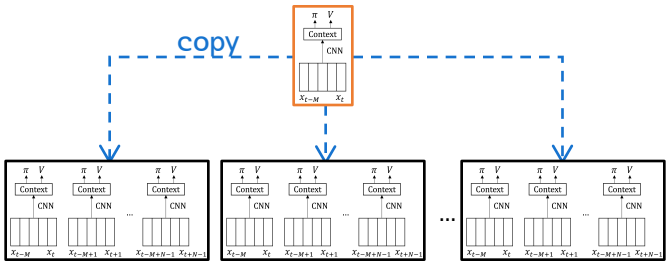
# $\theta_V$를 $\theta`_V$로 assign() 하는 op을 만들고

```
class Network(object):
  def __init__(self, sess):
    with tf.variable_scope('copy_from_target'):
      copy_ops = []
      for name in self.w.keys(): # w: network의 variable를 가진 dict
        copy_op = self.w[name].assign(global_network.w[name])
        copy_ops.append(copy_op)
    self.global_copy_op = tf.group(*copy_ops, name='global_copy_op')
```



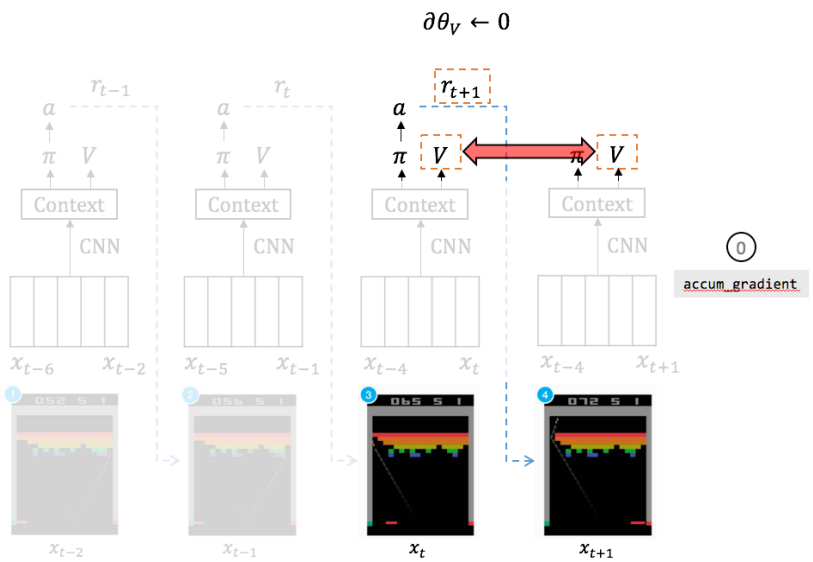
# copy_from_global()는 그 op을 실행

```
class Network(object):
  def copy_from_global(self):
    self.sess.run(self.global_copy_op)
```



# $optimizer$를 정의하고 $\partial loss/\partial \theta`_V$를 계산한 후

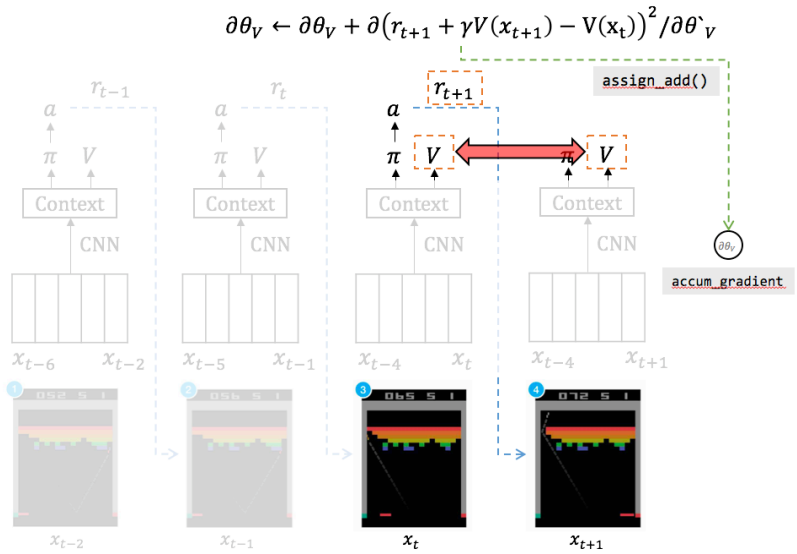
```
global_optim = tf.train.RMSPropOptimizer(learning_rate)

grads_and_vars = self.global_optim.compute_gradients(

                                        network.total_loss, network.w.values())
```



# $\partial loss/\partial \theta`_v$를 누적해서 저장할

# $Variable$을 만들고, assign_add()를 추가

```
for grad, var in tuple(grads_and_vars):

  self.accum_grads[name] = tf.Variable(tf.zeros(shape), trainable=False)

  new_grads_and_vars.append((tf.clip_by_norm(

      self.accum_grads[name].ref(), self.max_grad_norm), global_v))
```



# sess.partial_run_setup로

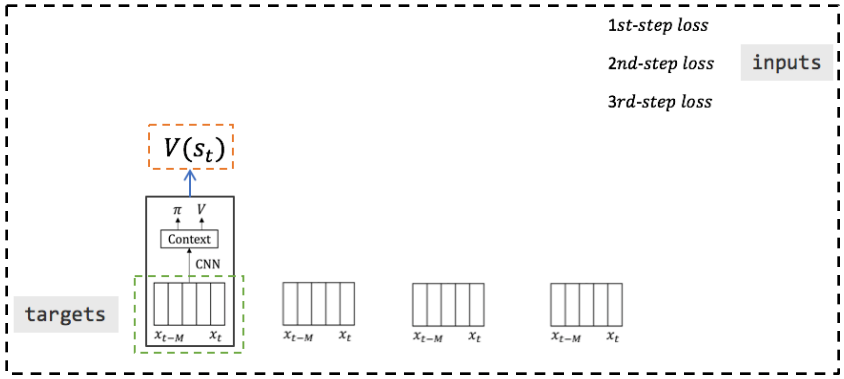
# partial_graph를 정의하고

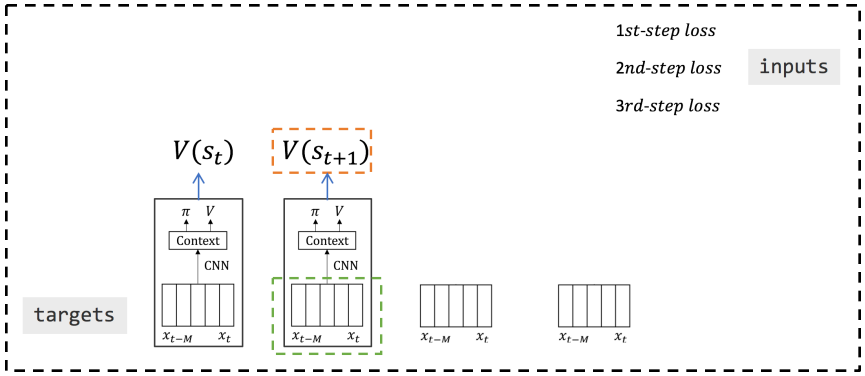
```
targets = [network.sampled_action for network in self.networks]
targets.extend([network.log_policy_of_sampled_action for network in self.networks])
targets.extend([network.value for network in self.networks])
targets.extend(self.add_accum_grads.values())
targets.append(self.fake_apply_gradient)

inputs = [network.s_t for network in self.networks]
inputs.extend([network.R for network in self.networks])
inputs.extend([network.true_log_policy for network in self.networks])
inputs.append(self.learning_rate_op)

self.partial_graph = self.sess.partial_run_setup(targets, inputs)
```

# sess.partial_run()으로

# 중간값을 유지하면서 action을 계산하면서

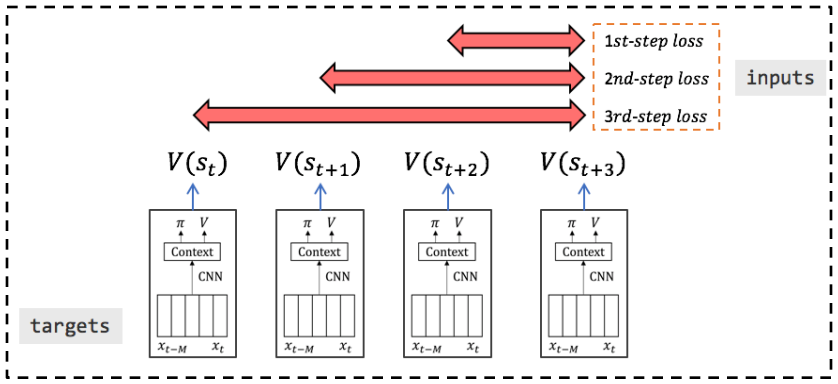
```
action, log_policy = self.sess.partial_run(
  self.partial_graph, [
    self.networks[network_idx].sampled_action,
    self.networks[network_idx].log_policy_of_sampled_action,
  ], {
    self.networks[network_idx].s_t: [s_t],
  }
)
```



# 저장된 중간값으로 gradient update를 합니다

```
self.sess.partial_run(self.partial_graph,
        [self.add_accum_grads[t] for t in range(len(self.prev_r) - 1)], data)

self.sess.run(self.apply_gradient, {self.learning_rate_op: lr})

# 3. Reset accumulated gradients to zero
self.sess.run(self.reset_accum_grad)
```



# 쓰레드로 실행된 함수를 정의하고

```
def worker_func(worker_id):
  model = A3C_FFs[worker_id]
  if worker_id == 0:
    model.train_with_log(global_t,
        saver, writer, checkpoint_dir, assign_global_t_op)
  else:
    model.train(global_t)
```

# 쓰레드 start() join() 하면 끝.

```
workers = []
for idx in range(config.n_worker):
  workers.append(Thread(target=worker_func, args=(idx,)))

# Execute and wait for the end of the training
for worker in workers:
  worker.start()

for worker in workers:
  worker.join()
```

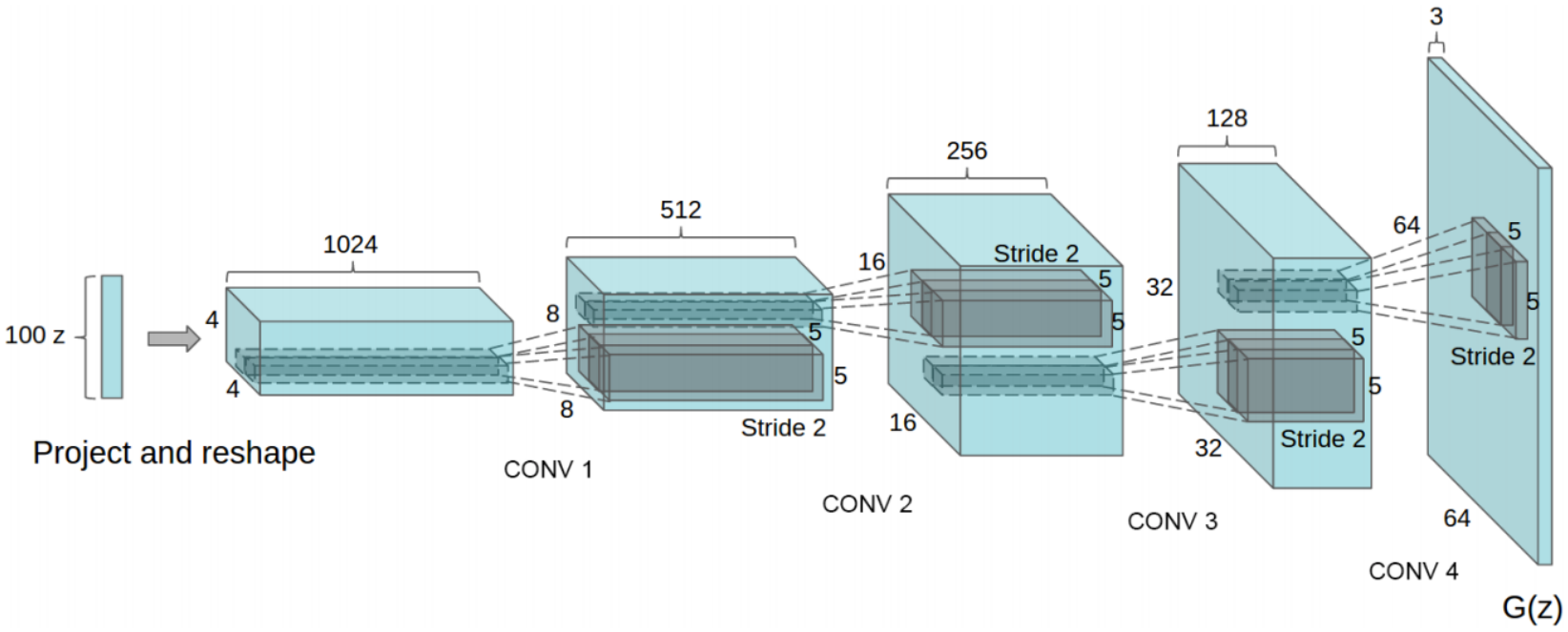
# Resources

- Paper “Asynchronous Methods for Deep Reinforcement Learning”
-  Chainer 구현: https://github.com/muupan/async-rl
- 읽기 쉬운 Deep Q-network 소개글
  - http://sanghyukchun.github.io/90/
  - Demystifying Deep Reinforcement Learning
- Reinforcement Learning를 **진지**하게 공부하고 싶다면
  - David Silver’s course
  - John Schulman’s CS294 course
  - Sutton’s “Reinforcement Learning: An Introduction”

# Resources

- OpenAI가 제안한 문제들 : Requests for Research
  - Improved Q-learning with continuous actions
  - Multitask RL with continuous actions.
  - Parallel TRPO
- 주목할만한 keywords
  - Continuous actions: [1], [2]
  - Multiagents: [3]
  - Hierarchical reinforcement learning: [4]
  - Better exploration: [5]

# 아쉽지만..

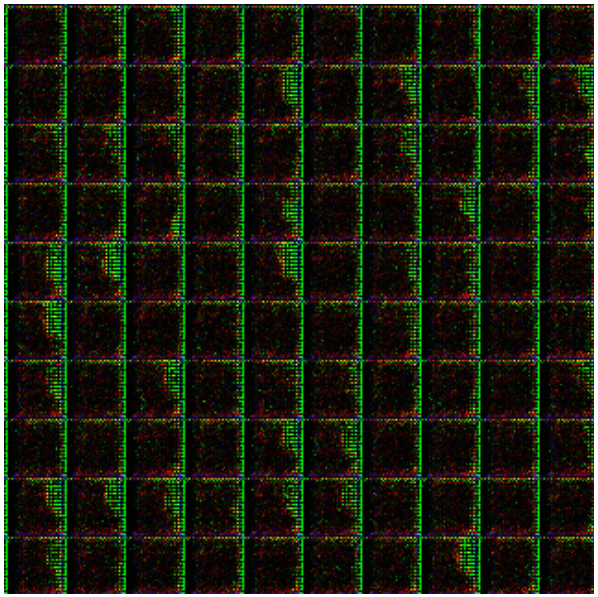



Deep Convolutional Generative Adversarial Networks

https://github.com/carpedm20/DCGAN-tensorflow

OpenAI's blog post, Generative Models

# 아쉽지만..

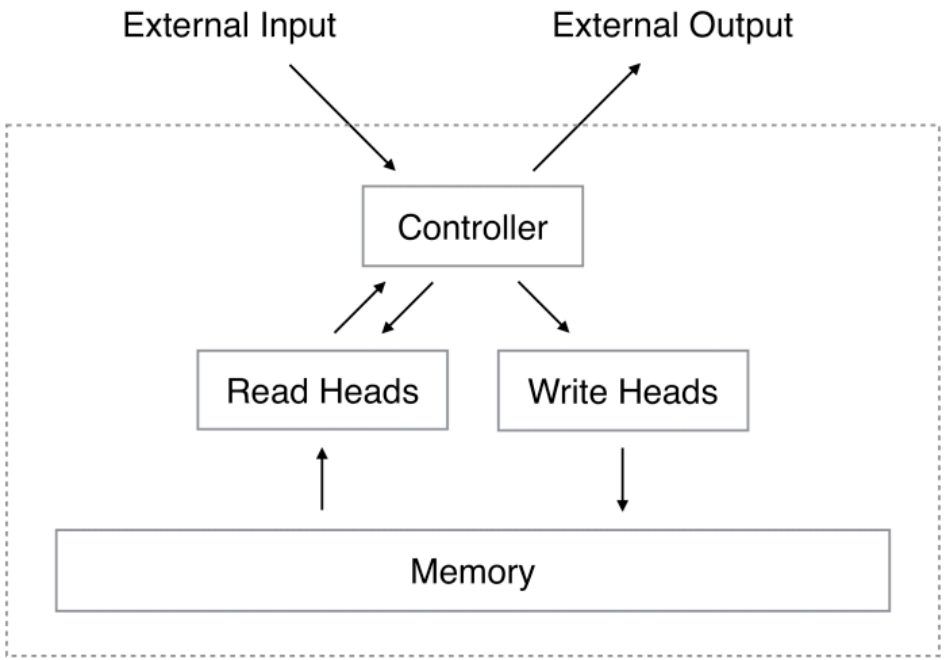


*"What I cannot create, I do not understand."*

—Richard Feynman

OpenAI's blog post, Generative Models

# 아쉽지만..




Neural Turing Machine

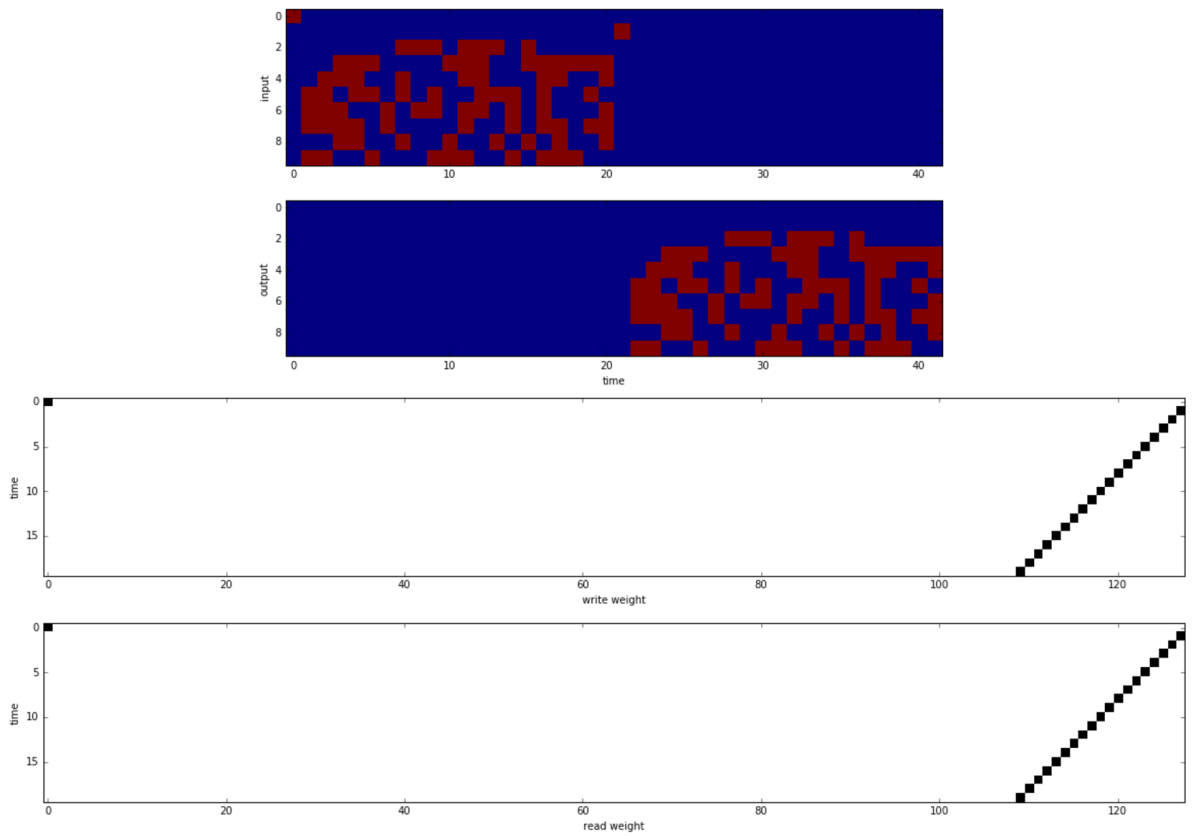
https://github.com/carpedm20/NTM-tensorflow

http://cpmarkchang.logdown.com/posts/279710-neural-network-neural-turing-machine

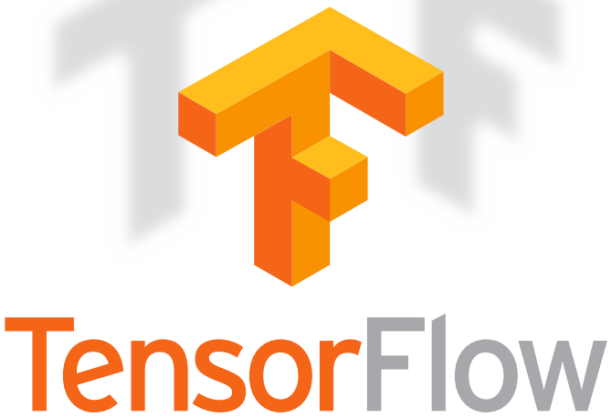
# 아쉽지만..

```
In [33]: plot(ntm, 20, sess)
```

```
Loss : 0.000009
```




http://cpmarkchang.logdown.com/posts/279710-neural-network-neural-turing-machine


TensorFlow

```
import tensorflow as tf
```

감사합니다

# Q&A

http://carpedm20.github.io/