

ML_9

Team BMS



INDEX

ML Study
9 Week

Index 01. Ice Breaking

Ice Breaking

Index 02. Chapter 14

Recurrent Neural Network

Index 03. Chapter 15

Autoencoder

Index 04. Chapter 16

Reinforcement Learning

Index 05. Before Finish Class

Before Finish Class



Ice Breaking

01 I. Ice Breaking

Ice Breaking



01 I. Ice Breaking

Ice Breaking

난이도가 높은 조합이에요~ : 벤피 개좆망함

근데 후반가면 파괴력이~ : 후반 못감

특정 조건 하에서는 : 상대방이 뇌없는 바보 천치일때
는

대각선의 법칙 : lck의 규칙

좋아요 좋습니다 : 곧 나빠질 것

이대로 사이드 주도권 가지고 : 만약 신이 도와 본대가
안 터진다면

해석의 차이 : 한쪽 생각이 개병신

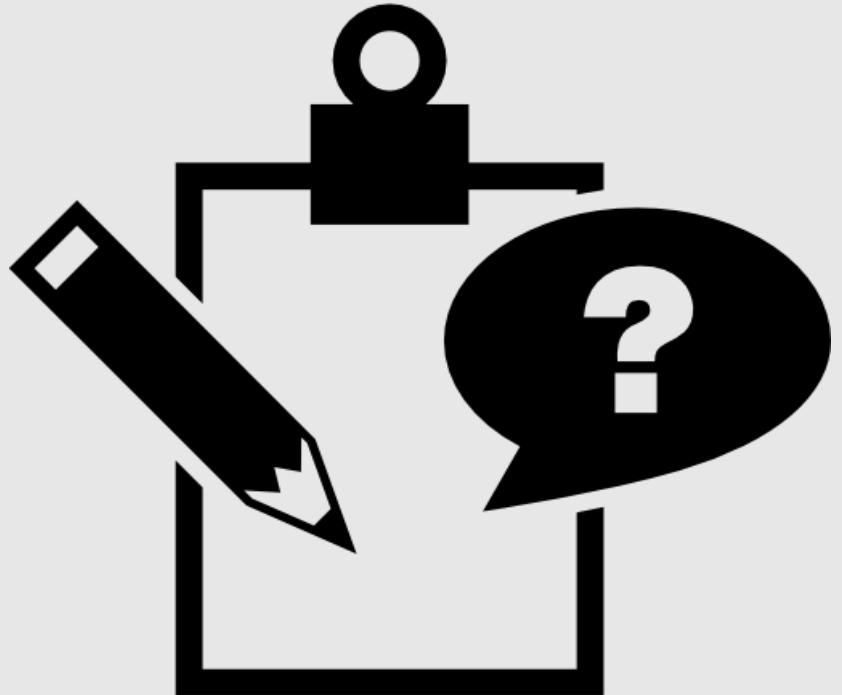
기분 나쁜 교환 구도 : 일방적 손해

대충 이정도로 생각하고 들으면 말하는대로 되더라

01 I. Ice Breaking

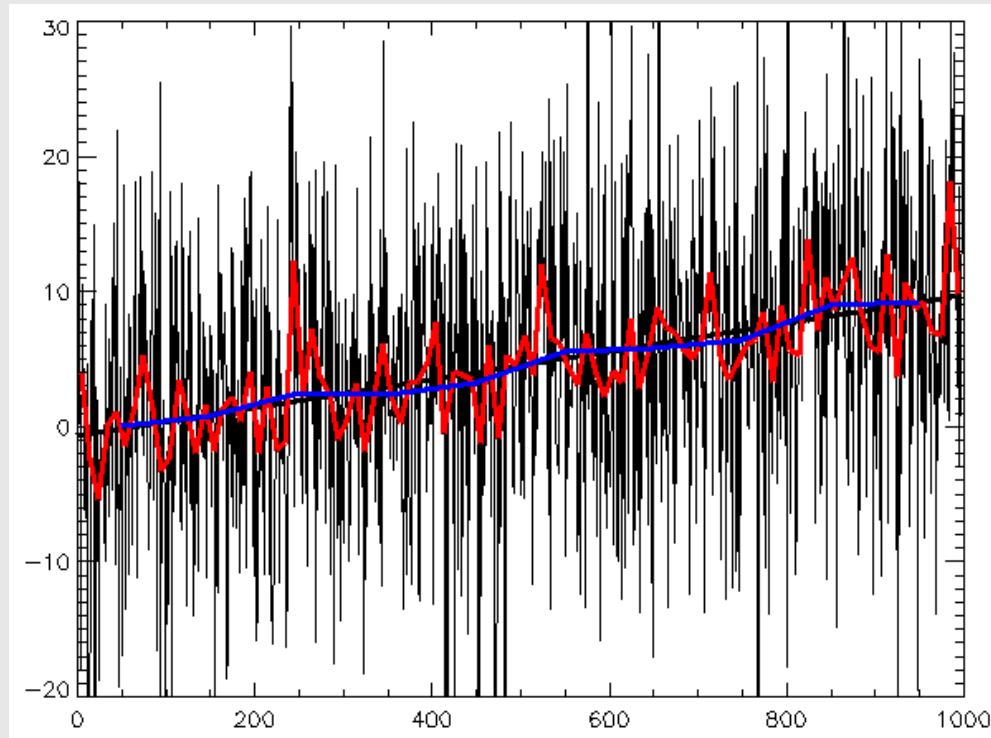
Before Start





Recurrent Neural Network

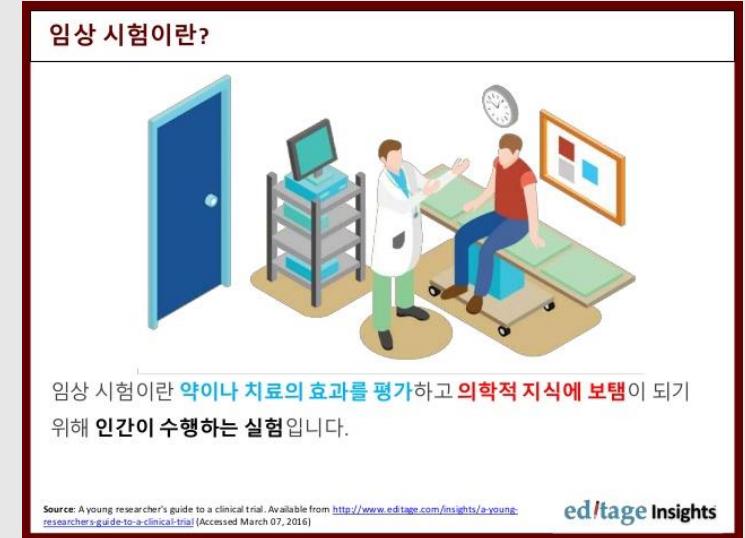
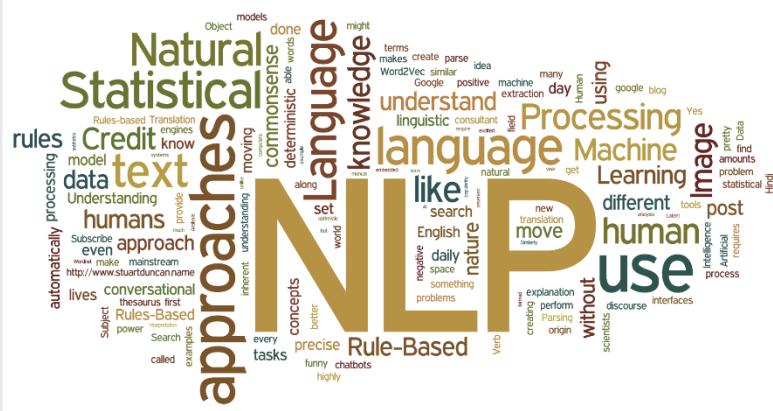
Time Series



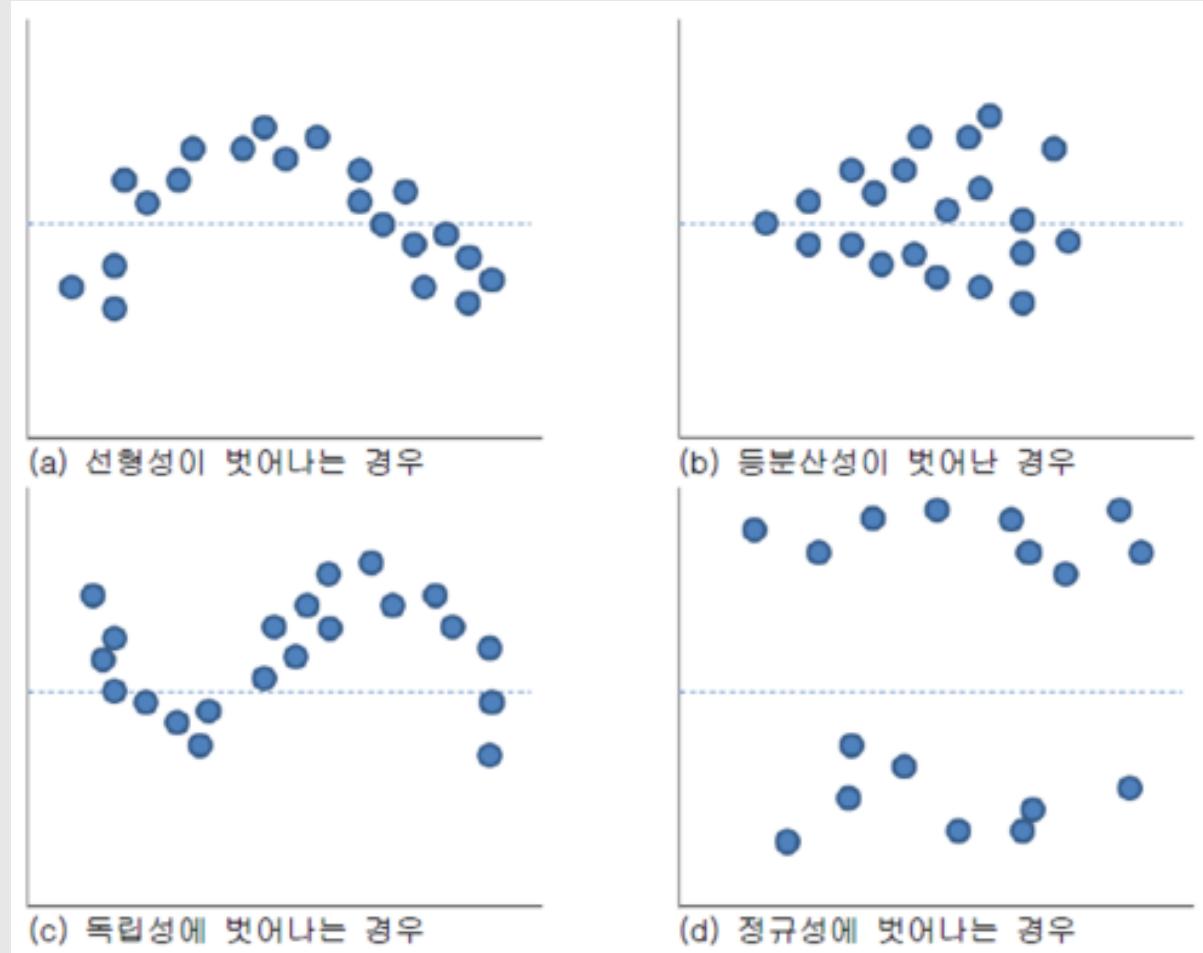
- 일정 시간 간격으로 배치된 데이터들의 수열
- Time Series Analysis:
시계열을 해석, 이해하는데 쓰이는 연구분야
- 공학, 과학계산, 금융시장의 주가 예측에
주로 사용
- 실제 응용에서 많이 쓰이는 범용 모델:
 - Autoregressive (AR) Model
 - Integrated (I) Model
 - Moving Average (MA) Model
- 선형 종속적

02 II. Recurrent Neural Network

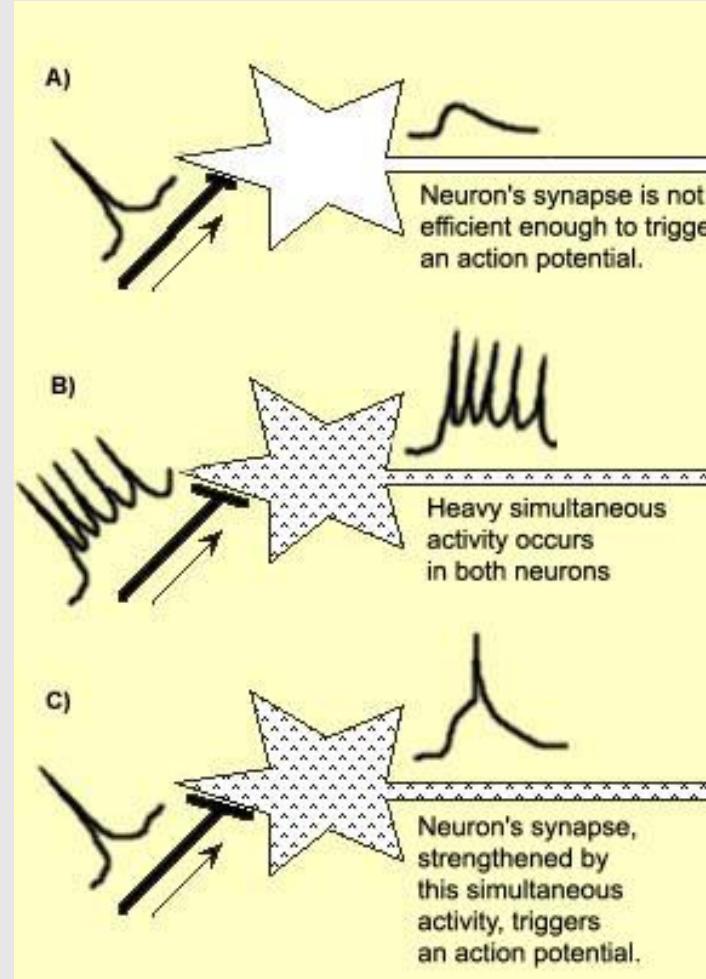
Time Series Data



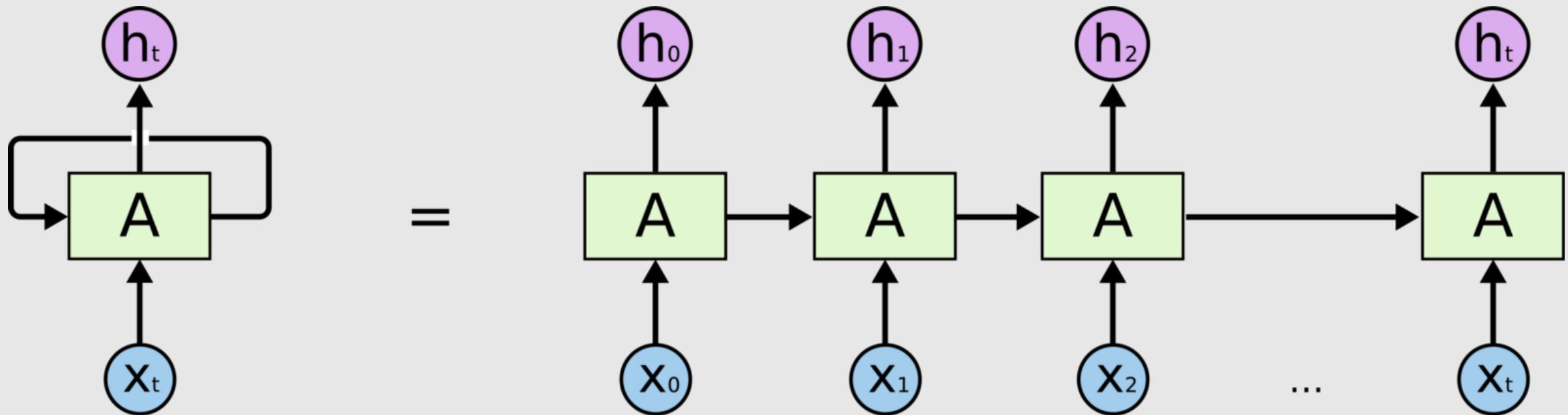
Independence



Independence

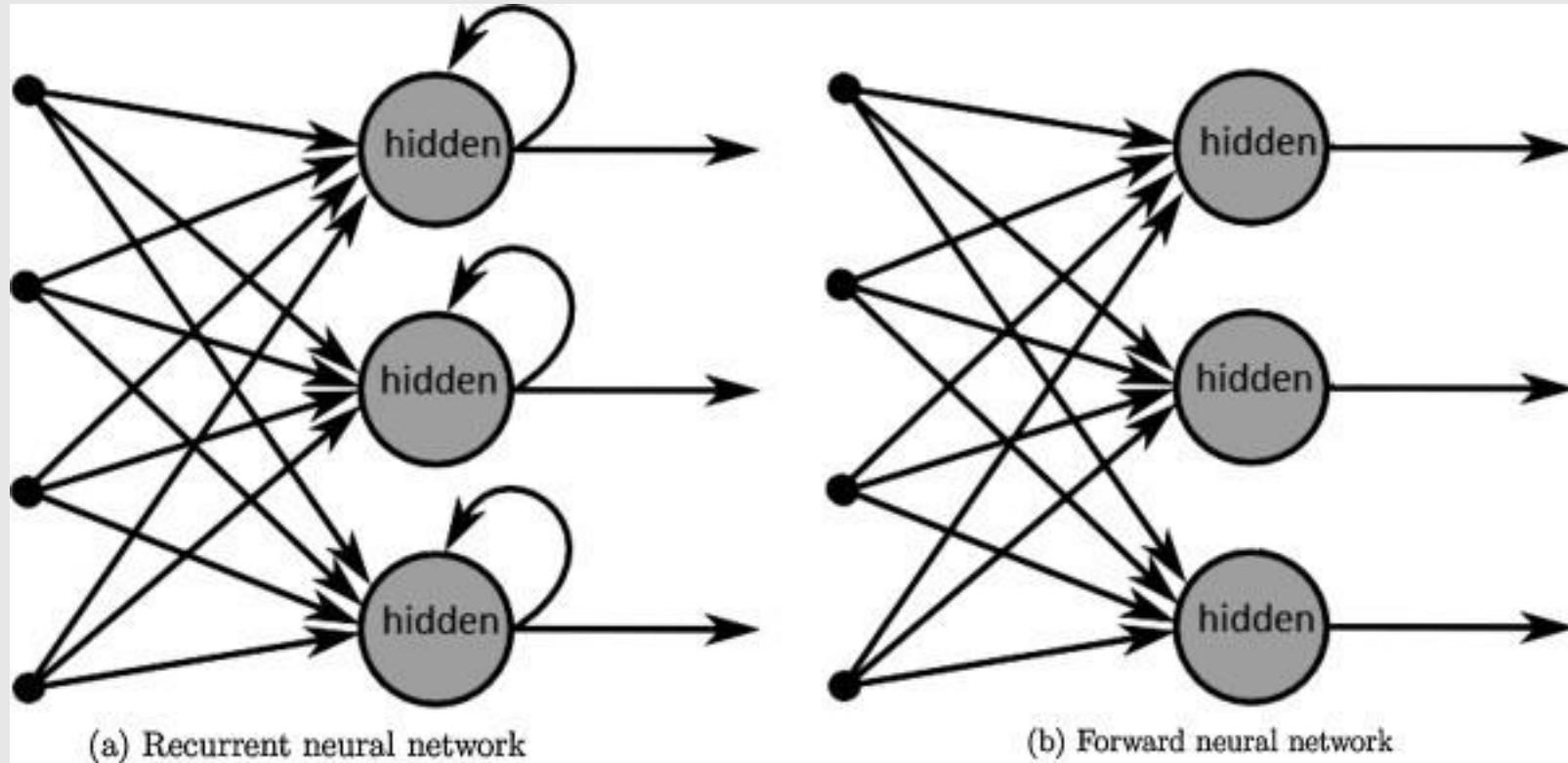


Recurrent Neural Network Architecture



<https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3>

Recurrent Neural Network Architecture

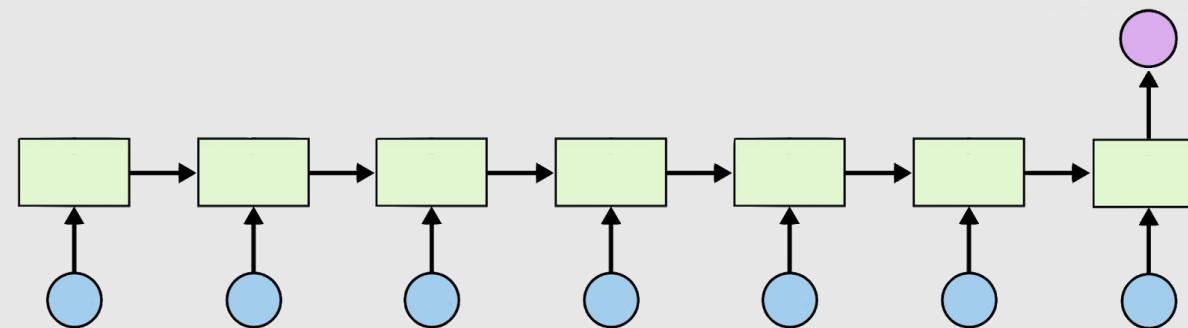
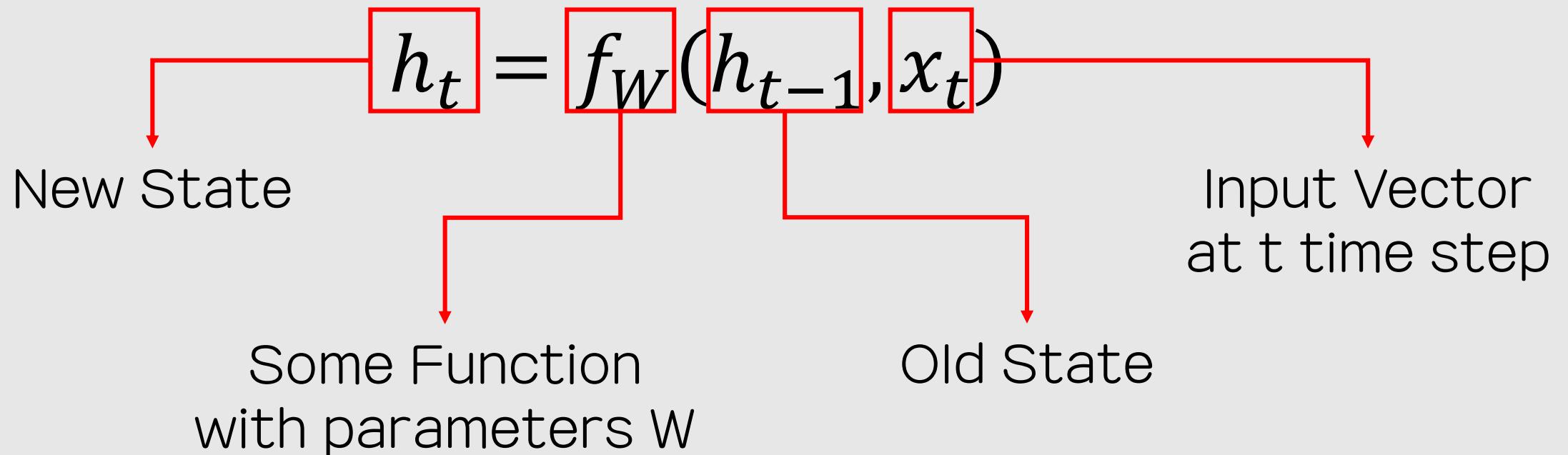


(a) Recurrent neural network

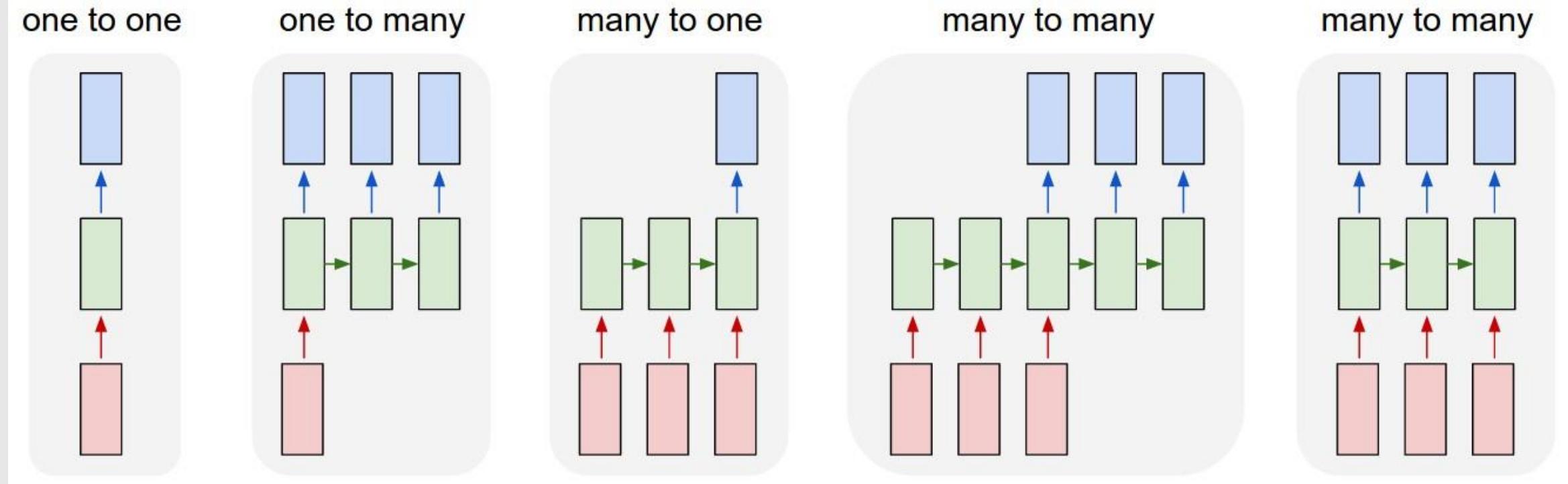
(b) Forward neural network

<https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3>

Recurrent Neural Network Architecture

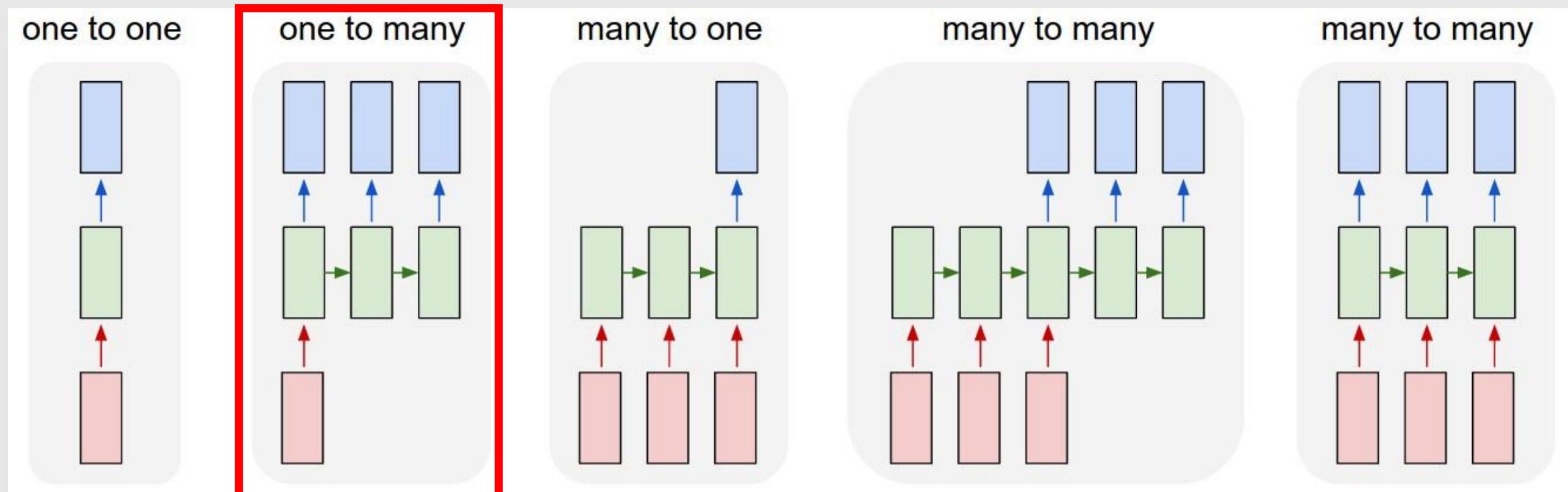


Recurrent Neural Network Architecture



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

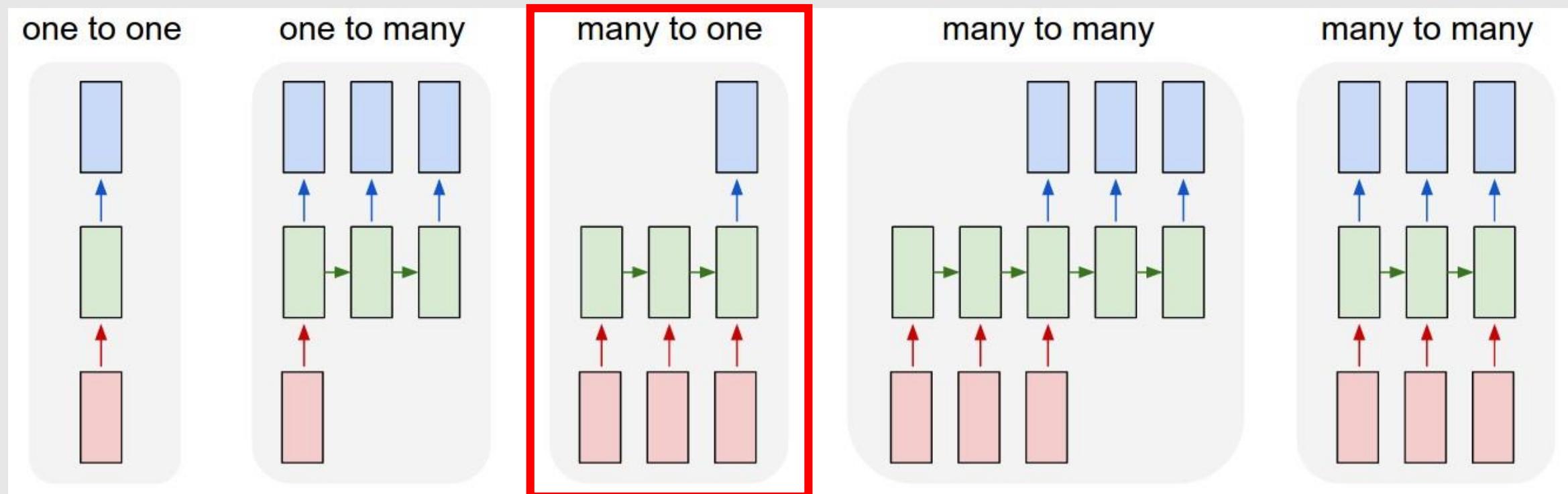
Recurrent Neural Network Architecture



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Image Captioning
→ Image to Sequence of words

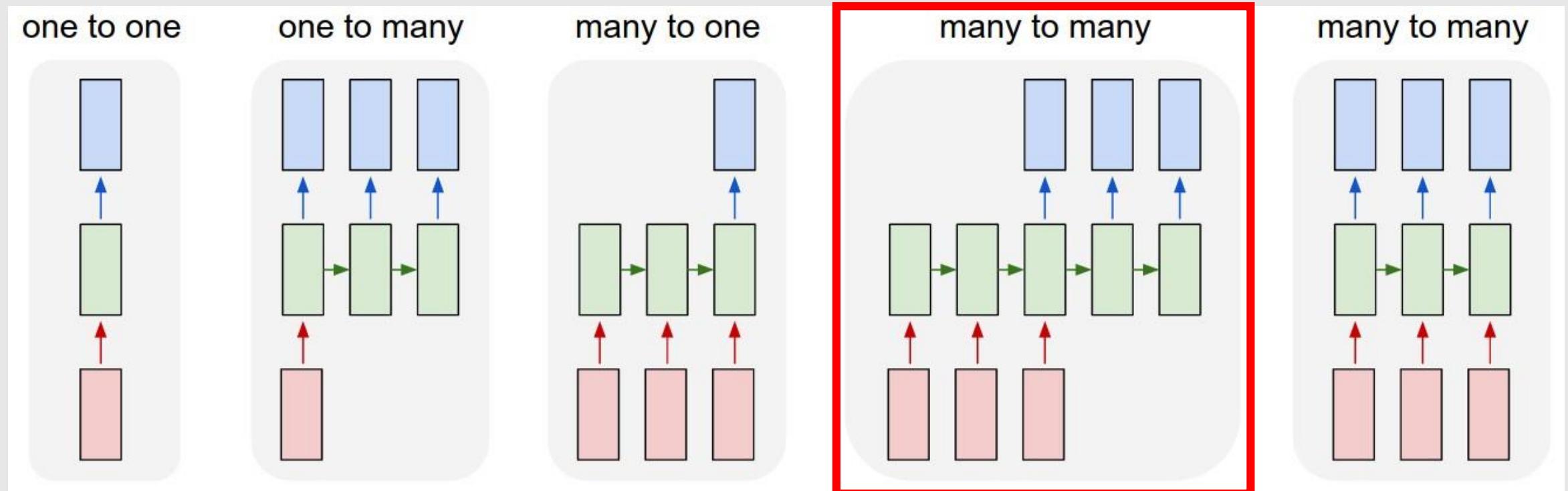
Recurrent Neural Network Architecture



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

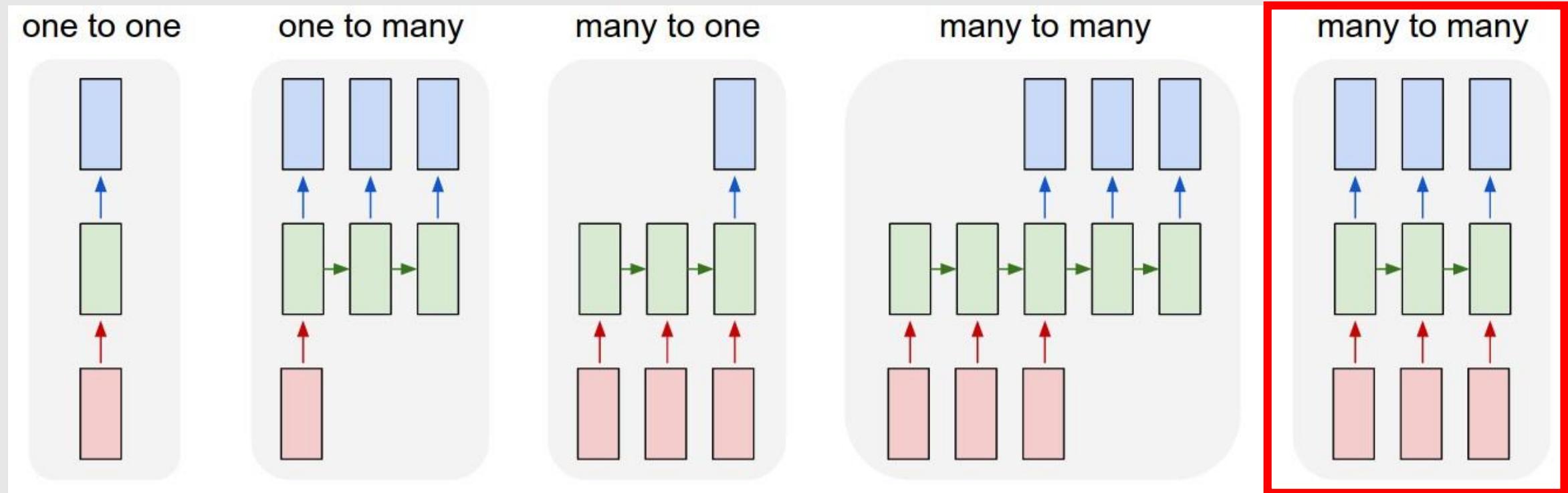
Sentiment Classification
→ Sequence of words to Sentiment

Recurrent Neural Network Architecture



Machine Translation
→ Seq of words to Seq of words

Recurrent Neural Network Architecture



<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Video Classification on
Frame level

Backpropagation Through Time (BPTT)

이제 RNNs 학습을 위해서 실제로 사용하는 **Backpropagation Through Time(BPTT)** 알고리즘을 살펴보자. BPTT에서 hidden layer에서 output간의 가중치인 W 는 일반 BP와 똑같은 형태로 다음과 같이 업데이트 할 수 있다.

$$W(t+1) = W(t) + \eta s(t)e_o(t)^T \quad (10)$$

하지만, Input Layer에서 Hidden Layer의 가중치 V 와 Hidden Layer에서 Hidden Layer로 순환되는 경로의 가중치인 U 는 다음과 같은 일정 시간에서의 에러 값들을 계산하고

$$e_h(t-\tau-1) = d_h(e_h(t-\tau)U, t-\tau-1) \quad (11)$$

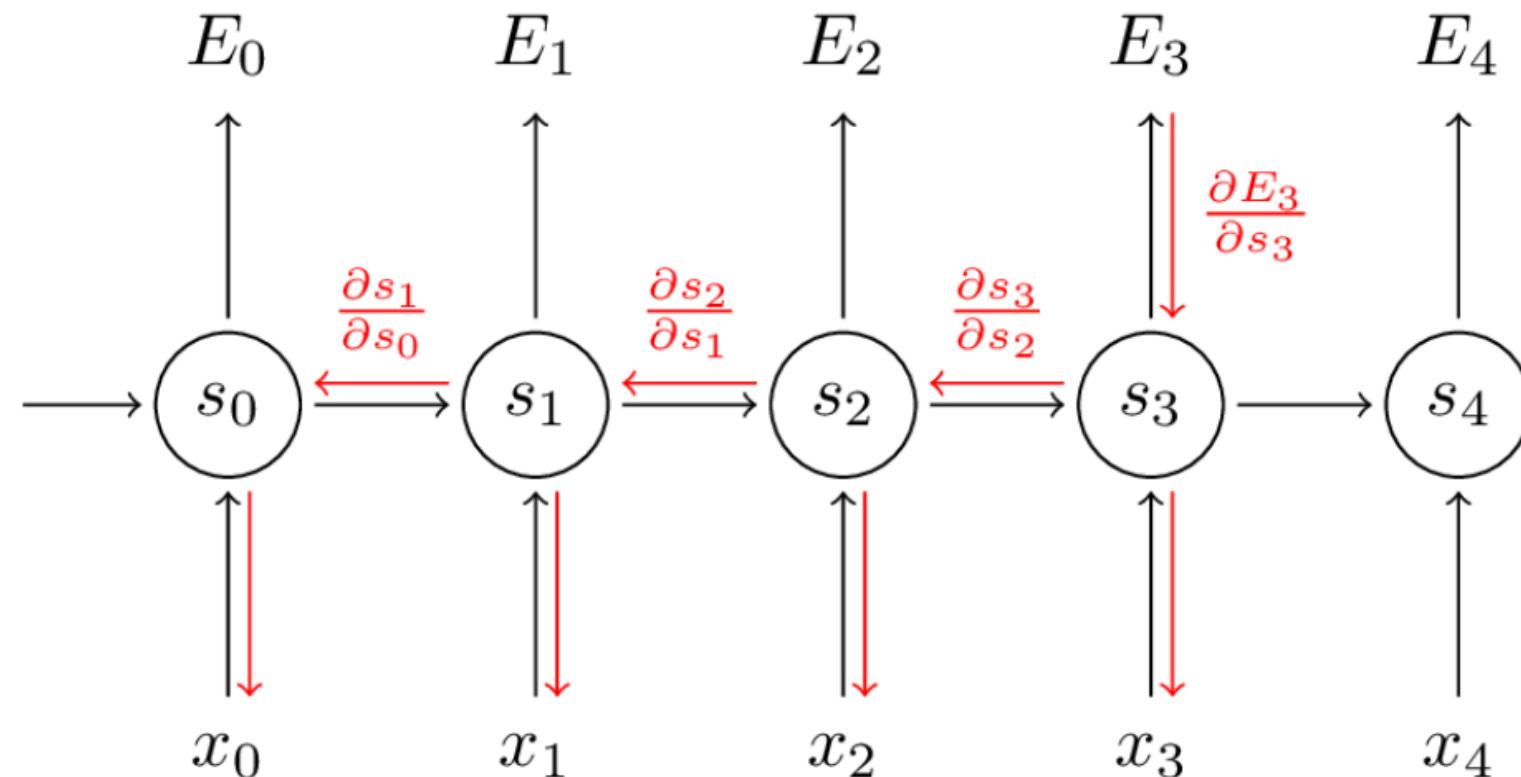
일정 시간 동안 에러 값을 합한 값을 역전파함으로써 Input Layer에서 Hidden Layer의 가중치 V 와 Hidden Layer에서 Hidden Layer로 순환되는 경로의 가중치인 U 를 업데이트한다.

$$V(t+1) = V(t) + \eta \sum_{z=0}^T x(t-z)e_h(t-z)^T \quad (12)$$

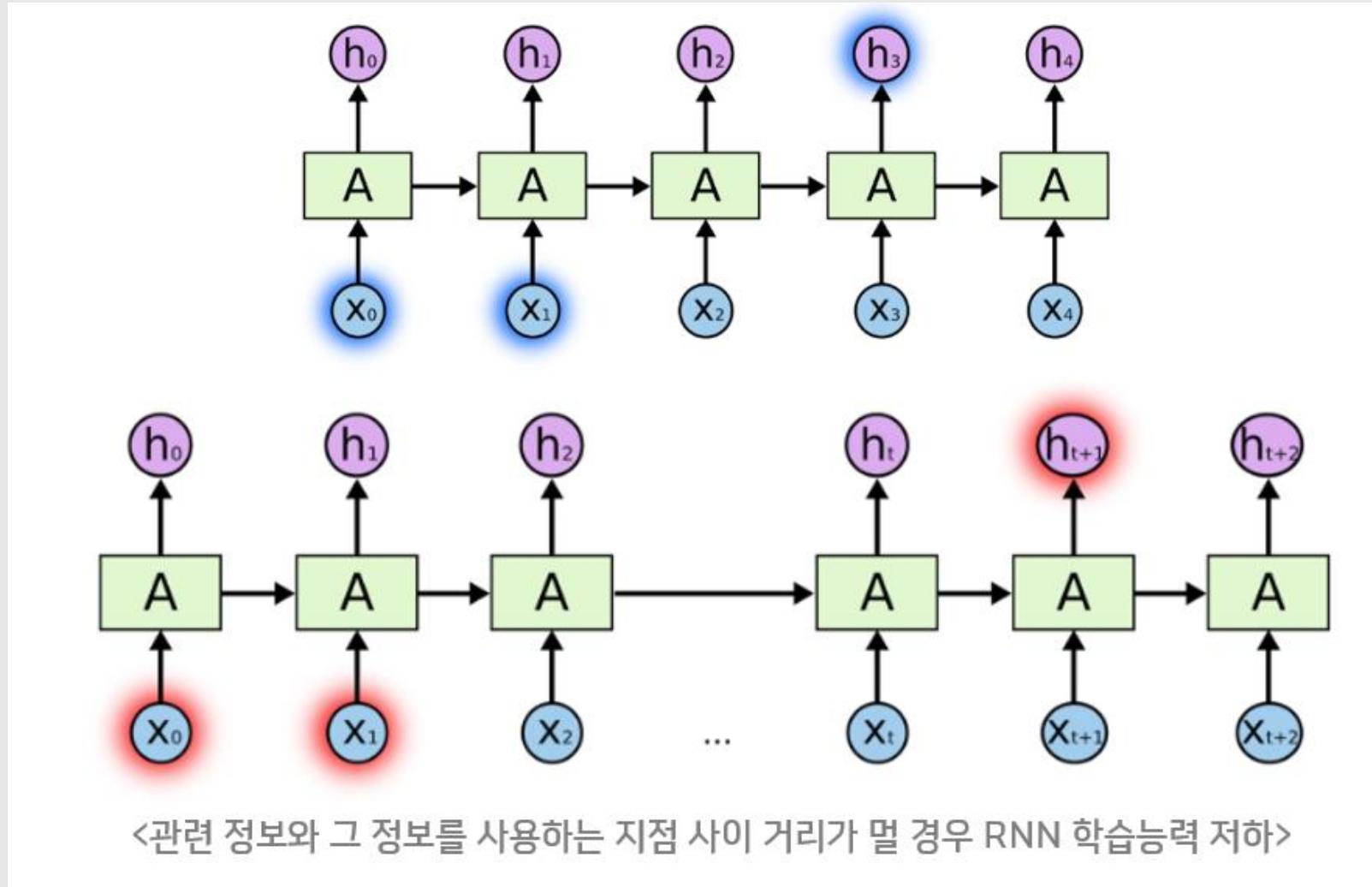
$$U(t+1) = U(t) + \eta \sum_{z=0}^T s(t-z-1)e_h(t-z)^T \quad (13)$$

Backpropagation Through Time (BPTT)

이렇게 업데이트를 진행하면, 현재 시간의 에러를 과거 시간의 상태까지 역전파할 수 있다. 이를 그림으로 나타내면 아래와 같다.



Long-Term Dependency



Long Short-term Memory (LSTM)

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

Sepp Hochreiter Fakultät für Informatik Technische Universität München 80290 München, Germany hochreit@informatik.tu-muenchen.de http://www7.informatik.tu-muenchen.de/~hochreit	Jürgen Schmidhuber IDSIA Corso Elvezia 36 6900 Lugano, Switzerland juergen@idsia.ch http://www.idsia.ch/~juergen
---	--

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

1 INTRODUCTION

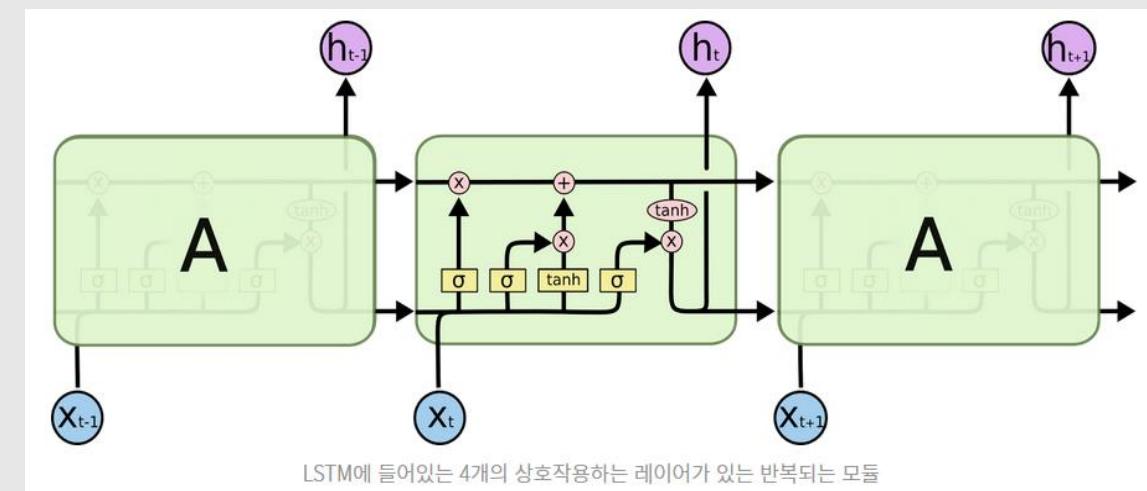
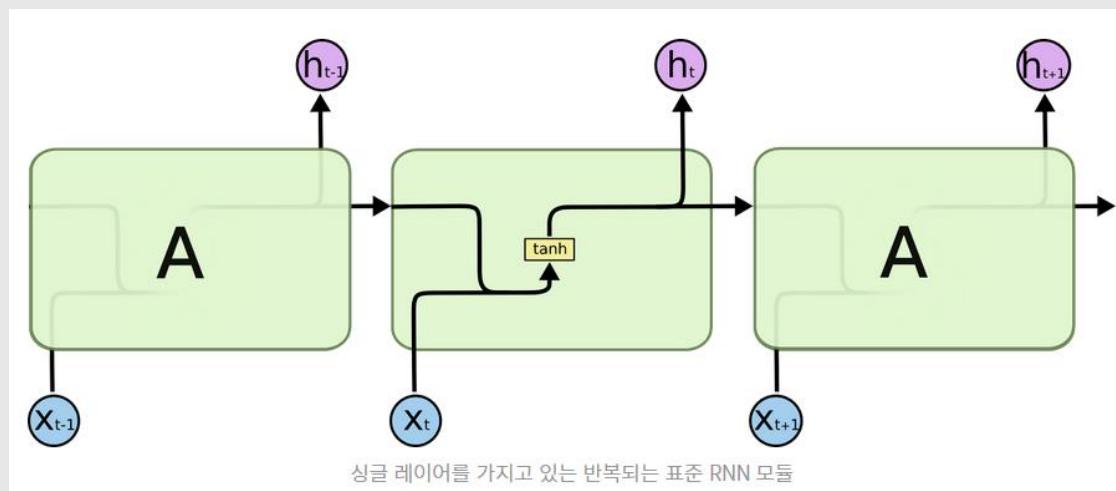
Recurrent networks can in principle use their feedback connections to store representations of recent input events in form of activations ("short-term memory", as opposed to "long-term memory" embodied by slowly changing weights). This is potentially significant for many applications, including speech processing, non-Markovian control, and music composition (e.g., Mozer 1992). The most widely used algorithms for learning *what* to put in short-term memory, however, take too much time or do not work well at all, especially when minimal time lags between inputs and corresponding teacher signals are long. Although theoretically fascinating, existing methods do not provide clear *practical* advantages over, say, backprop in feedforward nets with limited time windows. This paper will review an analysis of the problem and suggest a remedy.

The problem. With conventional "Back-Propagation Through Time" (BPTT, e.g., Williams and Zipser 1992, Werbos 1988) or "Real-Time Recurrent Learning" (RTRL, e.g., Robinson and Fallside 1987), error signals "flowing backwards in time" tend to either (1) blow up or (2) vanish: the temporal evolution of the backpropagated error exponentially depends on the size of the weights (Hochreiter 1991). Case (1) may lead to oscillating weights, while in case (2) learning to bridge long time lags takes a prohibitive amount of time, or does not work at all (see section 3).

The remedy. This paper presents "*Long Short-Term Memory*" (LSTM), a novel recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm. LSTM is designed to overcome these error back-flow problems. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag capabilities. This is achieved by an efficient, gradient-based algorithm for an architecture

- 장기 의존성 문제 해결을 위해 명시적으로 디자인
- 오랜 기간 동안 정보 기억 가능
- GRU와 더불어 굉장히 대중적

Long Short-term Memory (LSTM)



<<https://brunch.co.kr/@chris-song/9>>

Gated Recurrent Unit (GRU)



Gated Recurrent Unit (GRU)

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho
 Bart van Merriënboer Caglar Gulcehre Dzmitry Bahdanau
 Université de Montréal Jacobs University, Germany
 firstname.lastname@umontreal.ca d.bahdanau@jacobs-university.de

Fethi Bougares Holger Schwenk Yoshua Bengio
 Université du Maine, France Université de Montréal, CIFAR Senior Fellow
 firstname.lastname@lum.univ-lorraine.fr find.me.on.the.web

Abstract

In this paper, we propose a novel neural network model called RNN Encoder–Decoder that consists of two recurrent neural networks (RNN). One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of the proposed model are jointly trained to maximize the conditional probability of a target sequence given a source sequence. The performance of a statistical machine translation system is empirically found to improve by using the conditional probabilities of phrase pairs computed by the RNN Encoder–Decoder as an additional feature in the existing log-linear model. Qualitatively, we show that the proposed model learns a semantically and syntactically meaningful representation of linguistic phrases.

1 Introduction

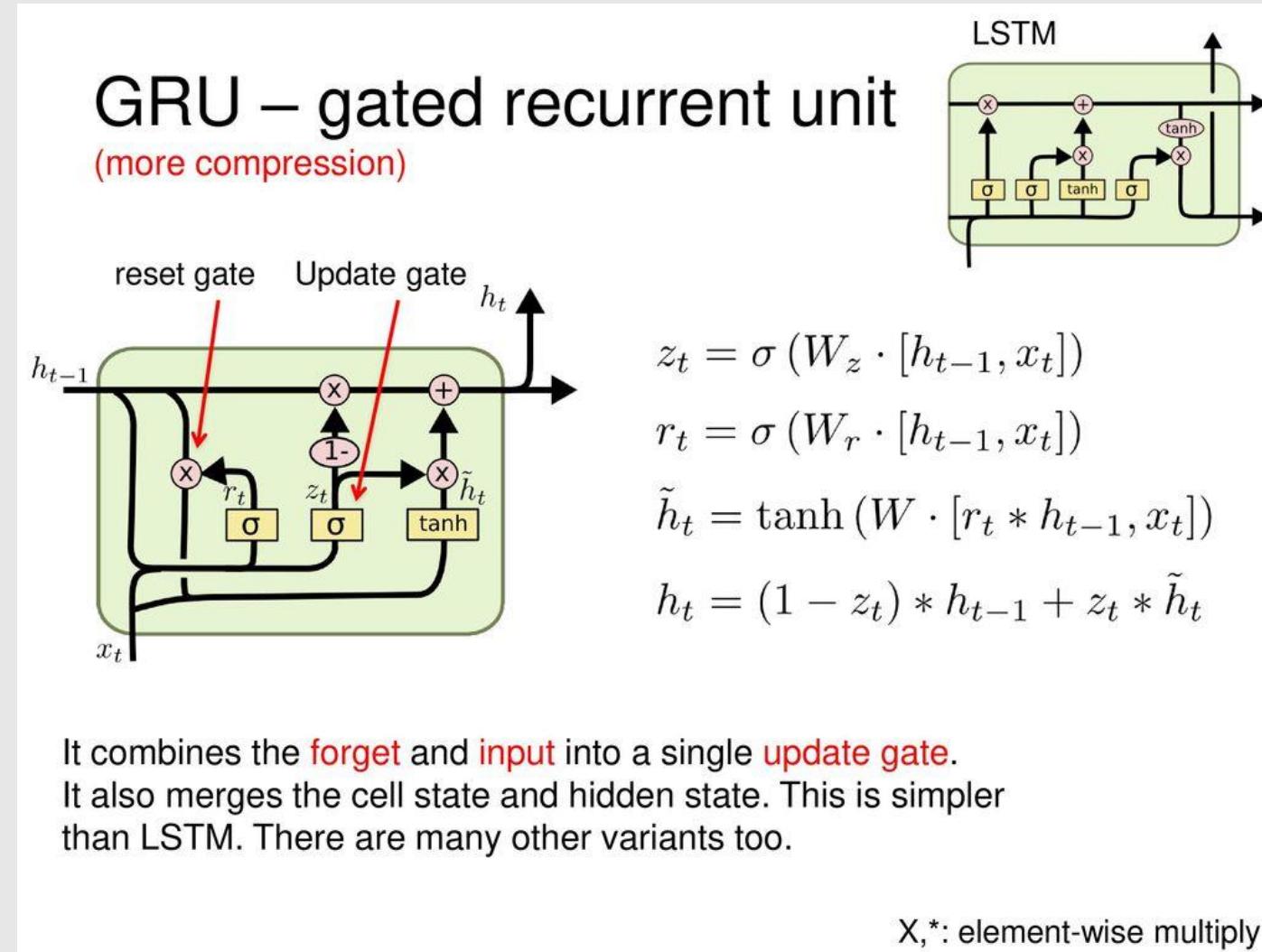
Deep neural networks have shown great success in various applications such as objection recognition (see, e.g., (Krizhevsky et al., 2012)) and speech recognition (see, e.g., (Dahl et al., 2012)). Furthermore, many recent works showed that neural networks can be successfully used in a number of tasks in natural language processing (NLP). These include, but are not limited to, language modeling (Bengio et al., 2003), paraphrase detection (Socher et al., 2011) and word embedding extraction (Mikolov et al., 2013). In the field of statistical machine translation (SMT), deep neural networks have begun to show promising results. (Schwenk, 2012) summarizes a successful usage of feedforward neural networks in the framework of phrase-based SMT system.

The proposed RNN Encoder–Decoder with a novel hidden unit is empirically evaluated on the task of translating from English to French. We train the model to learn the translation probability of an English phrase to corresponding French phrase. The model is then used as a part of a standard phrase-based SMT system by scoring each phrase pair in the phrase table. The empirical evaluation reveals that this approach of scoring phrase pairs with an RNN Encoder–Decoder improves the translation performance.

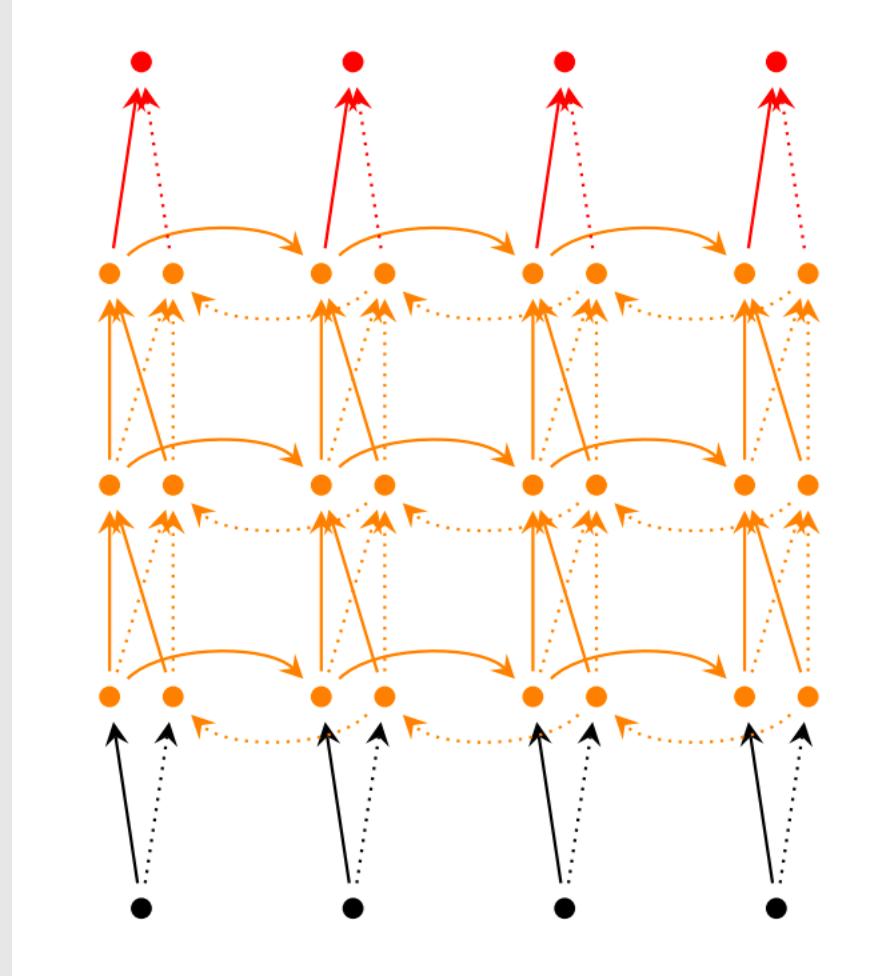
We qualitatively analyze the trained RNN Encoder–Decoder by comparing its phrase scores with those given by the existing translation model. The qualitative analysis shows that the RNN Encoder–Decoder is better at capturing the linguistic regularities in the phrase table, indirectly explaining the quantitative improvements in the overall translation performance. The further analysis of the model reveals that the RNN Encoder–Decoder learns a continuous space representation of a phrase that preserves both the semantic and syntactic structure of the phrase.

- 대한민국 조경현 교수의 논문
- LSTM의 장점은 유지하면서 계산복잡성을 낮춘 구조
- Update, Reset Gate로 구성

Gated Recurrent Unit (GRU)



Deep Recurrent Neural Network



02 II. Recurrent Neural Network

Reference

The screenshot shows a blog post titled "Recurrent Neural Network (RNN) Tutorial - Part 1". The post is by WildML and discusses RNNs. It includes a summary of what RNNs are and how they work, followed by a link to the full tutorial.

Recurrent Neural Network (RNN) Tutorial - Part 1

WildML이라는 블로그에 RNN에 관련된 좋은 튜토리얼(영어)이 있어서 번역해 보았습니다. 중간중간에 애매한 용어들은 그냥 영어로 남겨놓았는데, 번역이 이상한 부분을 발견하셨거나 질문이 있으시면 댓글로 알려주주세요!

RNN은 다양한 자연어처리(NLP) 문제에 대해 뛰어난 성능을 보이고 있는 인기있는 모델이다. 하지만 최근의 인기에 비해 실제로 RNN이 어떻게 동작하는지, 어떻게 구현해야 하는지에 대해 쉽게 설명해놓은 자료는 상당히 부족한 편이다. 따라서 이 튜토리얼에서는 아래 내용을 하나하나 자세히 다루고자 한다.

<https://aikorea.org/blog/rnn-tutorial-1/>



안녕하세요. 송호연입니다. 요즘.. 딥러닝에 꽤 빠져있어서..
최근 RNN 공부할겸 아래 블로그 글을 한글로 번역하였습니다.
원 저작자, Google Brain의 Chris Olah의 허락을 받고 번역하였습니다.
<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
<https://brunch.co.kr/@chris-song/9>

RNN과 LSTM을 이해해보자!

09 Mar 2017 | RNN

이번 포스팅에서는 Recurrent Neural Networks(RNN)과 RNN의 일종인 Long Short-Term Memory models(LSTM)에 대해 알아보도록 하겠습니다. 우선 두 알고리즘의 개념을 간략히 언급한 뒤 forward, backward compute pass를 천천히 들어보도록 할게요.

이번 포스팅은 기본적으로 미국 스탠퍼드대학의 CS231n 강좌를 참고하여 forward, backward pass 관련 설명과 그림은 제가 직접 만들었음을 밝힙니다. GRU(Gated Recurrent Unit)가 궁금하신 분은 [이곳](#)을 참고하시면 좋을 것 같습니다. 자, 그럼 시작하겠습니다!

<https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/>

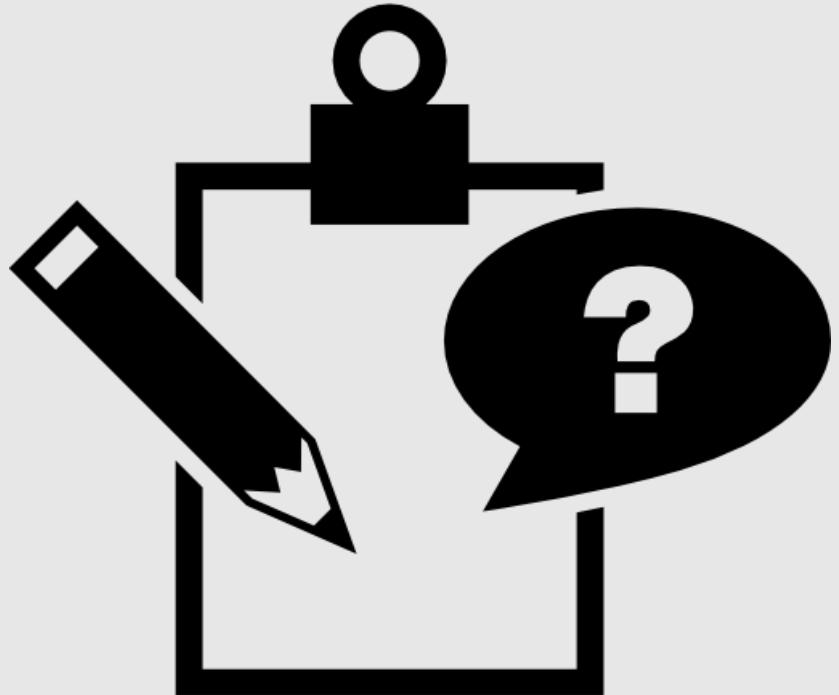
Recurrent Neural Networks(RNNs) 소개 – 기본 Architecture와 Backpropagation Through Time(BPTT) 알고리즘

2017년 5월 27일 by Solaris

이번 시간에는 Recurrent Neural Networks(RNNs)에 대해 알아보자. Recurrent Neural Networks(RNNs)은 Neural Networks의 구조 중 하나로, 특히 앞뒤순서가 존재하는 시계열 데이터에 대해 강력한 성능을 보여준다.

Recurrent Neural Networks(RNNs)의 architecture는 기본적인 Neural Networks 구조에 이전 시간($t - 1$)의 Hidden Layer의 output을 다음 시간(t)에 Hidden Layer로 다시 집어넣는 경로가 추가된 형태이다. 이 구조는 “recurrent-반복되는-”라는 단어에서 알 수 있듯이, 현재 시간 t 의 결과가 다음 시간 $t + 1$ 에 영향을 미치고, 이는 다시 다음 시간 $t + 2$ 에 영향을 미치는 과정이 끊임 없이 반복되는 neural network 형태이다.

<http://solarisailab.com/archives/1451>

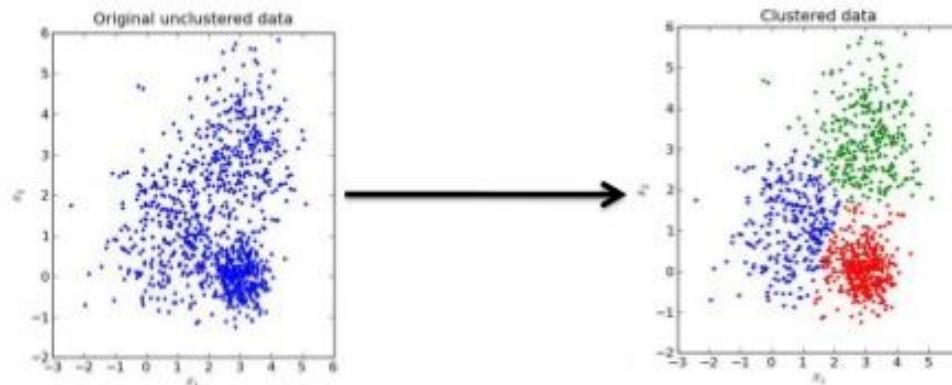


Autoencoder

Unsupervised Learning

비지도학습이란?

- 입력데이터만 주어졌을 때, 모델 스스로 데이터안에서 어떠한 관계를 찾아내는 것
- 정답데이터가 주어지지 않는다는 점에서 지도학습과의 차이점을 보임



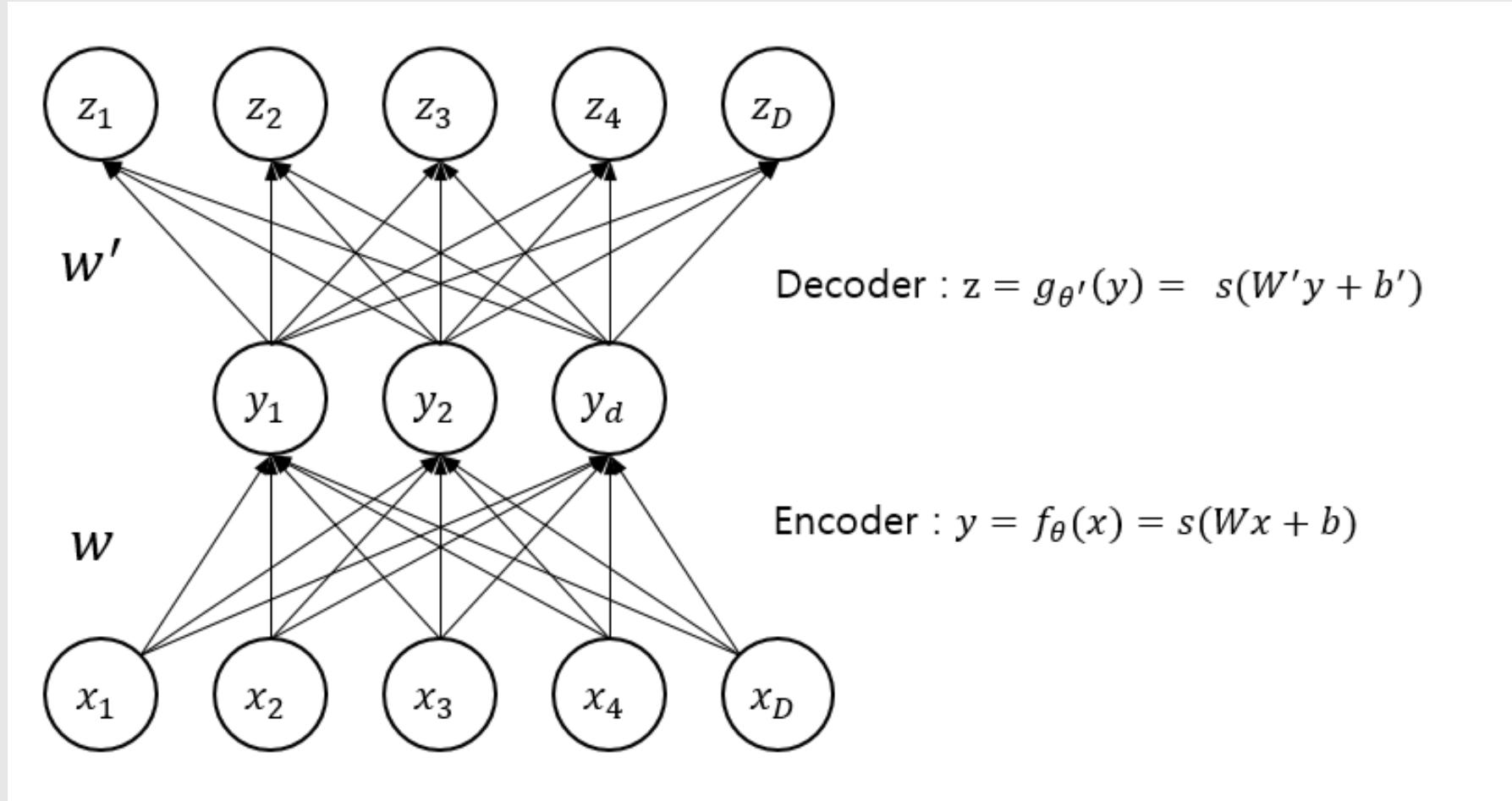
03 III. Autoencoder

Pattern



[⟨https://www.perrygargano.com/products/animal-chess-set⟩](https://www.perrygargano.com/products/animal-chess-set)

Autoencoder Architecture



Autoencoder Architecture

기존의 차원축소 문제들은 Unsupervised Learning이었는데, 이를 Encoder와 Decoder로 이루어져있는 NN을 이용하여 Supervised Learning 문제로 해결하였다. 학습시킬 Data를 Encoder에 넣고 이를 다시 Decoder에 넣어서 나온 학습 결과가 기존 Data와 비슷하게 되도록, 이 Error를 줄이도록 학습을 시켜서 기존의 데이터를 잘 보존하는 가운데 Latent Feature들을 얻는다는 데에 의미가 있다.

→ 데이터를 압축해서 다시 복원을 해주는데 트레이닝 데이터에 대해서는 이 복원을 잘 해주는, 즉 최소한의 성능 보존해주는 그러한 차원 축소를 하고 싶다고 해석할 수 있다.

03 III. Autoencoder

Autoencoder Training

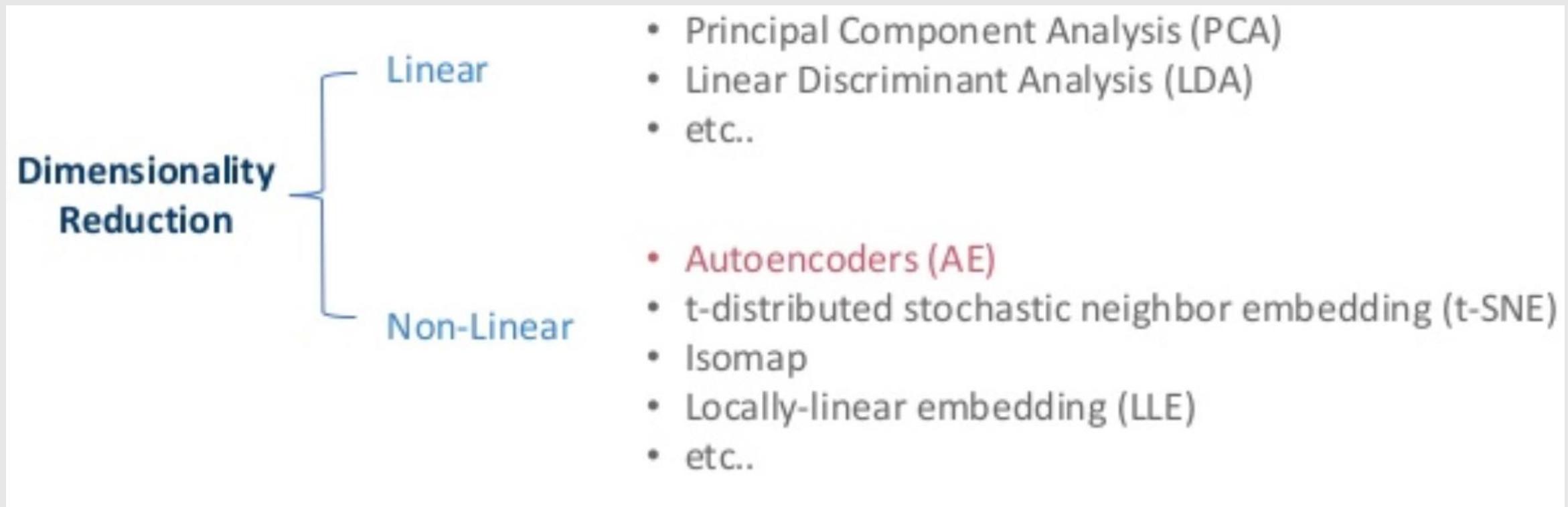
step	내용	부가
1	input vector $x_u = (r_1, r_2, \dots, r_i, \dots, r_m)$ 은 user u 의 m 개의 영화에 대한 rating이다.	rating은 1~5 사이의 값이고, rating 없으면 0이다.
2	input vector(user 단위) 가 하나 network 로 들어간다.	
3	input vector x_u 는 vector z 로 인코딩 된다. x 에서 z 로 갈 때 mapping function에 의해 차원이 축소된다.	mapping function : $z = f(Wx+b)$ (f : sigmoid, tanh , etc)
4	z 가 output vector y 로 decoding 된다. y 는 x vector 와 같은 차원을 가진다.	y 가 x 의 복사본이 되게 하는 것이 학습의 목적이다.
5	reconstruction error $d(x,y)$ 를 계산한다. 이 error function 을 최소화시킨다.	error function : $d(x,y)=\ x-y\ $
6	back-propagation 을 이용해, error의 값이 역전파되고, W, b 값들이 tuning 된다.	learning rate에 따라 학습 정도가 달라진다.
7	step 1~6 을 반복하면서 파라메터들을 업데이트한다.	만약 vector 하나씩 넣으면서 update 시키면 Reinforcement Learning 이고, 여러 batch 쪽 한꺼번에 넣으면서 학습시키면 Batch Learning
8	전체 데이터셋을 한번 다 학습시켰다면, epoch 단위로 몇번 더 학습한다.	

오토인코더의 학습은 다음과 같이 이루어진다.

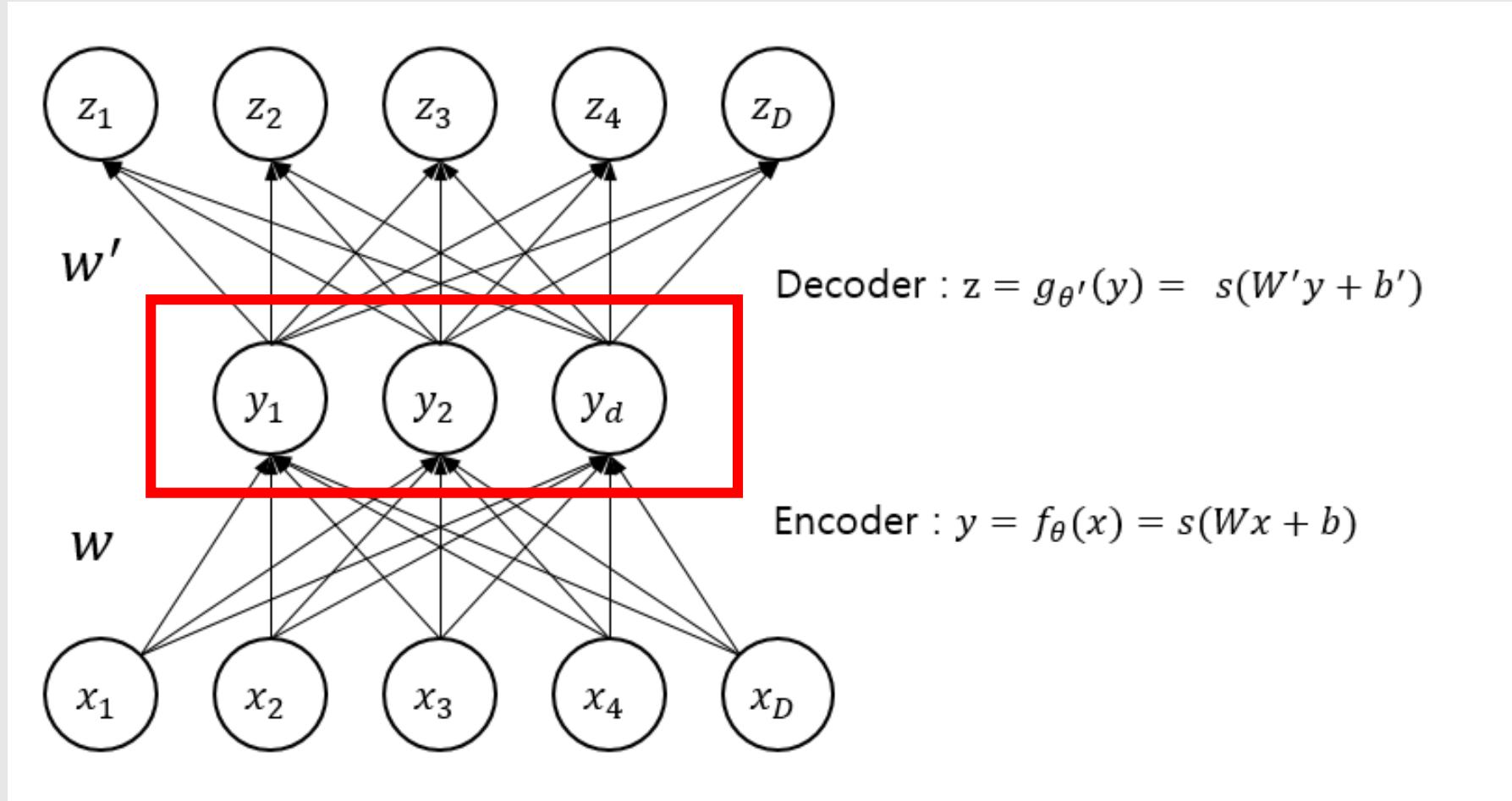
1. 인풋과 히든 레이어의 가중치를 계산해 시그모이드 함수를 통과시킨다.
2. 1의 결과물과 출력 레이어의 가중치를 계산해 시그모이드 함수를 통과시킨다.
3. 2의 값을 이용해 MSE(Mean Squared Error)를 계산한다.
4. 3의 결과로 나온 loss 값을 SGD로 최적화시키고
5. 오류역전파를 사용하여 가중치를 갱신한다.

[⟨https://artoria.us/22⟩](https://artoria.us/22)

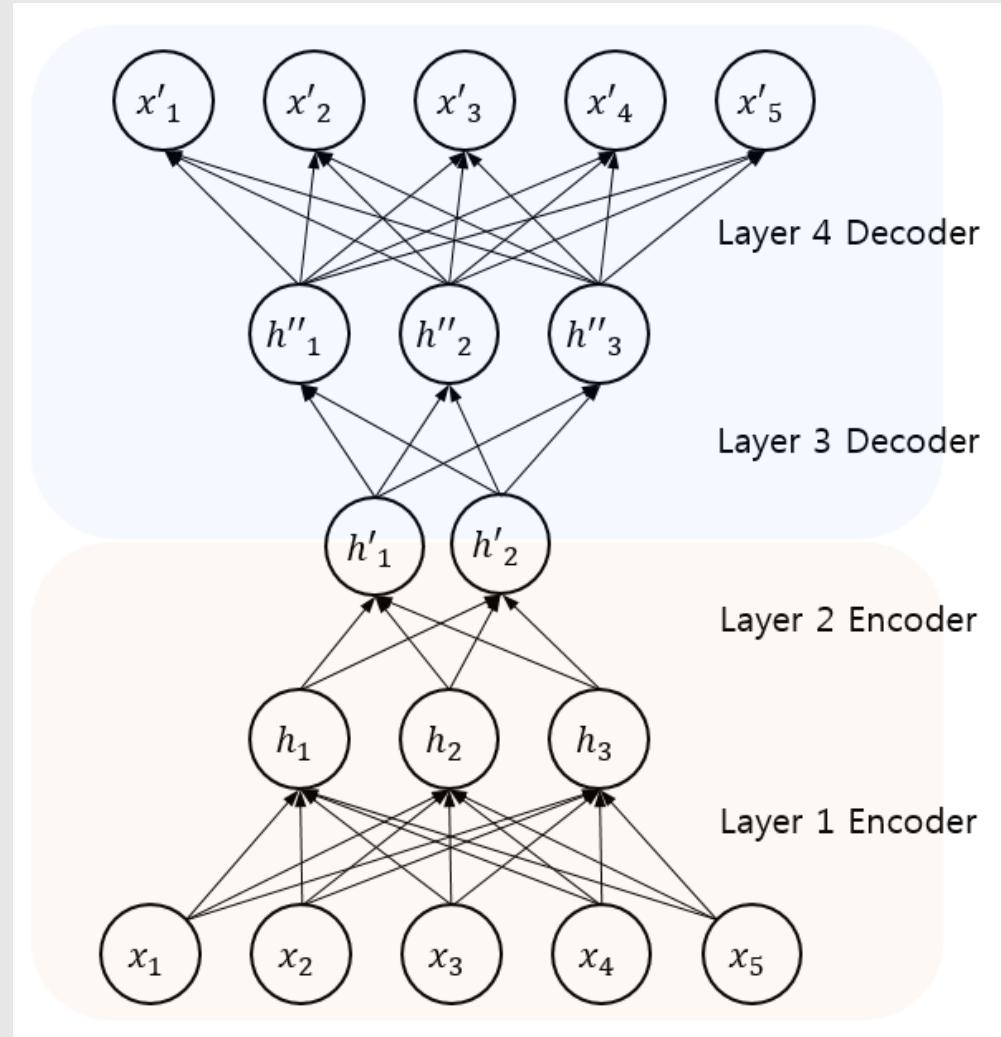
Basic Autoencoder



Basic Autoencoder



Stacked Autoencoder

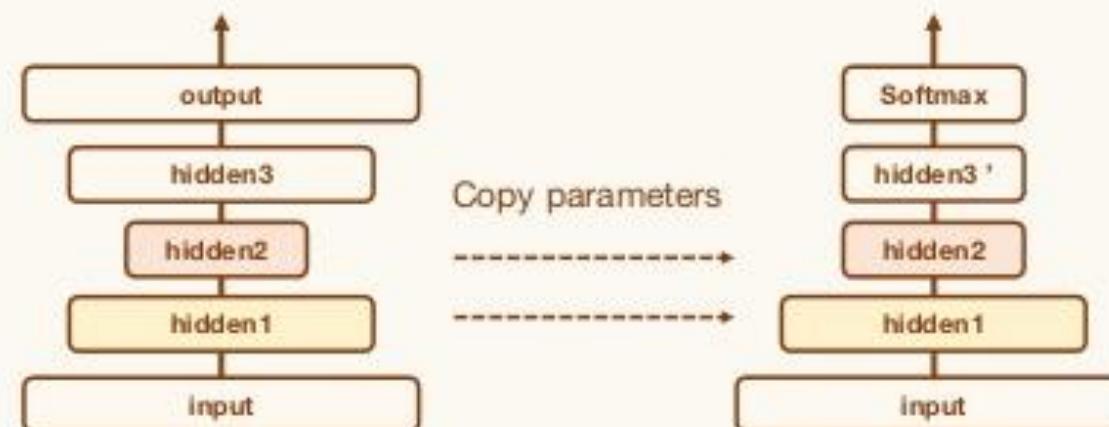


- 다른 NN과 마찬가지로 층을 깊게 쌓을 수 있음
- 다만 너무 깊으면 Overfitting되기 쉬움

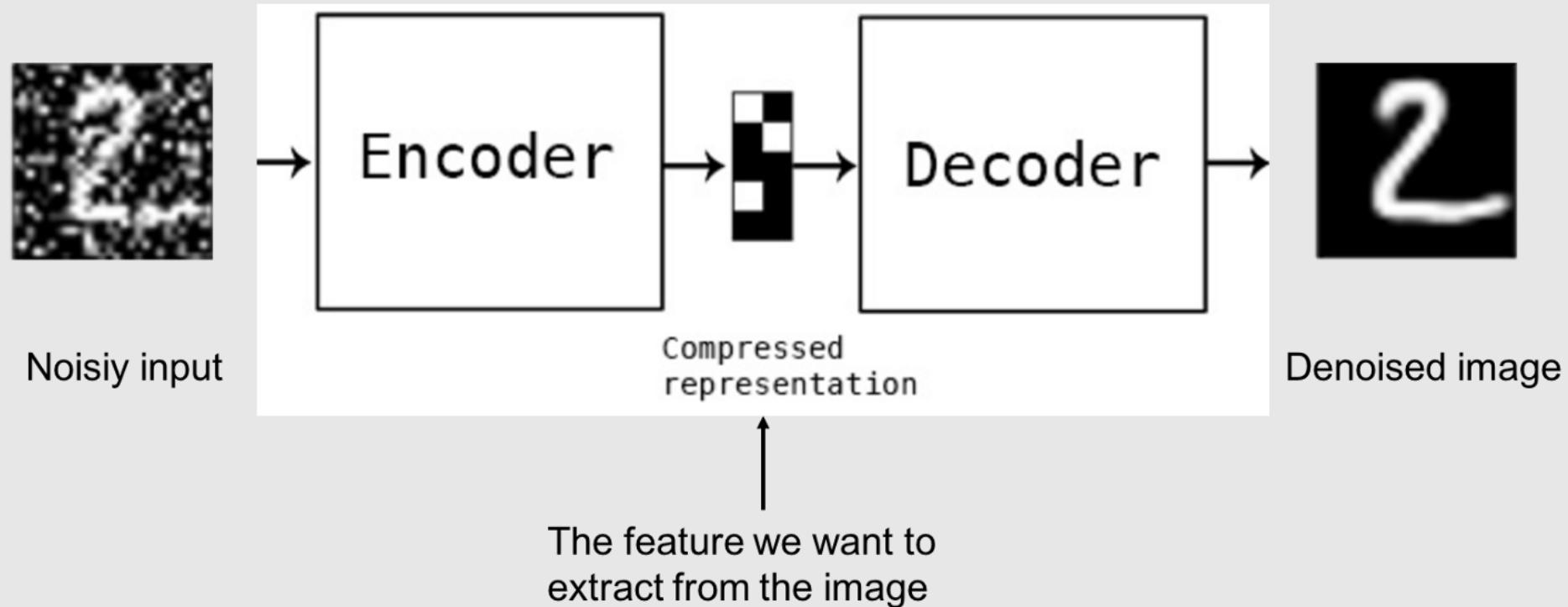
Stacked Autoencoder

Pretraining Using (Stacked) Autoencoders

- Train a Stacked Autoencoder with using all the data, then reuse the lower layers to create a network for your task.



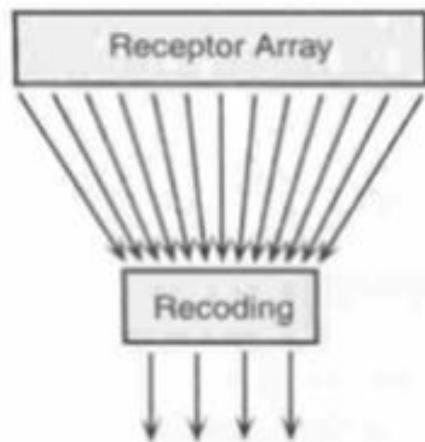
Denoising Autoencoder



<https://blog.sicara.com/keras-tutorial-content-based-image-retrieval-convolutional-denoising-autoencoder-dc91450cc511>

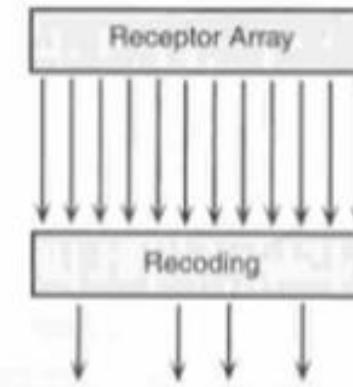
Sparse Coding

Compact coding



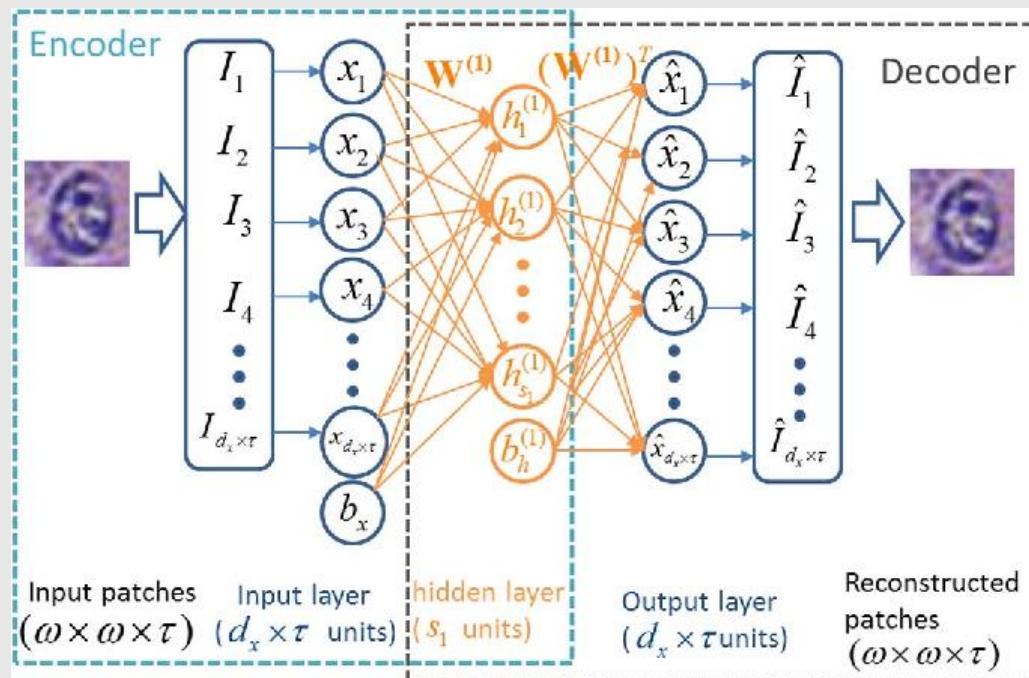
Represents data w/ the
Minimal number of units

Sparse coding



Represents data w/ the
Minimal number of active units

Sparse Autoencoder



위 그림에서 Hidden Layer에는 총 500개의 Unit이 존재하지만 기본적인 Autoencoder처럼 reconstruction error에 기반한 weight matrix W 를 구하는 것이 아니라 여기에 Sparsity 조건을 강제함으로써 Hidden Unit에 존재하는 뉴런의 활성화를 제한하는 Autoencoder

Variational Autoencoder

하고 싶은 것은 ?

- 이미지와 같은 고차원 데이터 X 의 저차원 표현 z 를 구할 수 있다면
- z 를 조정하여 training set에서 주어지지 않은 새로운 이미지 생성이 가능
- 카메라 각도, 조명 위치, 표정등의 조정이 가능



Other examples

- random faces
- MNIST
- Speech

These are not part of
the training set !

<https://www.youtube.com/watch?v=XNZIN7Jh3Sg>

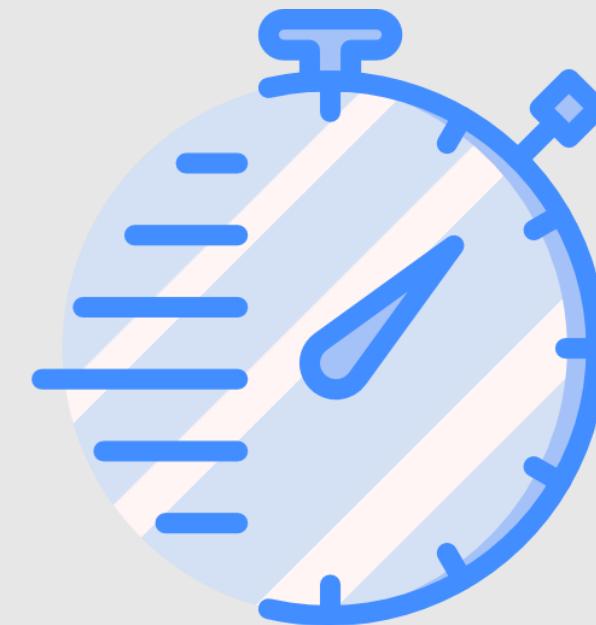
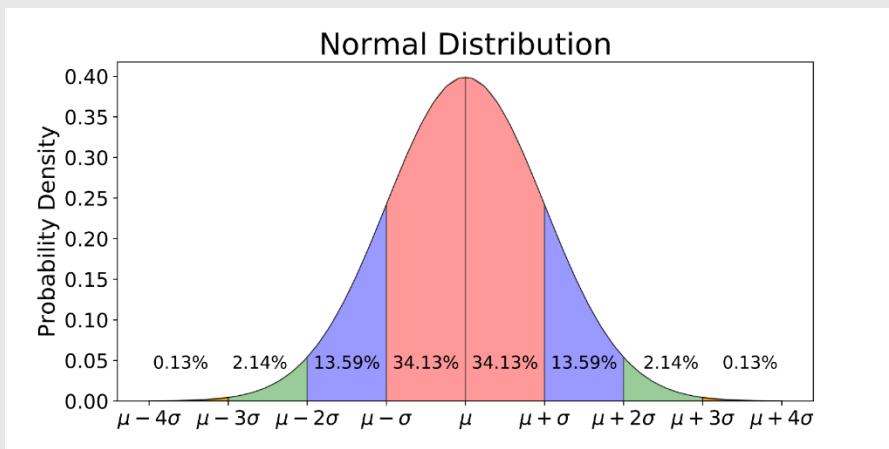
Basic Generative Model's Problem

데이터 구조의 가정과 모델의 근사가 필요

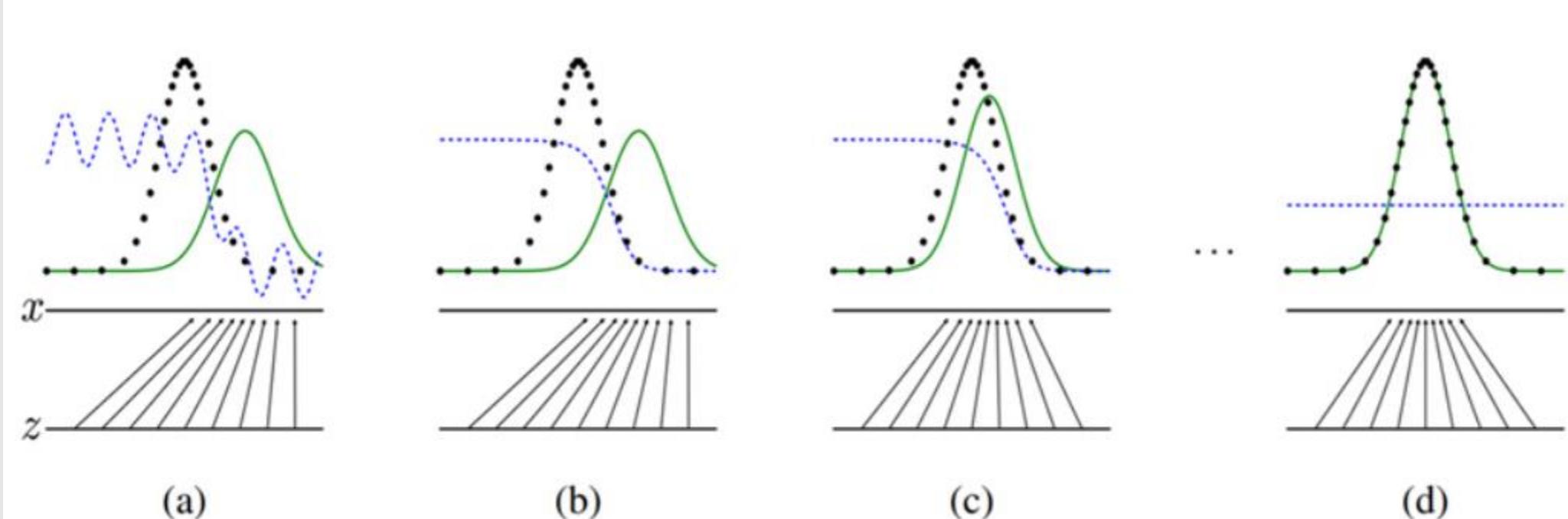
- 어떠한 분포의 설정이 필요
- 설정한 분포에 모델이 대응해야 함

시간이 많이 소요됨

- MCMC와 같은 복수 샘플링이 필요



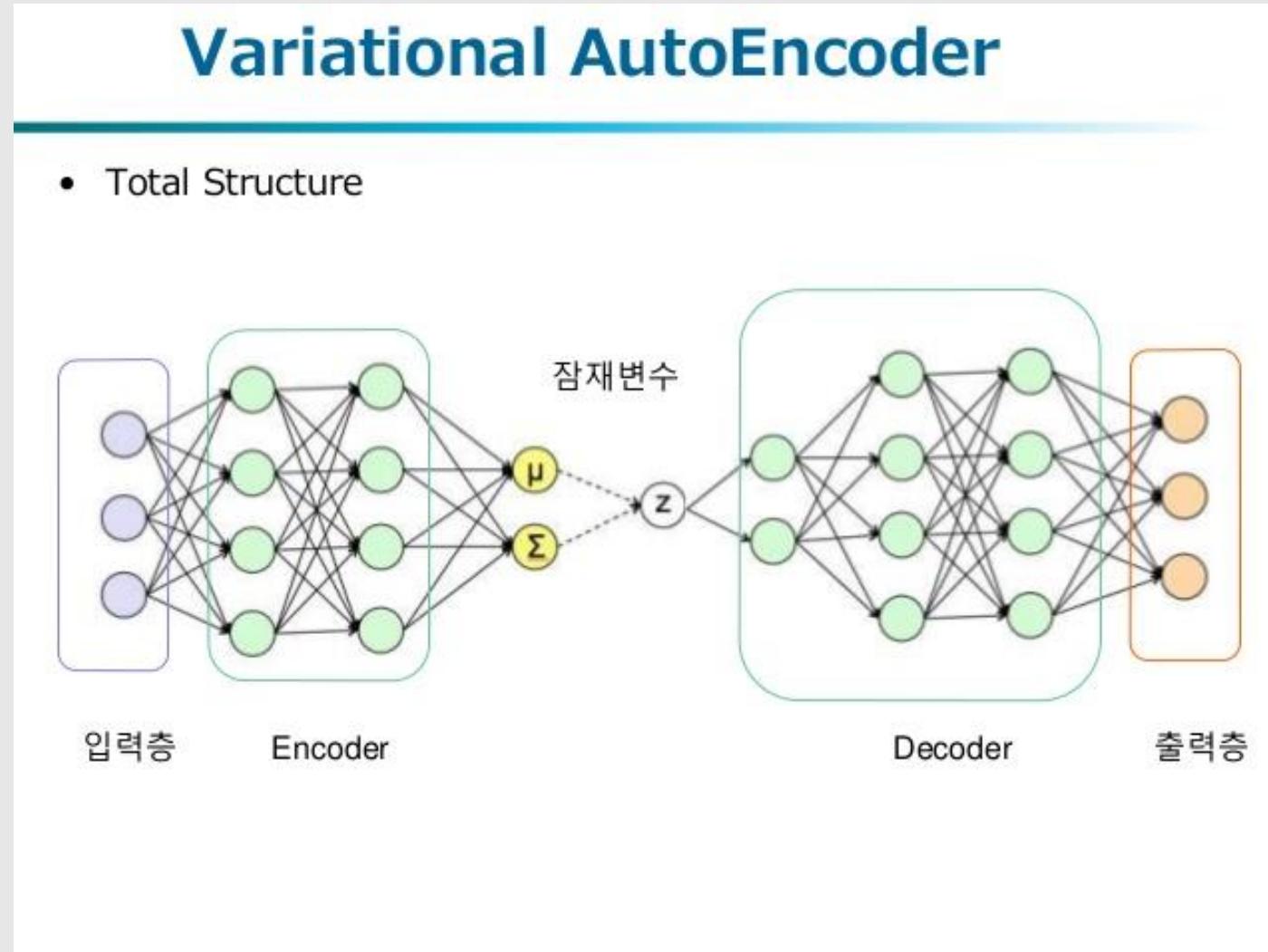
Generative Adversarial Networks; GAN



* 검은 점선: 원 데이터의 확률분포, 녹색 점선: GAN이 만들어 내는 확률분포, 파란 점선: 분류자의 확률분포

GAN에서 학습을 통해 확률분포를 맞추어 나가는 과정
 - Ian.J.Goodfellow의 'Generative Adversarial Networks' 논문 인용

Variational Autoencoder



03 III. Autoencoder

Variational Autoencoder & GAN



03 III. Autoencoder Reference

검색어를 입력하세요.

Deep Learning 이론과 실습 ... / 1. Introduction Auto- ...

1. Introduction Auto-Encoder

- 발표 내용 요약
- Autoencoder 의 종류와 특성
- Autoencoder 의 투토리얼 소개
- Autoencoder 의 튜토리얼 실습

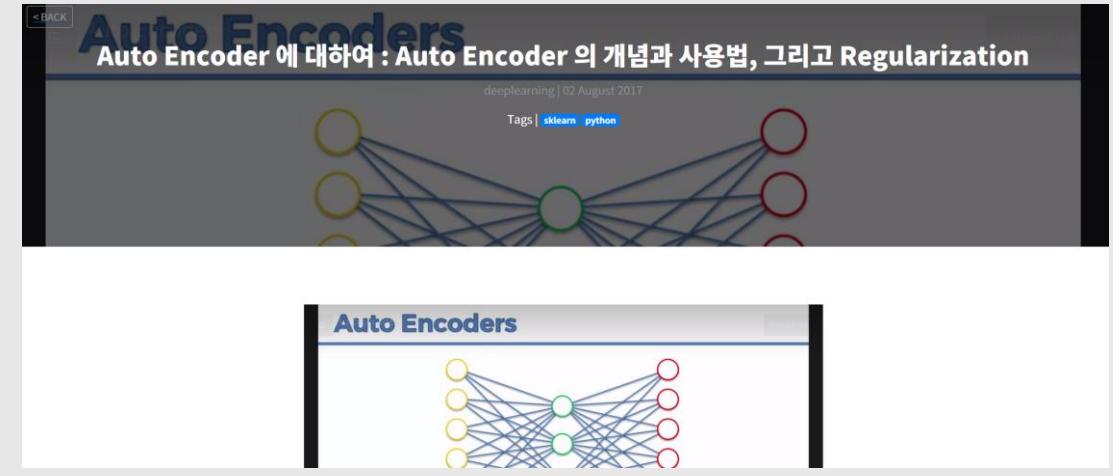
1.2. Simple Tutorial on Auto-Encoder
Tutorial 1.2.1. Sparse Autoencoder
Tutorial 1.2.2. Sparse Autoencoder (My Image)
Tutorial 1.2.3. Stacked AE (Iris data)
Tutorial 1.2.4. Denoising AE (MNIST and MyImage)
Tutorial 1.2.4.x dAE MyImage Training
Tutorial 1.2.5 Anomaly Detection by Deep AE
Tutorial 1.2.6 이상지 짐지 (업데이트 예정)
1.3. Conclusion (Auto-Encoder 소감)

1. Introduction Auto-Encoder

1. Auto-Encoder 종류와 특성

1.1 Auto-Encoder (Basic form)

<https://wikidocs.net/3413>



[머신러닝] 18. 머신러닝 학습 방법(part 13) – AutoEncoder(5) | 쉽게 읽는 머신 러닝 / Academy
2017. 2. 24. 19:15
<https://laonple.blog.me/220943887634>

쉽게 읽는 머신러닝

Machine Learning 18. 머신러닝 학습방법(PART13)

쉽게 읽는 머신 러닝 – 학습방법 (part 13) – AutoEncoder5

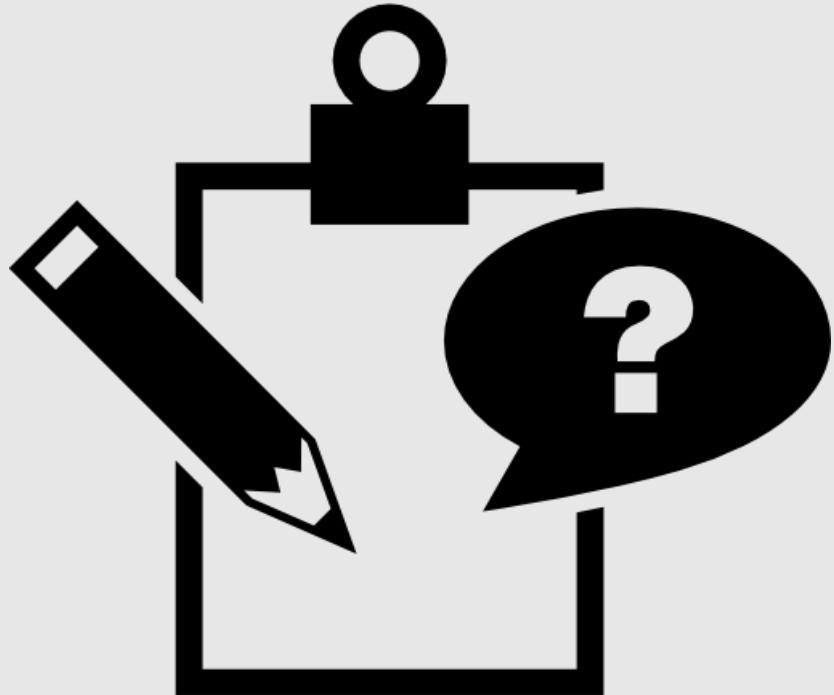
<https://laonple.blog.me/220943887634>

PRETRAINING | Autoencoder
Stacking Autoencoder

A diagram illustrating a stack of autoencoders. It shows an input layer of 784 nodes, followed by two hidden layers of 1000 nodes each, and an output layer of 500 nodes. The output of the final layer is labeled 'Target'. A person is pointing at the diagram. The video player interface shows a play button, volume control, and a progress bar from 1:28:15 to 1:41:36.

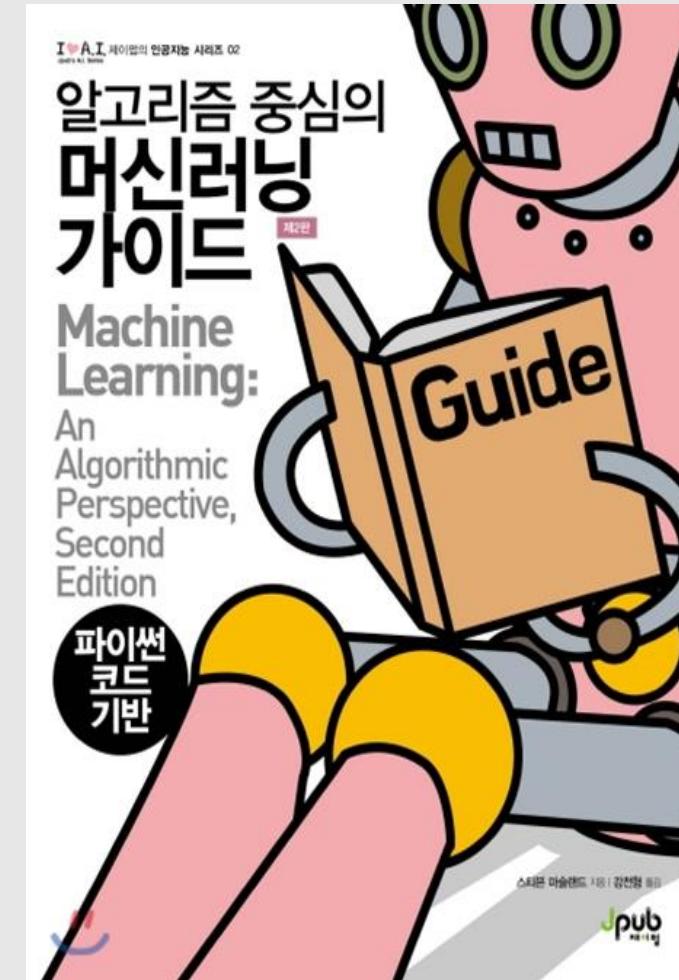
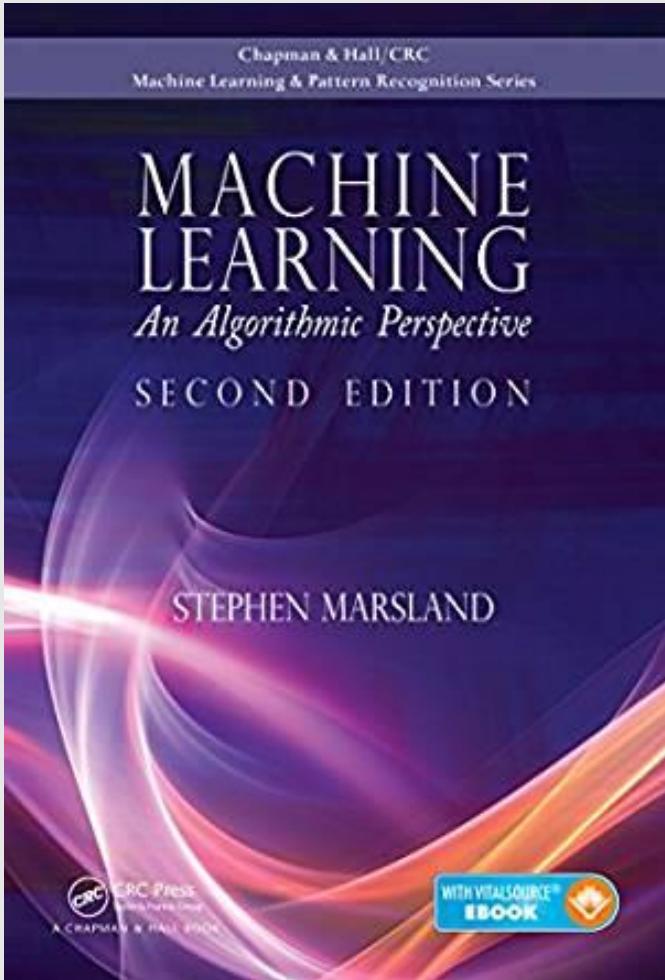
오토인코더의 모든 것 - 1/3
조회수 8,112회

https://www.youtube.com/watch?v=o_peo6U7IRM&t=5295s



Reinforcement Learning

Machine Learning: An Algorithmic Perspective



Reinforcement Learning Introduction

Reinforcement learning fills the gap between supervised learning, where the algorithm is trained on the correct answers given in the target data, and unsupervised learning, where the algorithm can only exploit similarities in the data to cluster it. The middle ground is where information is provided about whether or not the answer is correct, but not how to improve it. The reinforcement learner has to try out different strategies and see which work best. Search is a fundamental part of any reinforcement learner: the algorithm searches over the state space of possible inputs and outputs in order to try to maximize a reward.

(강화학습은 목표 데이터에서 주어진 정답에 대한 알고리즘을 학습하는 지도학습과 알고리즘을 클러스터링하기 위한 데이터의 유사성만 활용할 수 있는 비지도학습간의 차이를 채웁니다. 대답이 정확한지 아닌지에 관한 정보가 있지만 개선방법은 정보가 없는 곳입니다. 강화학습은 여러 다른 전략을 시도하고 가장 잘 작동하는지 확인해야 합니다. 탐색은 모든 강화학습기의 기본적인 부분으로, 알고리즘은 보상을 극대화하기 위해 가능한 입력 및 출력의 상태 공간을 탐색합니다.)

Reinforcement Learning Introduction

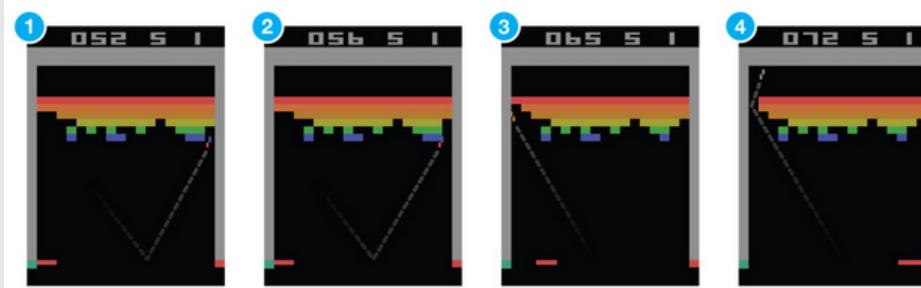
Reinforcement learning is usually described in terms of the interaction between some agent and its environment. The agent is the thing that is learning, and the environment is where it is learning, and what it is learning about. The environment has another task, which is to provide information about how good a strategy is, through some reward function.

(강화학습은 agent와 그 주변의 환경 사이의 상호 작용으로 보통 설명됩니다. Agent는 학습을 하는 자를 의미하며, 환경은 학습이 이루어지는 곳, 그리고 학습의 대상을 나타냅니다. 환경은 또 다른 전략이 얼마나 좋은지에 대항 정보를 보상 함수를 사용해서 알려주는 일을 합니다.)

The importance of reinforcement learning for psychological learning theory comes from the concept of trial-and-error learning, which has been around for a long time, and is also known as the Law of Effect. This is exactly what happens in reinforcement learning.

(심리학 학습 이론을 위한 강화학습의 중요성은 오랜 기간 동안 시행되어온 시행 오 학습의 개념에서 유래되었으며, 또한 효과 법칙(Law of Effect)으로 알려져 있습니다. 이것은 정확히 강화학습에서 일어나는 일입니다.)

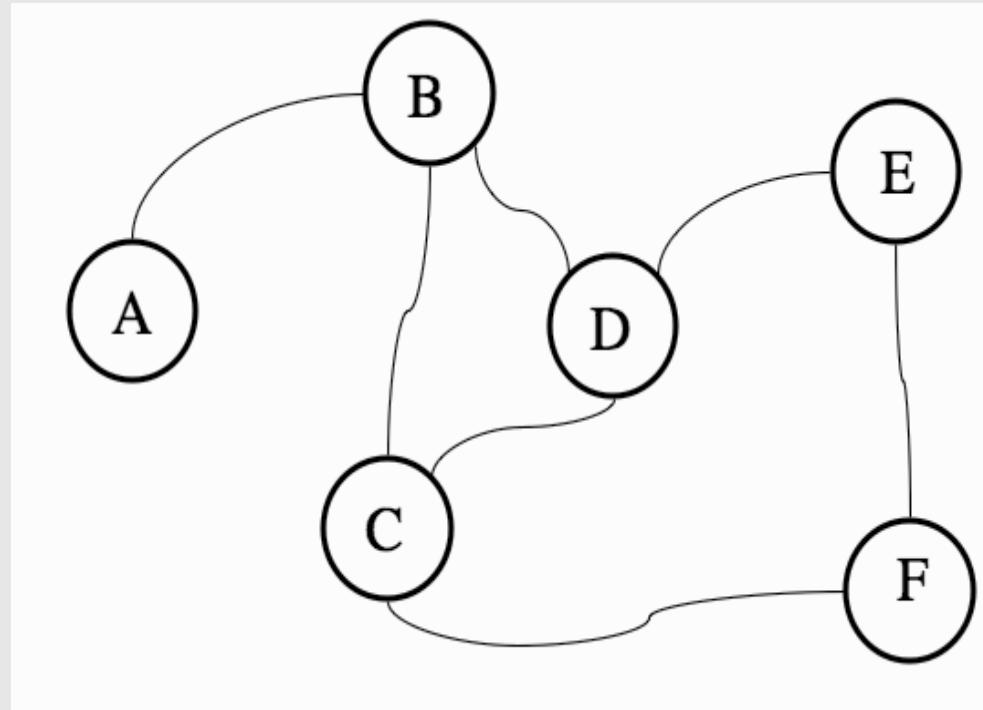
Reinforcement Learning Introduction



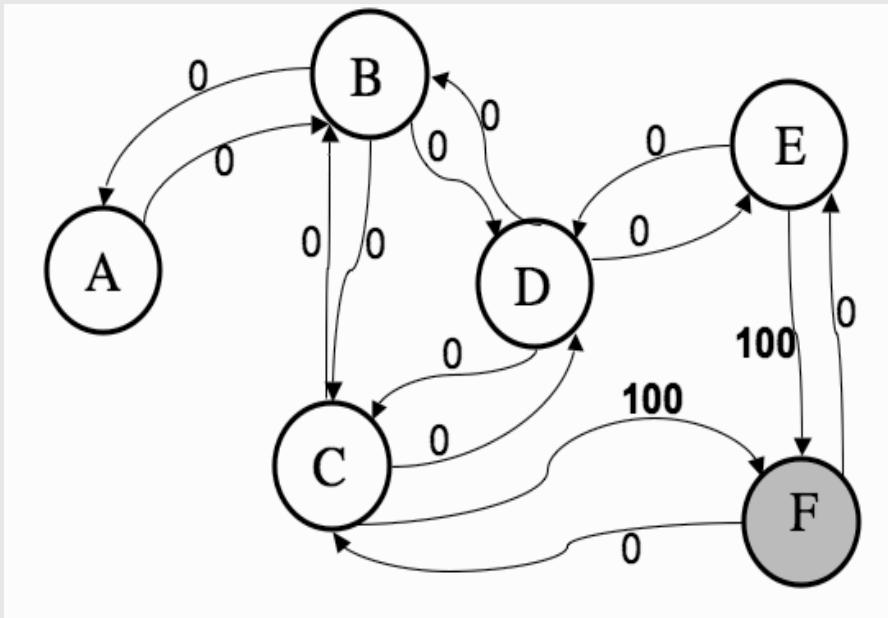
뉴럴네트워크에게 이 게임을 가르친다고 생각해봅시다. 네트워크에 입력하는 값들은 화면 이미지일 것이고, 결과는 세 가지 행동으로 이루어집니다. 왼쪽, 오른쪽, 발사(처음 공을 띠울때). 이 문제는 분류문제로 생각 할 수 있습니다. 결정해야하는 각각의 게임의 화면에서, 당신은 왼쪽, 오른쪽, 혹은 공을 발사시키는 행동을 할 것입니다. 쉬워 보이죠? 물론 트레이닝 예시들이 필요하고, 이 예시들은 정말 많이 필요합니다. 물론 여러분들이 나가서 전문 게이머들이 게임하는 장면들을 녹화할 수도 있겠지만, 우리가 진짜 배우려고 하는 것들은 이런 것들이 아닙니다. 각각의 화면에서 어디로 움직여야 할지 누군가에게 수백만번 말하라고 할 필요는 없습니다. 잘 하고 있는지 가끔 피드백을 받고 그것으로 우리가 모든 것들을 알아낼 수 있습니다.

이것이 바로 강화학습이 풀고자 하는 문제입니다. 강화학습은 지도학습과 비지도학습 사이 어딘가에 위치하고 있습니다. 지도학습은 각각의 트레이닝 예시를 위한 목표 라벨이 있는 반면 비지도 학습은 라벨이 아예 없는데, 강화학습은 뜨문 뜨문히, 시간이 지연되는 리워드 라벨들을 가지고 있습니다. 이러한 리워드만을 바탕으로 각각의 상황에서 에이전트들은 어떻게 행동해야할지 학습해야합니다.

Example: Getting Lost



Example: Getting Lost



Current State	Next State					
	A	B	C	D	E	F
A	-5	0	-	-	-	-
B	0	-5	0	0	-	-
C	-	0	-5	0	-	100
D	-	0	0	-5	0	-
E	-	-	-	0	-5	100
F	-	-	0	-	0	-

Example: Getting Lost

- S - state들의 set을 의미한다. 앞에서 예를 든 2족 보행 로봇의 경우 모든 가능한 관절의 상태와 환경 등이 state가 된다. 참고로 state와 다음에 기술한 action의 개수가 유한하다면 $|S| < \infty, |A| < \infty$, 주어진 MDP를 finite MDP라고 부른다.
- A - action들의 set을 의미한다. 2족 보행 로봇의 경우 어떻게 관절을 control할 것인가를 의미한다.
- $P_{sa} : (s_t, a_t) \rightarrow s_{a_t}$ - State의 transition probability로, 특정 state에서 특정 action을 취했을 때 다음 state는 어떤 state가 될지에 대한 확률 값이다.
- $R : S \times A \rightarrow \mathbb{R}$ - 주어진 state에 action을 수행했을 때 얻게 되는 reward를 function으로 표현한 것이다. Reinforcement learning의 목표는 시간이 T 만큼 흘렀을 때 최종적으로 얻게 되는 모든 reward들의 합을 (정확하게는 그 것의 expectation을) maximization하는 policy를 learning하는 것이다.
- $\gamma \in [0, 1)$ - 앞에서 설명한 reward의 discount factor로, 시간이 지날수록 reward의 가치를 떨어뜨리고, 처음 받은 reward의 가치를 더 키워주는 역할을 한다. 즉, time t 에서 얻은 reward를 r_t 라고 했을 때, 전체 reward R_{tot} 는 $\sum_{t=0}^T \gamma^t r_t$ 가 된다.

Example: Getting Lost

Now that the reward has been broken into two parts—an immediate part and a pay-off in the end—we need to think about the learning algorithm a bit more. The thing that is driving the learning is the total reward, which is the expected reward from now until the end of the task (when the learner reaches the terminal state or accepting state—the backpacker's in our example). At that point there is generally a large pay-off that signals the successful completion of the task. However, the same thing does not work for continual tasks, because there is no terminal state, so we want to predict the reward forever into the infinite future, which is clearly impossible.

(보상이 두 부분으로 나뉘어 졌으므로 이제는 학습 알고리즘에 대해 좀 더 생각해 볼 필요가 있습니다. 학습을 주도하고 있는 것은 총 보상입니다. 지금부터 과제의 끝 (학습자가 터미널 상태에 도달하거나 상태를 수용 할 때)까지 예상되는 보상입니다. 그 시점에서 일반적으로 작업의 성공적인 완료를 알리는 큰 보상이 있습니다. 그러나 터미널 상태가 없기 때문에 같은 일이 지속적인 작업에는 적용되지 않으므로 우리는 보상을 영원히 무한 미래로 예측하려고 합니다. 이는 분명히 불가능합니다.)

Example: Getting Lost → Discounting

The solution to this problem is known as discounting, and means that we take into account how certain we can be about things that happen in the future: there is lots of uncertainty in the learning anyway, so we should discount our predictions of rewards in the future according to how much chance there is that they are wrong. The rewards that we expect to get very soon are probably going to be more accurate predictions than those a long time in the future, because lots of other things might change. So we add an additional parameter $0 \leq \gamma \leq 1$, and then discount future rewards by multiplying them by γ^t , where t is the number of timestamps in the future this reward is from.

(이 문제에 대한 해결책은 할인(discounting)이라고 알려져 있습니다. 미래에 일어날 일에 대해 확신할 수 있는 방법을 고려해야 합니다. 어쨌든 학습에 많은 불확실성이 있으므로, 우리는 보상예측에 대한 할인을 해야 합니다. 그들이 얼마나 틀렸는지에 따라 미래가 달라질 수 있습니다. 우리가 빨리 얻을 것으로 예상되는 보상은 다른 것들이 많이 바뀔 수 있기 때문에 미래의 장래의 예측보다 더 정확한 예측이 될 것입니다. 따라서 우리는 $0 \leq \gamma \leq 1$ 인 추가 매개변수를 추가 한 다음 t 를 곱하여 미래의 보상을 할인합니다. 여기서 t 는 미래의 보상의 출처 타임스탬프의 수입니다.)

As γ is less than 1, so γ^2 is smaller again, and $\gamma^k \rightarrow 0$ as $k \rightarrow \infty$, so that we can ignore most of the future predictions. This means that our prediction of the total future reward is:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{k-1} r_k + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

(γ 가 1보다 작으면, γ^2 는 다시 작아지고, $k \rightarrow \infty$ 와 같이 $\gamma\gamma^k \rightarrow 0$ 이므로, 미래 예측의 대부분을 무시할 수 있다. 이것은 총 미래 보상에 대한 우리의 예측이 다음과 같다는 것을 의미합니다.)

Example: Getting Lost → Discounting

The point of the reward function is that it gives us a way to choose what to do next—our predictions of the reward let us exploit our current knowledge and try to maximize the reward we get. Alternatively, we can carry on exploring and trying out new actions in the hope that we find ways to get even larger rewards. The methods of exploration and exploitation that we carry out are the methods of action selection that we perform.

(보상함수의 핵심은 다음에 무엇을 할 것인지를 선택할 수 있는 방법을 제공한다는 것입니다. 보상에 대한 우리의 예측은 우리가 현재 가지고 있는 지식을 활용하고 보상을 최대화하려고 합니다. 또는 더 큰 보상을 얻을 수 있는 방법을 찾기 위해 새로운 행동을 시도하고 시험해 볼 수 있습니다. 우리가 수행하는 탐험과 활용의 방법은 우리가 수행하는 행동선택방법입니다.)

Example: Getting Lost → Action Selection

Greedy: Pick the action that has the highest value of $Q_{s,t}(a)$, so always choose to exploit your current knowledge.

(현재 지식을 탐험해서 가장 높은 $Q_{s,t}(a)$ 값을 갖는 행동을 선택합니다.)

Example: Getting Lost → Action Selection

ϵ -Greedy: This is similar to the greedy algorithm, but with some small probability ϵ we pick some other action at random. So nearly every time we take the greedy option, but occasionally we try out an alternative in the hope of finding a better action. This throws some exploration into the mix. ϵ -greedy selection finds better solutions over time than the pure greedy algorithm, since it can explore and find better solutions.

(이는 탐욕적인 알고리즘과 유사하지만 작은 확률 ϵ 를 사용하여 무작위로 다른 작업을 선택합니다. 거의 매번 우리는 탐욕스러운 선택을 하지만 때로는 더 나은 행동을 찾기 위해 대안을 시도합니다. 이것은 약간의 탐험 방법을 혼합함을 의미합니다. ϵ -greedy선택은 더 나은 솔루션을 찾을 수 있으므로 순수한 욕심 많은 알고리�보다 시간이 지남에 따라 더 나은 솔루션을 찾습니다.)

Example: Getting Lost → Action Selection

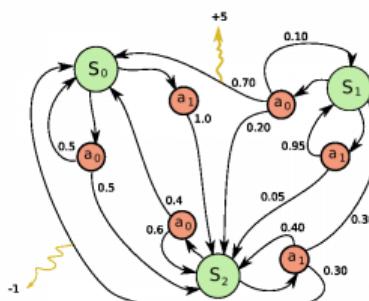
Soft-max: One refinement of ϵ -greedy is to think about which of the alternative actions to select when the exploration happens. The ϵ -greedy algorithm chooses the alternatives with uniform probability. Another possibility is to use the soft-max function to make the selection:

$$P(Q_{s,t}(a)) = \frac{\exp(Q_{s,t}(a)/\tau)}{\sum_b \exp(Q_{s,t}(a)/\tau)}$$

(ϵ -greedy의 개선 방안으로 탐험이 이뤄질 때 어떤 대안의 행동을 선택할지를 생각해 보면 선택을 균일분포 확률로 정합니다. 다른 대안으로는 소프트맥스 함수를 사용하는 것입니다.)

Markov Decision Processes

여러분이 에이전트라 생각하고, 어떤 환경(예를 들면, 벽돌게임)에 처해져있다고 생각해봅시다. 환경은 특정한 **상태**안에 있습니다(바의 위치, 공의 방향과 위치, 모든 벽돌의 존재 유무 등). 에이전트는 환경안에서 특정한 **행동**을 수행합니다(바를 왼쪽이나 오른쪽으로 옮기는 것). 때로는 이 행동들은 결과로 리워드를 얻습니다(점수가 올라가는 것). 행동들은 환경을 바꾸고, 에이전트가 또 다른 행동을 만들어내는 새로운 상태로 이끌어줍니다. 이런 행동들을 선택하는 방법을 **지침**이라 합니다. 보통 환경은 확률적으로 만들어지는데, 이는 다음 상태가 어느정도는 무작위적이라는 것입니다.(예를 들면, 공을 놓쳐버리고 난 후에는 새로운 공이 발사되죠. 이 공이 어느 방향으로는 모르죠.)



상태와 행동들의 모임으로 구성되어있고, 하나에서 또 다른 하나로 이전되는 규칙은 **마르코프 의사결정과정**을 만듭니다. 유한한 상태, 행동, 리워드들로 이 과정의 한 에피소드(예를 들면, 게임 한 턴)를 만듭니다.

Markov Decision Processes

The Markov Decision Process formalism is a powerful one that can deal with additional uncertainties. For example, it can be extended to deal with the case where the true states are not known, only an observation of the state can be made, which is probabilistically related to the state, and possibly the action. These are known as partially observable Markov Decision Processes (POMDPs), and they are related to the Hidden Markov Models that we will see in Section 16.3.

(Markov Decision Process 공식은 추가적인 불확실성을 처리 할 수 있는 강력한 식입니다. 예를 들어, 참인 상태를 알 수 없는 상황을 다루기 위해 확장될 수 있으며, 상태에 대한 확증이 이루어 질 수 있습니다. 이는 확률적으로 상태와 관련이 있으며 확률과 행동을 연관시킵니다. 이것들은 부분적으로 관찰 가능한 Markov Decision Process (POMDP)로 알려져 있으며, 16.3절에서 보게 될 은닉 마르코프 모델과 관련이 있습니다.)

Values

$$V(s) = E(r_t | s_t = s) = E \left\{ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s \right\}$$
$$Q(s, a) = E(r_t | s_t = s, a_t = a) = E \left\{ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} | s_t = s, a_t = a \right\}$$

상태가치 & 행동가치

Q-Learning

Q-러닝에서는 함수가 각 지점에서 계속 최적값을 찾으면서 상태에서 행동를 수행할 때 차감된 미래의 리워드(discounted future reward)를 나타내는 것이라고 정의합니다.

만약 아직까지도 확신이 서지 않는다면, 그 함수가 무엇을 내포하는지 생각해보세요. 여러분이 상태 안에 있고, 행동 와 둘 중에 어떤 행동을 선택해야 할지 숙고하고 있다고 생각해 보세요. 게임이 끝날 때에 가장 높은 점수를 얻을 수 있는 행동을 선택하고 싶다고 해 봅시다. 그리고 마법의 Q-함수를 가지고 있다면 답은 정말 간단해집니다. 최고 높은 Q-값과 함께 행동을 취하면 됩니다!

이는 각각의 상태에서 어떤 행동을 선택하는 규칙인 지침을 의미합니다.

Q-Learning

The Q-Learning Algorithm

- Initialisation
 - set $Q(s, a)$ to small random values for all s and a
- Repeat:
 - initialise s
 - repeat:
 - * select action a using ϵ -greedy or another policy
 - * take action a and receive reward r
 - * sample new state s'
 - * update $Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
 - * set $s \leftarrow s'$
 - For each step of the current episode
- Until there are no more episodes

Q-Learning

Note that we can do exactly the same thing for $V(s)$ values instead of $Q(s, a)$ values. There is one thing in this algorithm that is slightly odd, which is in the computation of $Q(s', a')$. We do not use the policy to find the value of a' , but instead choose the one that gives the highest value. This is known as an off-policy decision. Modifying the algorithm to work on-policy is very easy. It gets an interesting name based on the fact that it uses the set of values $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, which reads ‘sarsa’.

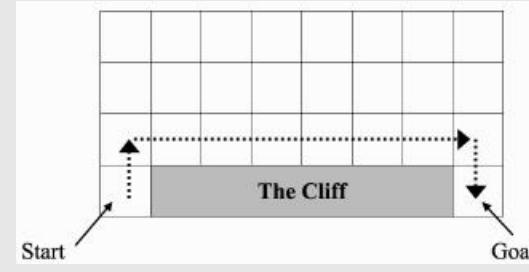
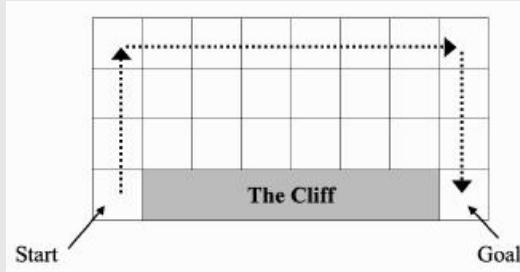
(또한 $Q(s, a)$ 대신 $V(s)$ 에 대해서 똑같은 적용을 할 수 있습니다. 약간의 다른 점은 $Q(s', a')$ 를 계산하는 방법인데, a' 값을 찾기 위해 정책을 사용하지 않고 가장 높은 값을 제공하는 값을 선택합니다. 이를 오프-정책(off-policy) 결정이라 합니다. 이를 온-정책(on-policy)로 수정하는 것은 매우 쉽습니다. ‘sarsa’라 불리는 $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ 값을 사용하면 됩니다.)

Q-Learning

The Sarsa Algorithm

- Initialisation
 - set $Q(s, a)$ to small random values for all s and a
- Repeat:
 - initialise s
 - choose action a using the current policy
 - repeat:
 - * take action a and receive reward r
 - * sample new state s'
 - * choose action a' using the current policy
 - * update $Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma Q(s', a') - Q(s, a))$
 - * $s \leftarrow s'$, $a \leftarrow a'$
 - for each step of the current episode
- Until there are no more episodes

The Difference between ‘Sarsa’ and ‘Q-Learning’



Left: The sarsa solution is far from optimal, but it is safe.

Right: The Q-learning solution is optimal, but occasionally the random search will tip it over the cliff.

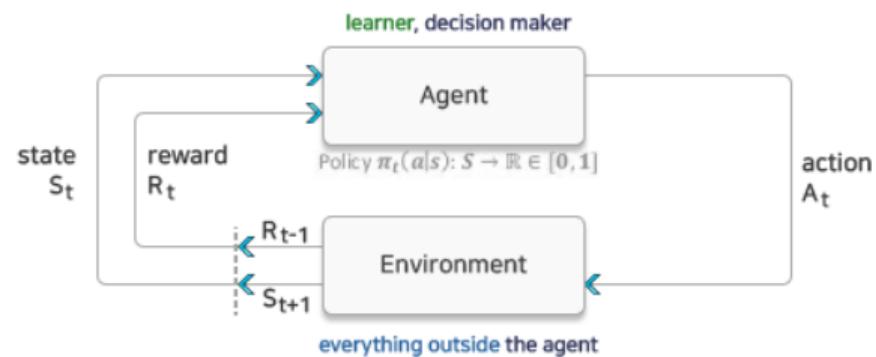
(왼쪽: sarsa솔루션은 최적에서 거리가 멀지만, 안전합니다.)

오른쪽: Q-learning은 최적에 가깝지만 임의의 탐색 탓에 절벽으로 떨어질 수 있습니다.)

Reinforcement Learning Summary

강화 학습(Reinforcement Learning)의 정의

강화 학습(Reinforcement Learning, 이하 RL)은 기계 학습이 다루는 문제들 중 하나로 어떤 환경 안에서 정의된 에이전트가 현재의 상태를 인식하여, 선택 가능한 행동들 중 보상을 최대화하는 행동 혹은 행동 순서를 선택하는 방법입니다.



- ✓ 에이전트(Agent) : 상태를 관찰, 행동을 선택, 목표지향
- ✓ 환경(Environment) : 에이전트를 제외한 나머지 (불리적으로 정의하기 힘듦)
- ✓ 상태(State) : 현재 상황을 나타내는 정보
- ✓ 행동(Action) : 현재 상황에서 에이전트가 하는 것
- ✓ 보상(Reward) : 행동의 좋고 나쁨을 알려주는 정보

04 IV. Reinforcement Learning

Reference

Ddanggle in MI · 5 minutes

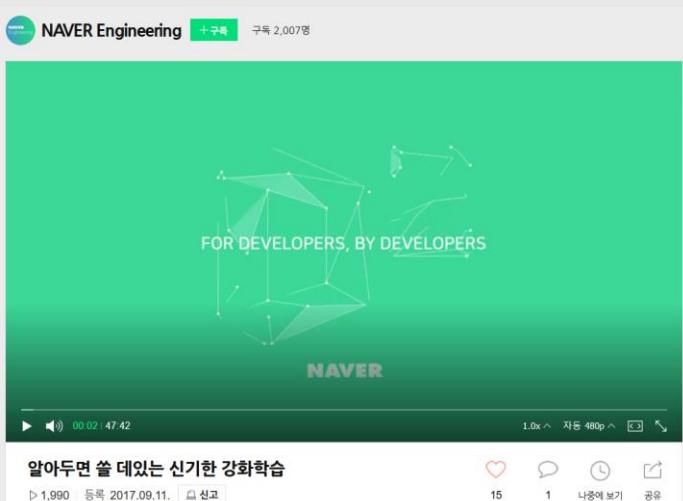
딥 강화학습 쉽게 이해하기

이 글은 머신러닝 블로그 nervanasys 저자 Tambet의 허락을 받아 원문 DEMYSTIFYING DEEP REINFORCEMENT LEARNING의 딥러닝 글을 번역한 것입니다. 원문도 꼭 읽어보셨으면 합니다.

저자는 DEMYSTIFYING DEEP REINFORCEMENT LEARNING 의 글이 더 정확하므로 여기를 참고하고 알려왔습니다. 저도 번역한 후에 이 곳을 다시참조해서 업데이트 하였습니다.

이 글은 딥 강화학습 시리즈의 첫 번째 글입니다. NEON 딥러닝 툴킷을 사용하여 실제 향상을 하는 것은 두 번째 글 Deep Reinforcement Learning with Neon로 이어집니다.

<http://ddanggle.github.io/demystifyingDL>

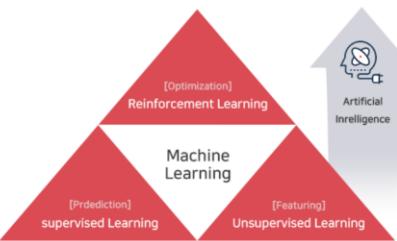


<https://tv.naver.com/v/2051482>

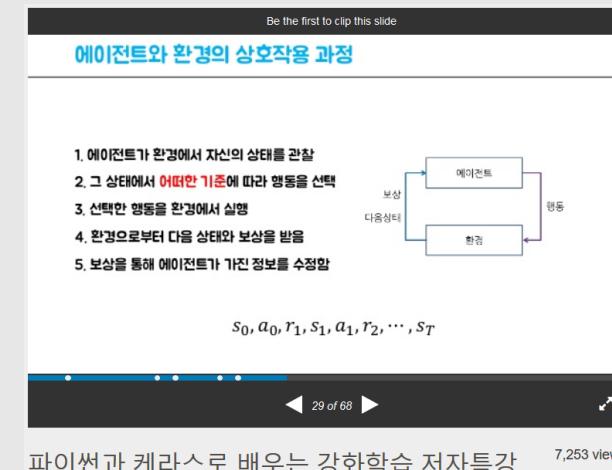
강화학습(Reinforcement Learning)

개념

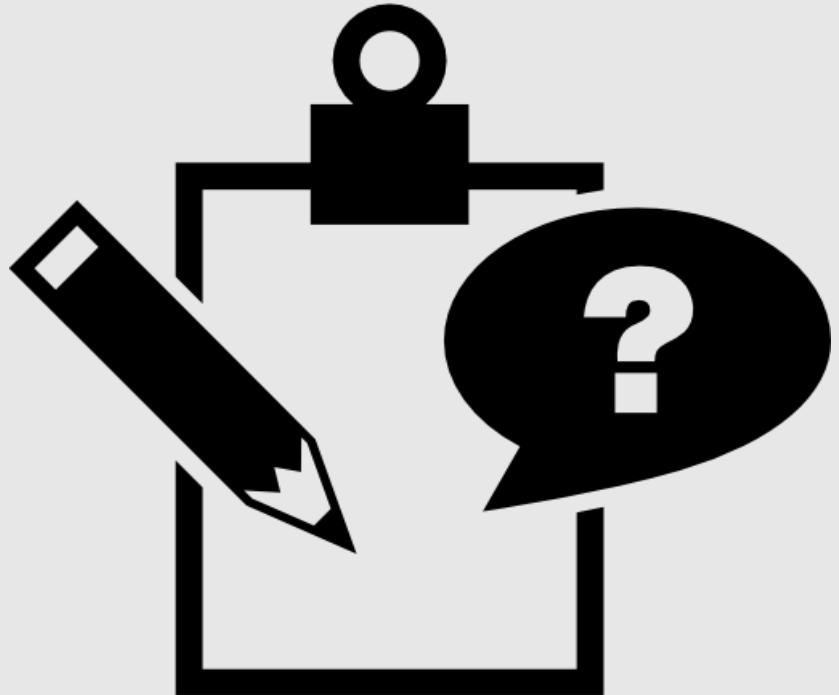
머신러닝 기법중 최적화 분석으로 가장 효과적인 강화학습(Reinforcement Learning)이 활용되고 있습니다. 미래의 가치를 극대화하기 위해 의사결정을 스스로 학습하는 방법으로서 머신러닝 또 다른 기법인 지도학습과 달리 Target은 성과(Reward)이고 예측값은 정책 혹은 수행 전략(Action)이 됩니다. 비즈니스 상황에 맞는 State, Reward, Environment, Action 등의 최적화된 설계가 구현에서 중요한 항목이 됩니다.



<http://www.agilesoda.com/kr/ai/aiTrends>

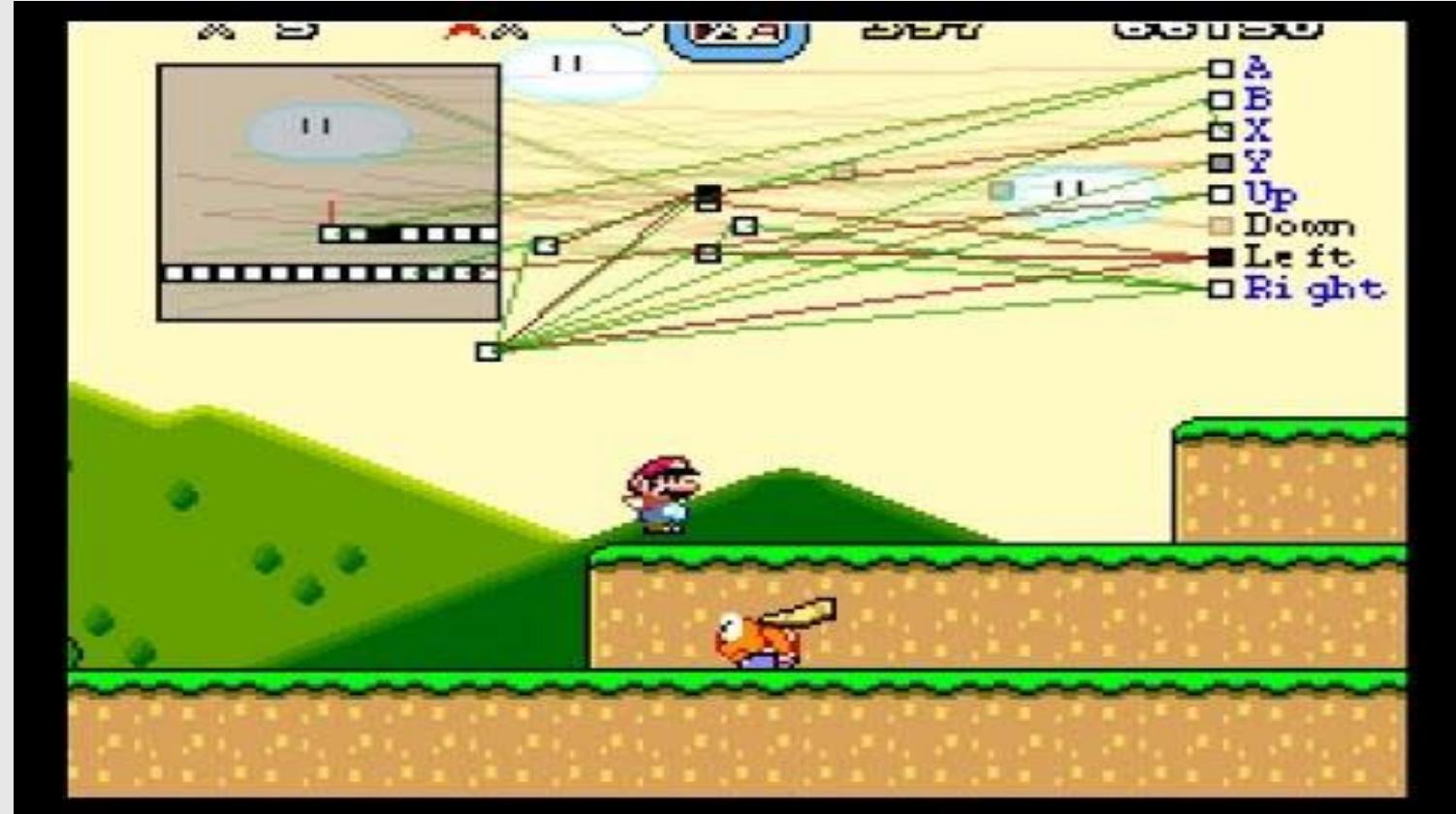


https://www.slideshare.net/WoongwonLee/ss-78783597?from_m_app=android



Before
Finish Class

Reinforcement Learning: Super Mario Bros.



<https://www.youtube.com/watch?v=qv6UVOQ0F44>

05 V. Before Finish Class

Reinforcement Learning: 쿠키런



<<https://www.youtube.com/watch?v=exXD6wJLJ6s>>

Schedule

12월 14일:

총 정리 + 시험 + Dacon 4차 회의 + 망년회

가락시장? 싱싱회천국?

+

시험 1등과 꽃등의 포상 및 벌칙?



Ideas worth spreading

- TED Talks

고생하셨습니다.