GLI Study
Deep Learning 101

From Pointer Network to GFlowNet for Combinatorial Optimization

Instructor: **Byungkook Oh**

Assistant Professor
Department of Computer Science and Engineering
Konkuk University

ver.251223

CONTENT

- **RNN Architectures**
- Combinatorial Optimization
- Pointer Network
- GFlowNet

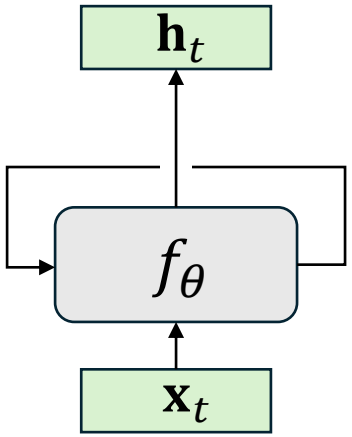
Recurrent Neural Networks (RNNs)

- Recurrent Neural Networks (RNNs)
  - Models **temporal** information
  - Hidden states as a function of inputs and previous time step information

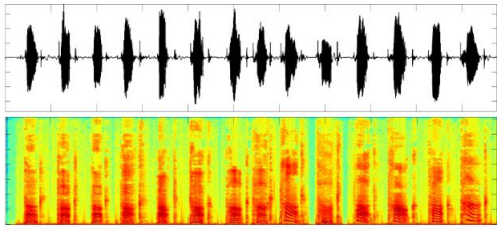
function parameterized by θ

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta)$$

new state | old state | input vector at time step t



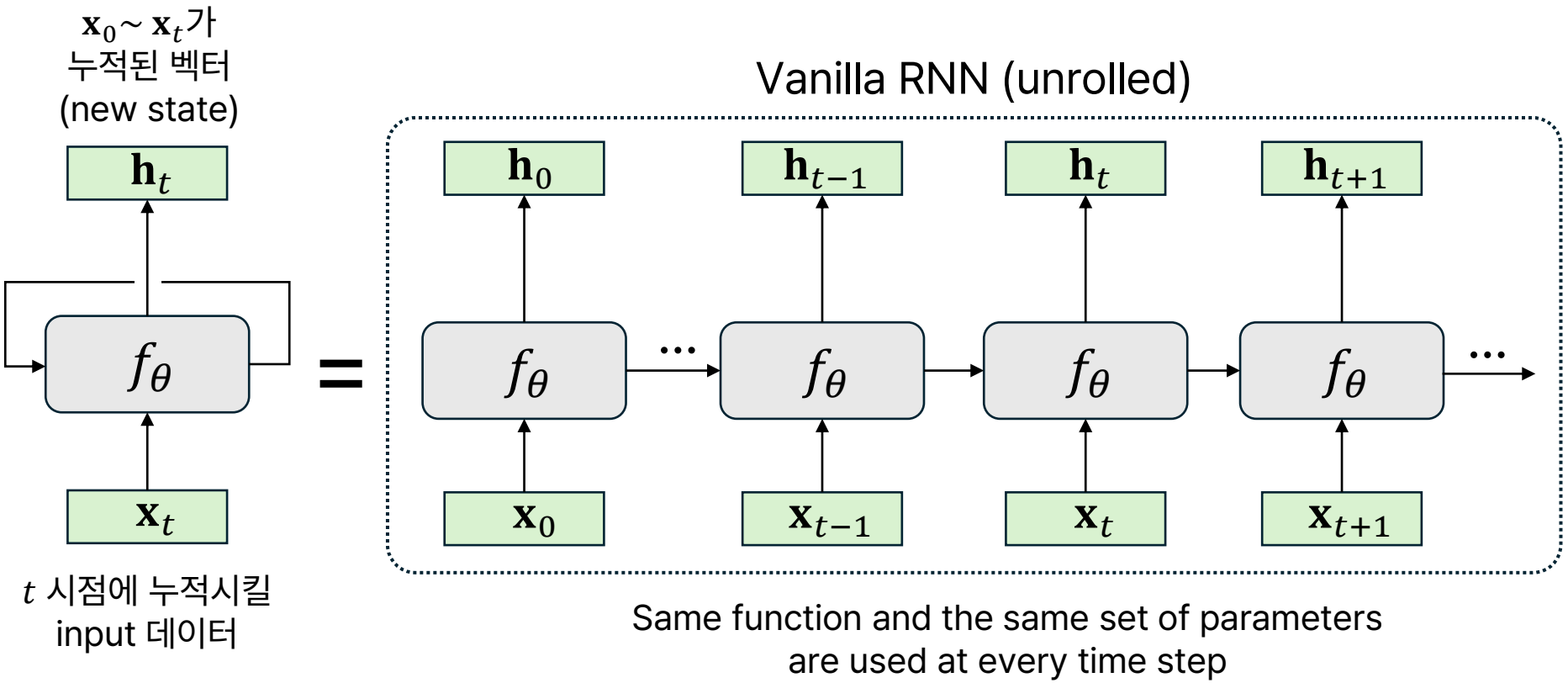
- Temporal information is important in many applications
  - Language
  - Speech
  - Video



Recurrent Neural Networks (RNNs)

- Process a sequence of vectors by applying recurrence formula **at every time step** :

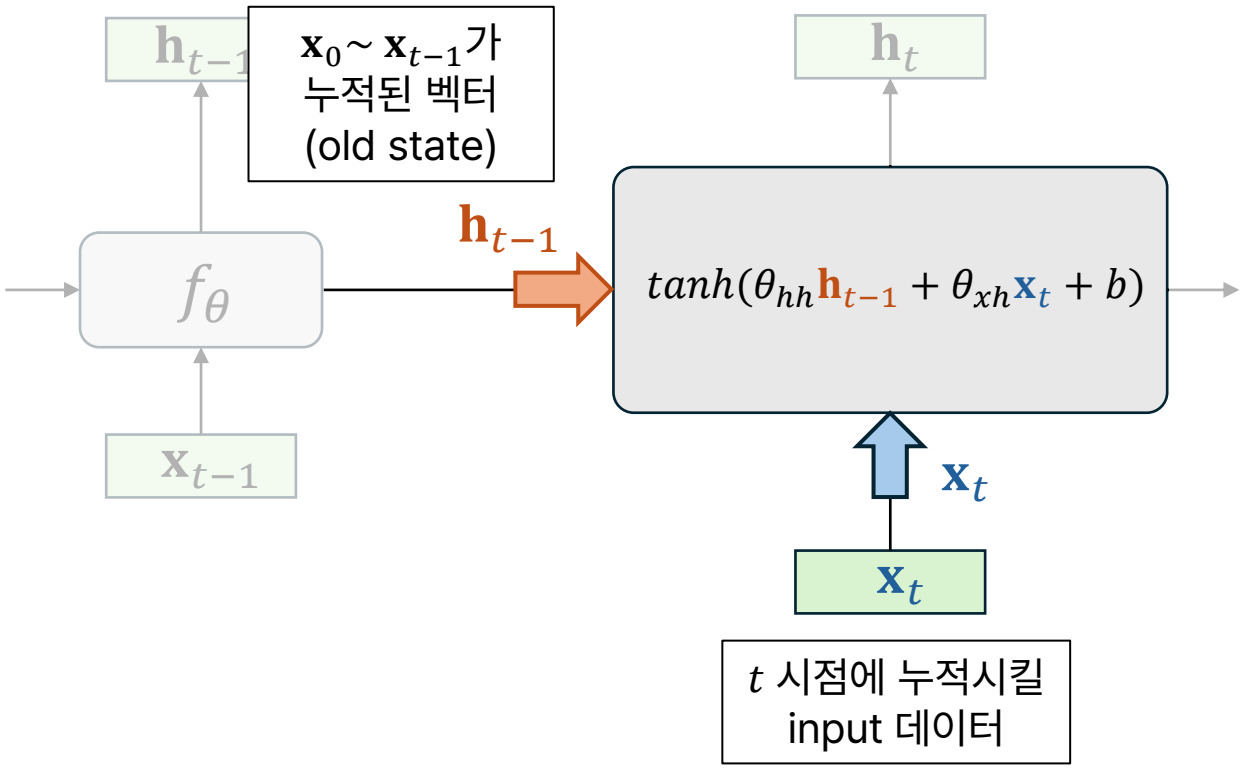
$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta)$$



Recurrent Neural Networks (RNNs)

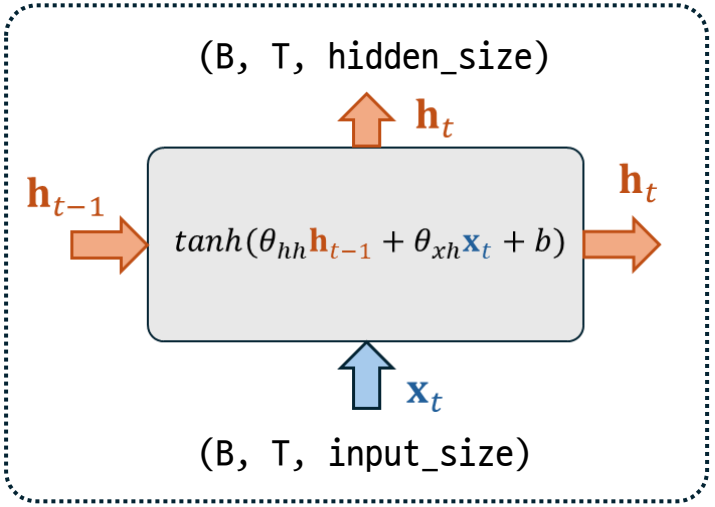
- Process a sequence of vectors by applying recurrence formula **at every time step** :

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta)$$



Recurrent Neural Networks (RNNs)

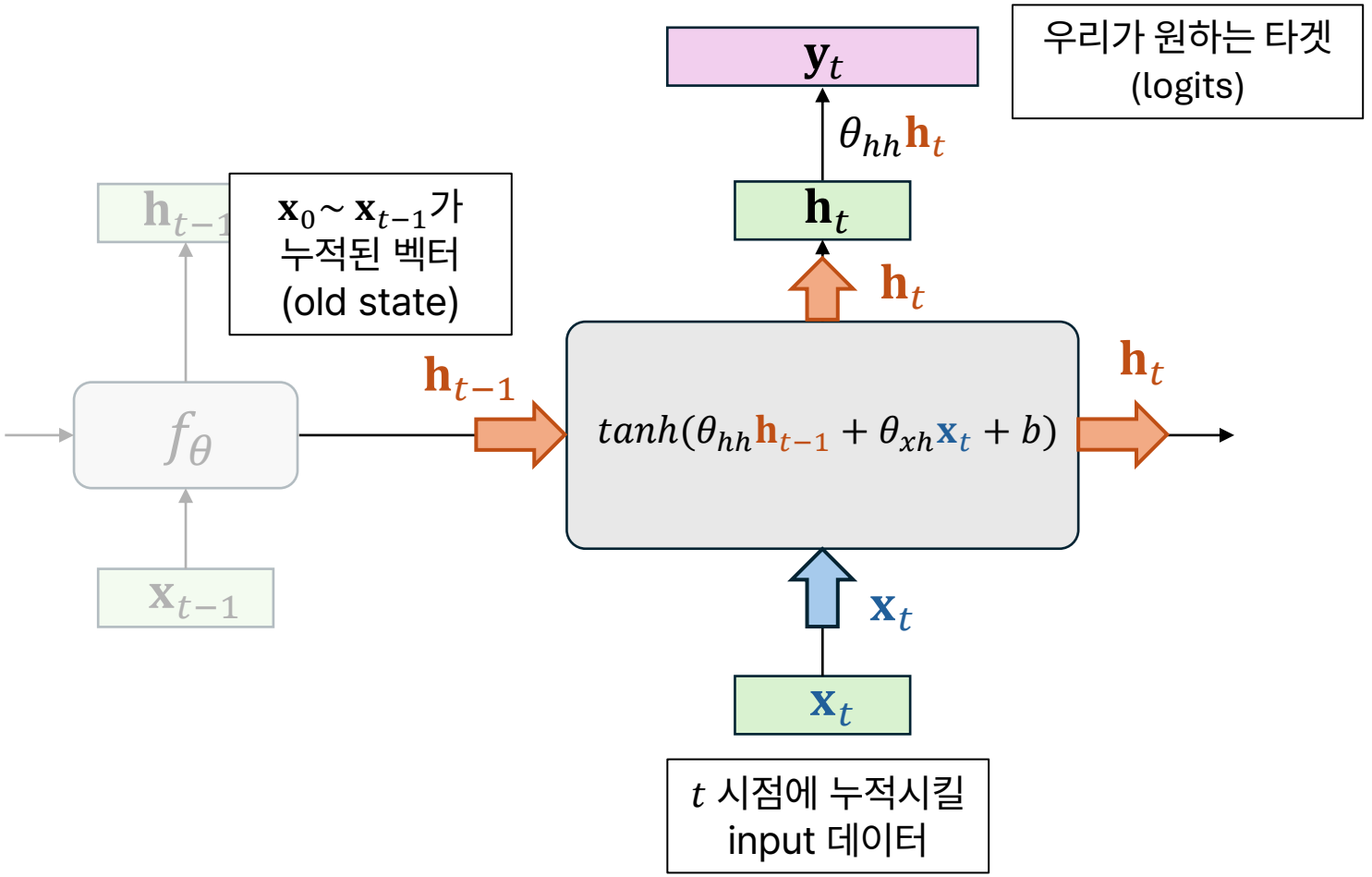
```
class VanillaRNN(nn.Module):
    def __init__(self, input_size: int, hidden_size: int):
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size

        self.W_x = nn.Linear(input_size, hidden_size, bias=True)
        self.W_h = nn.Linear(hidden_size, hidden_size, bias=True)

    def forward(self, x: torch.Tensor, h0: torch.Tensor | None = None):
        B, T, _ = x.shape
        if h0 is None:
            h = x.new_zeros(B, self.hidden_size)
        else:
            h = h0

        outputs = []
        for t in range(T):
            xt = x[:, t, :]
            h = torch.tanh(self.W_x(xt) + self.W_h(h))
            outputs.append(h.unsqueeze(1))

        y = torch.cat(outputs, dim=1)  # (B, T, H)
        return y, h
```



Recurrent Neural Networks (RNNs)

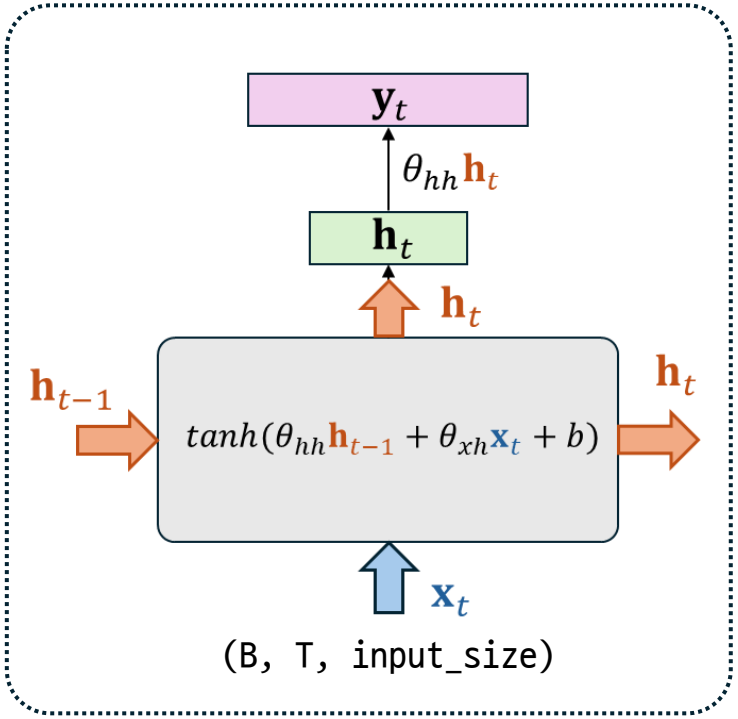
- Process a sequence of vectors by applying recurrence formula **at every time step** :

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \theta)$$



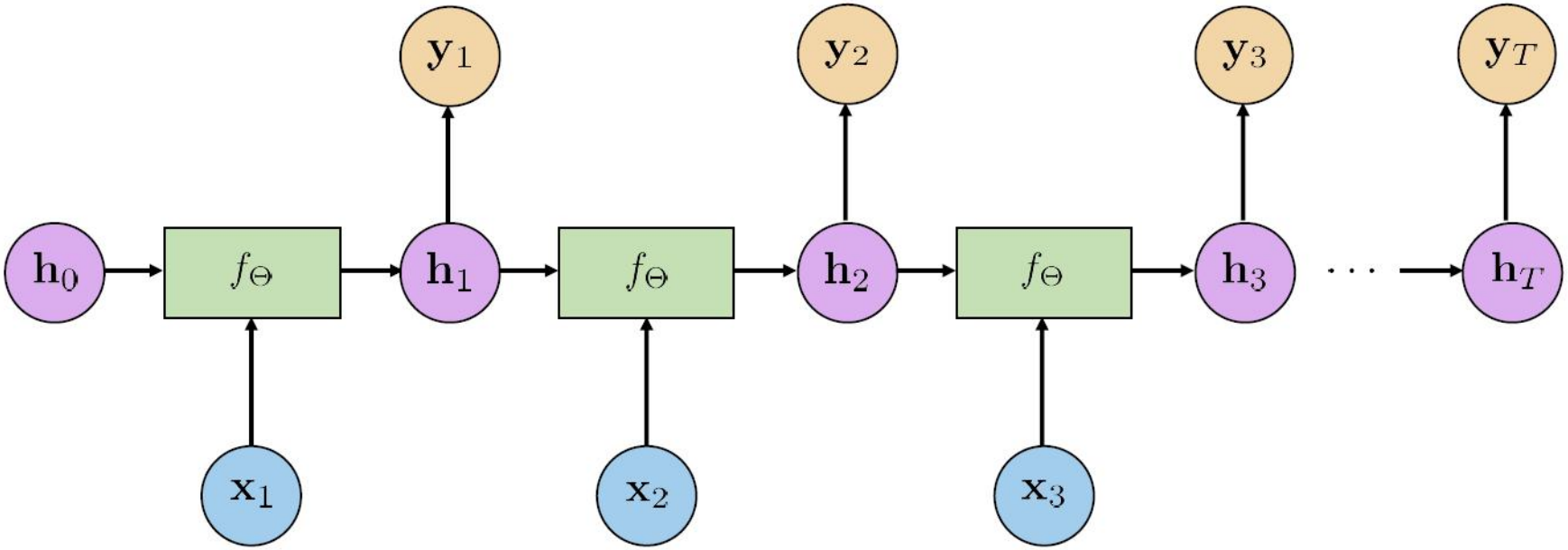
Recurrent Neural Networks (RNNs)

```
import torch
import torch.nn as nn

B, T, D, H, K = 2, 5, 16, 32, 10  # K = number of actions / classes

x = torch.randn(B, T, D)

rnn = VanillaRNN(input_size=D, hidden_size=H)
head = nn.Linear(H, K)  # decision head

y_rnn, hT_rnn = rnn(x)          # hT_rnn: (B, H)
logits = head(hT_rnn)           # (B, K)

print("logits shape:", logits.shape)
```



(EX) Recurrent Neural Networks (RNNs)

- Many-to-Many

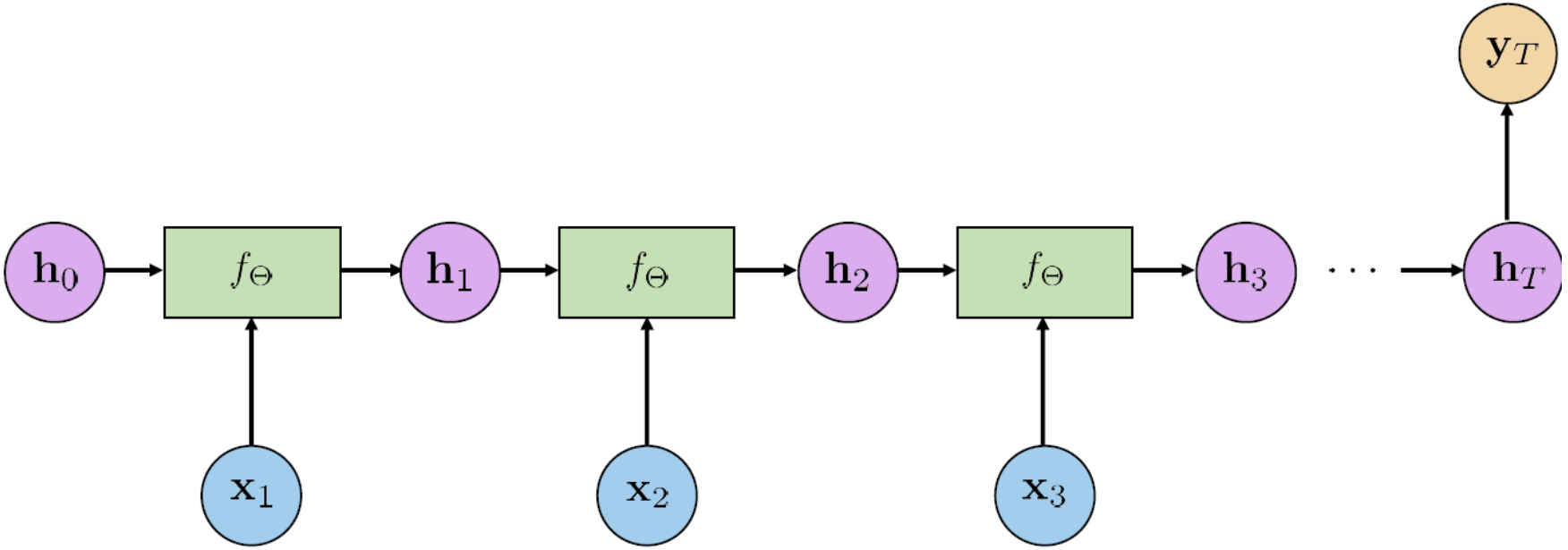


e.g., **Machine Translation**
(Sequence of words → Sequence of words)

Input sentence:	*Translation (PBMT):*	*Translation (GNMT):*	*Translation (human):*
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

(EX) Recurrent Neural Networks (RNNs)

- Many-to-One

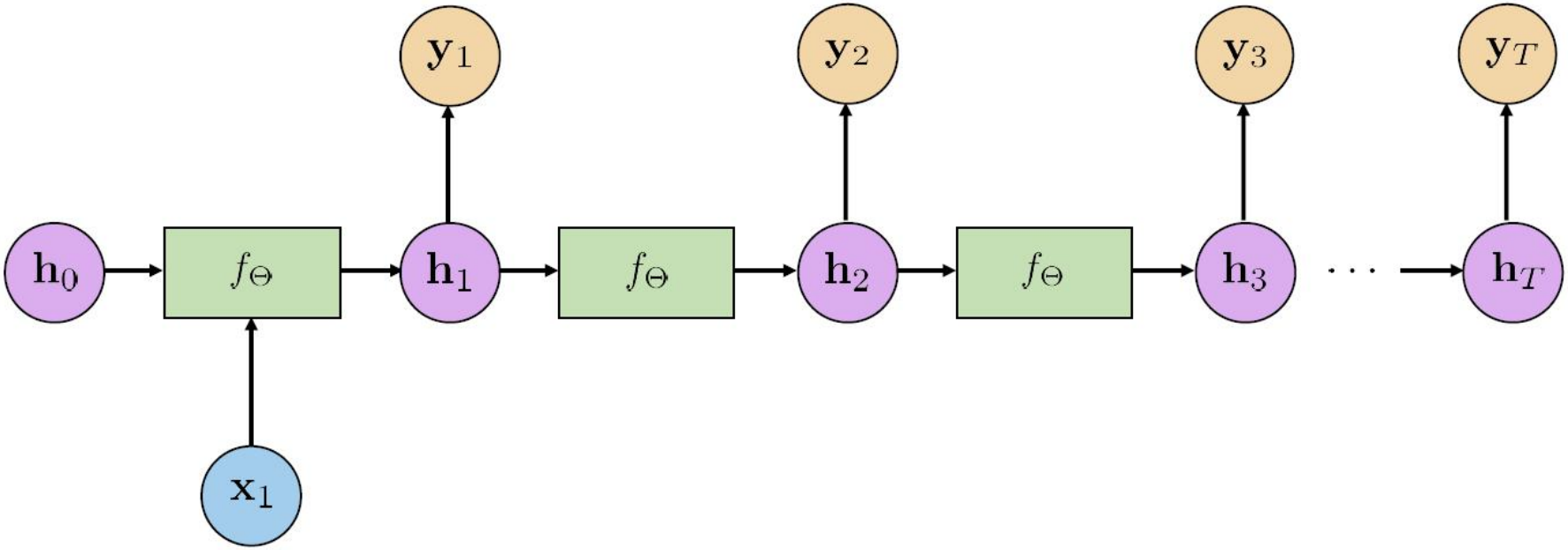


e.g., **Sentiment Classification**
(Sequence of words → sentiment)

→ Good paper or not?

(EX) Recurrent Neural Networks (RNNs)

- Many-to-One

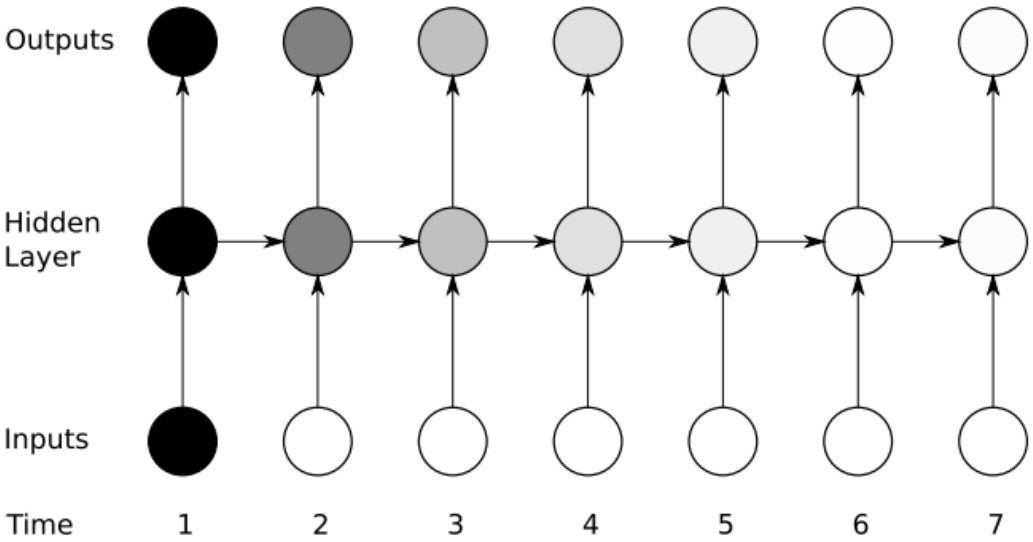


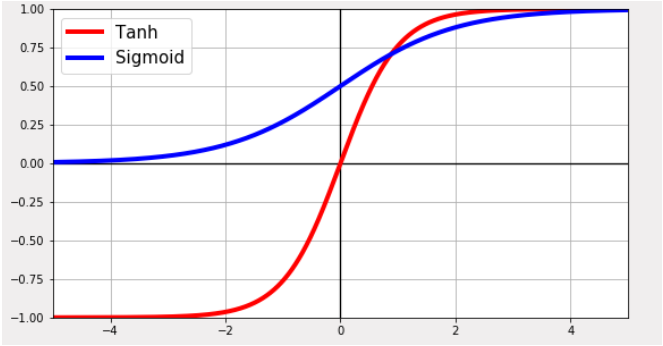
e.g., **Image Captioning**
(Image → sequence of words)

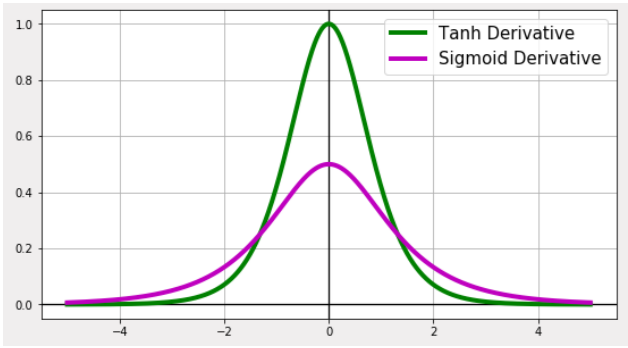
Vanishing Gradient Over Time

✓ 미분값의 상한이 1보다 작음

- This is more problematic in vanilla RNN (with tanh/sigmoid activation)
  - When trying to handle long temporal dependency
  - Similar to previous example, the gradient vanishes over time



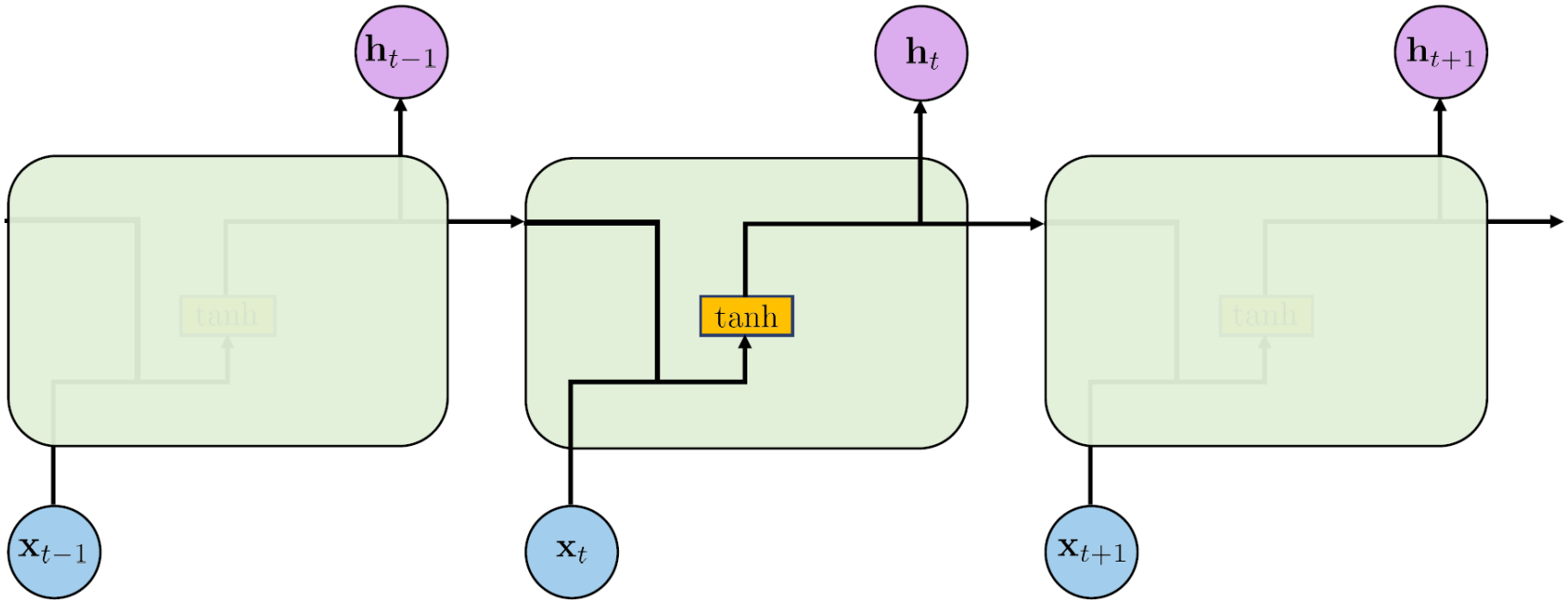




RNNs vs LSTM

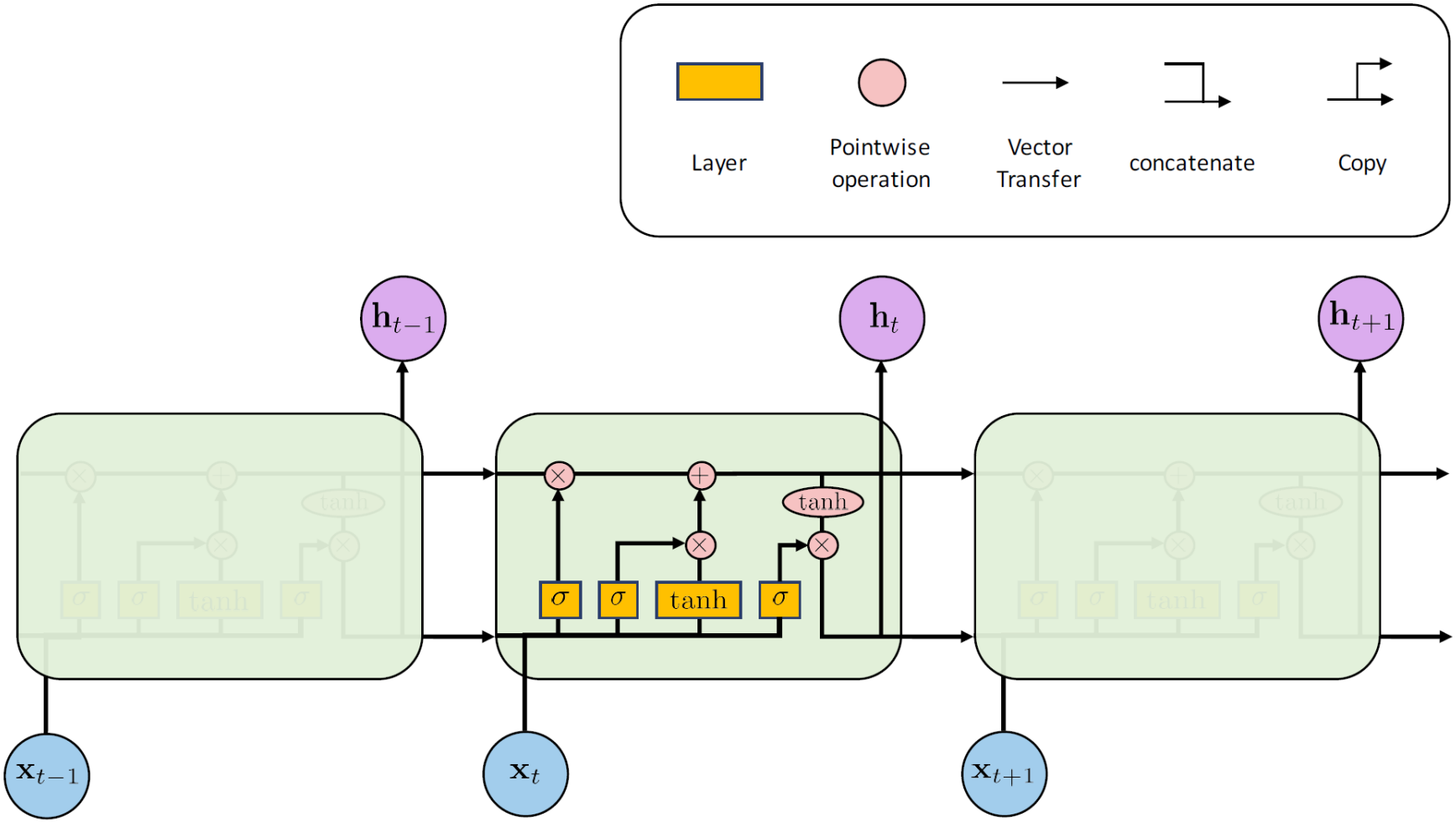
- RNNs

$$\mathbf{h}_t = tanh(\theta_{hh}\mathbf{h}_{t-1} + \theta_{xh}\mathbf{x}_t + b)$$



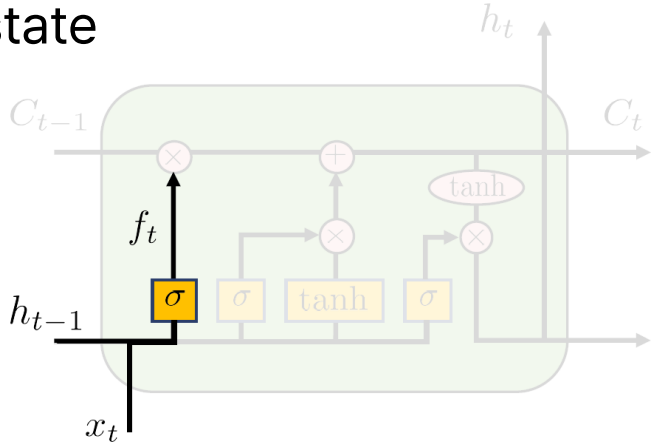
RNNs vs LSTM

- LSTM



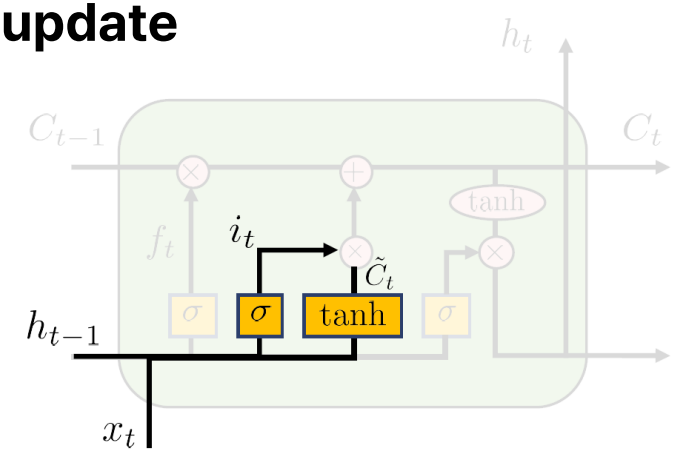
LSTM

- Step 1 : Decide what information we're going to **throw away** from the cell state



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

- Step 2 : Decide what information we're going to **store** in the cell state and **update**

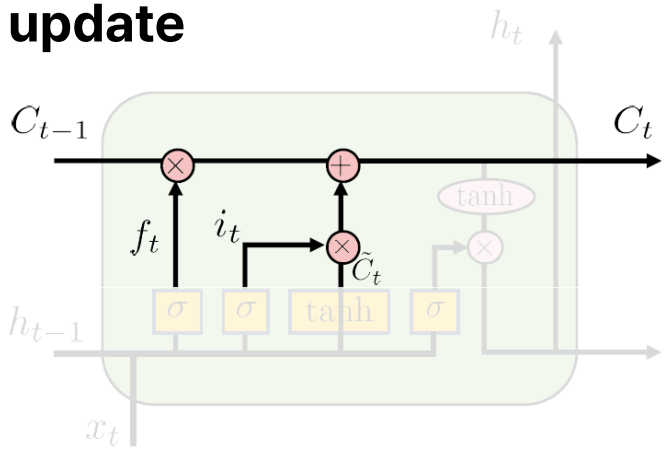


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

LSTM

- Step 2 : Decide what information we're going to **store** in the cell state and **update**

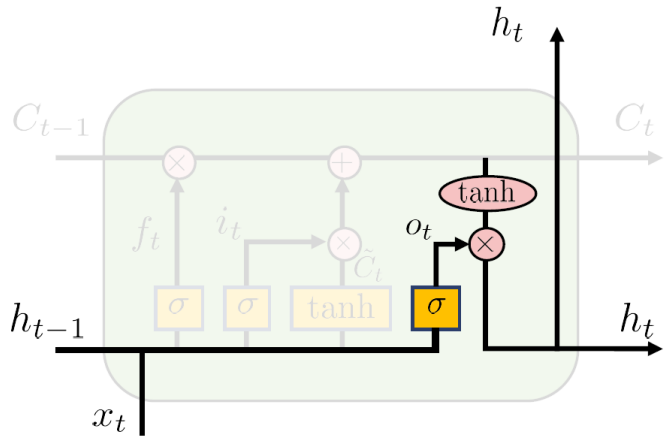


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

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

- Step 3 : Decide what we're going to **output**

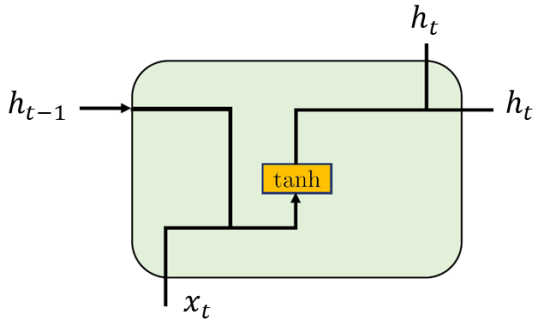


$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

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

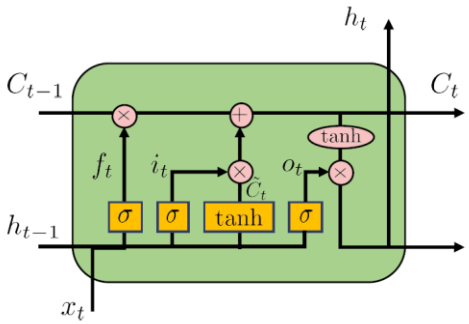
RNNs vs LSTM vs GRU

- RNNs



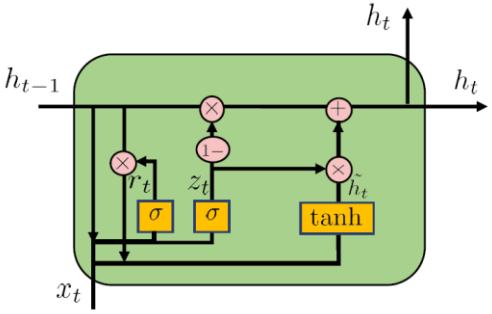
$$h_t = tanh(\theta_{hh}\mathbf{h}_{t-1} + \theta_{xh}\mathbf{x}_t + b)$$

- LSTM



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

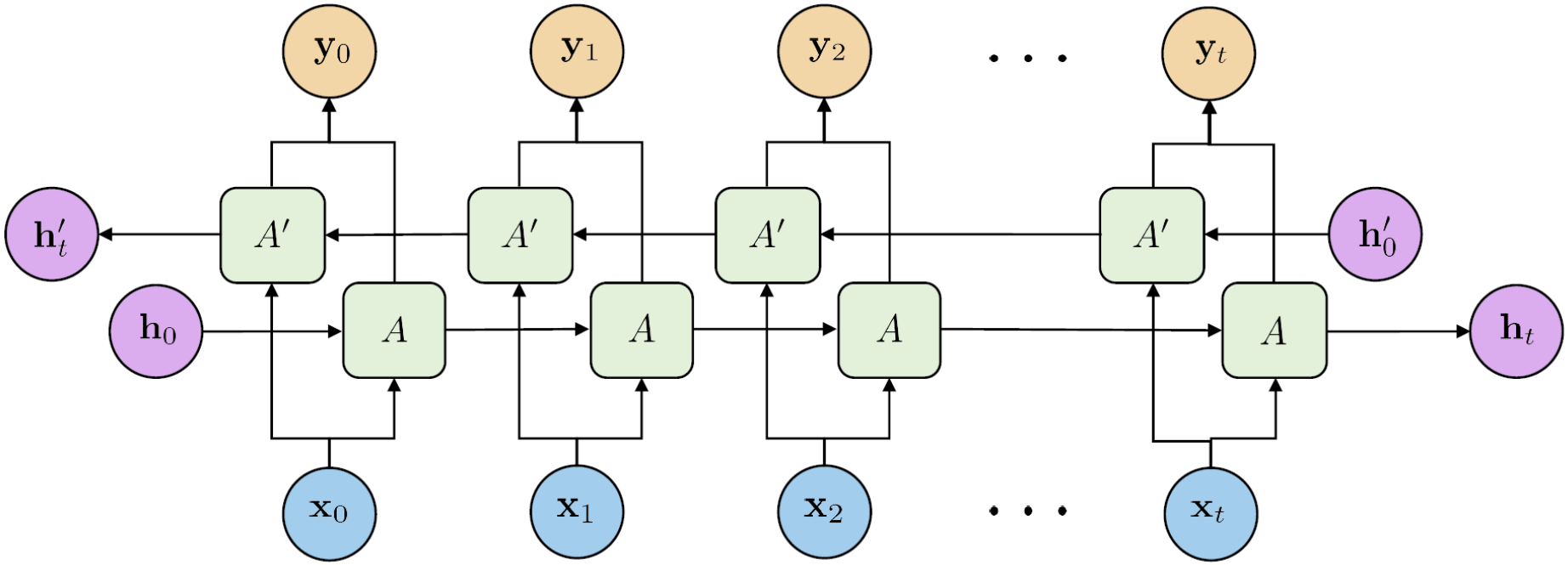
- GRU



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Bidirectional RNNs

- We can also extend RNNs into bi-directional models
  - The repeating blocks could be any types of RNNS (Vanilla RNN, LSTM, or GRU)
  - The only difference is that there are additional paths from future time steps



CONTENT

- RNN Architectures
- **Combinatorial Optimization**
- Pointer Network
- GFlowNet

What is Combinatorial Optimization?

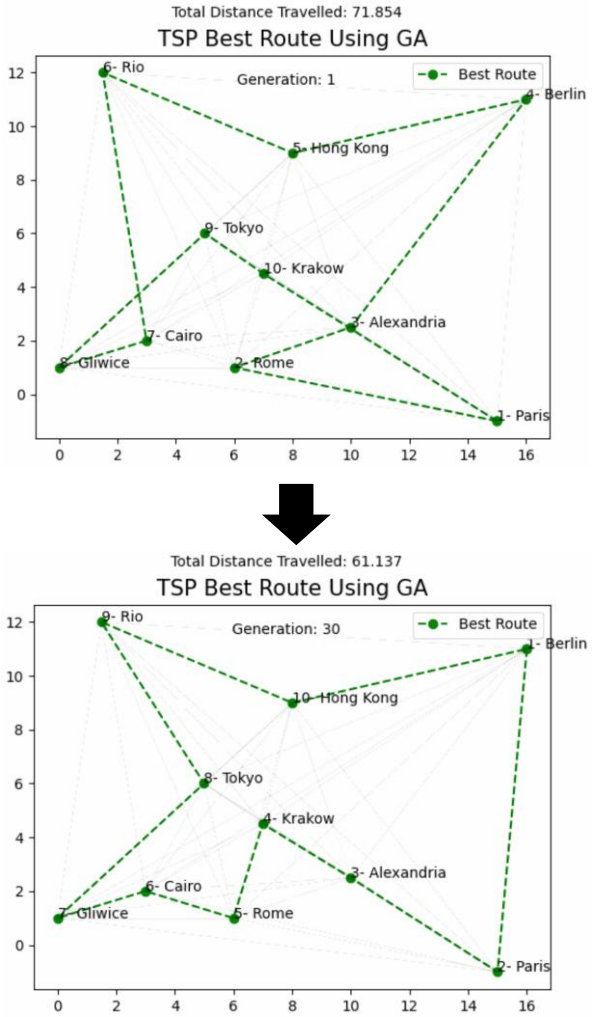
✓ 조합 최적화 문제

유한한 선택지 속에서, 주어진 제약 조건을 만족하며,
목적 함수를 최적화(최소화/최대화)하는 해를 찾는 문제

- Key Characteristics
  - 이산적 특성
    - 연속적인 값(Continuous)이 아닌, 이산적인(Discrete) 값을 다룸
    - 단순한 정렬이 아니라, 복잡한 관계를 고려한 최적의 순서나 조합을 찾아야 함
  - 미분의 어려움
    - 해의 공간이 불연속적(계단형) →해 자체에 대한 Gradient Descent가 불가능함
    - 아주 조금(0.001) 움직이면 비용이 얼마나 줄어들지(기울기) 계산할 수 없음

- Examples
  - Traveling Salesman Problem (TSP): 모든 도시를 한 번씩 방문하는 최단 경로 찾기
  - Vehicle Routing Problem (VRP): 용량이 제한된 트럭들의 배달 경로 최적화
  - Job-Shop Scheduling: 한정된 자원과 시간 내 작업 순서 배치
  - Knapsack Problem: 한정된 무게 내에서 가치가 가장 높은 물건 담기

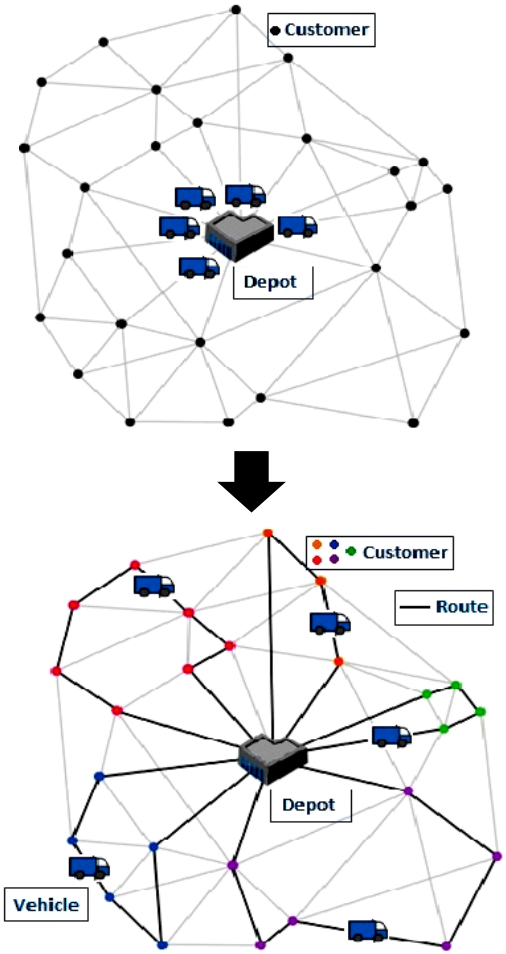
Traveling Salesman Problem (TSP)



- Input
  - 도시 좌표 (N): 방문해야 할 N개의 도시 위치 (x_i, y_i)
  - 거리 행렬 d_{ij}: 도시 i에서 도시 j로 이동하는 비용(거리)
    - 좌표가 주어지면 유클리드 거리로 계산 가능
- Output
  - 방문 순서: 모든 도시를 한 번씩 방문하고 돌아오는 순열
    - ex) 도시 1 → 도시 3 → 도시 2 → … → 도시 1
  - 의사결정 변수 (x_{ij}): 도시 i에서 j로 이동하면 1, 아니면 0
    - $x_{ij} \in \{0, 1\}$
- Objective
  - Minimize: $\sum d_{ij} x_{ij}$ (총 이동 거리의 최소화)
  - Subject to:
    - 방문 제약: 각 도시에 정확히 한 번 들어가고, 한 번 나와야 함
    - Subtour 방지: 전체가 하나의 큰 원을 이뤄야 함

Vehicle Routing Problem (VRP)

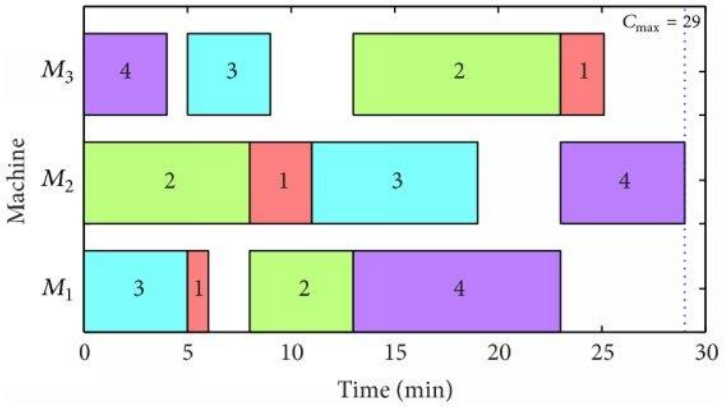
- Input
  - 물류 센터: 모든 트럭의 출발지이자 도착지
  - 고객 위치 (N): 방문해야 할 배송지 좌표 (x_i, y_i)
  - 고객 수요량 (q_i): 각 고객이 요청한 물건의 개수
  - 차량 정보: 트럭의 대수 (K) 및 트럭당 최대 적재 용량 (Q)
- Output
  - 차량별 경로 집합: K대의 트럭 각각에 할당된 배송 순서
    - ex) 트럭 A: 센터 → 고객 1 → 고객 3 → 센터
    - ex) 트럭 B: 센터 → 고객 2 → 고객 4 → 센터
  - 의사결정 변수 (x_{ijk}): 차량 k가 지점 i에서 j로 이동하면 1, 아니면 0
- Objective
  - Minimize: $\sum d_{ij} x_{ij}$ (모든 트럭의 총 이동 거리 합 최소화)
  - Subject to:
    - 용량 제약: 한 트럭이 싣는 물건의 합은 용량 Q를 넘을 수 없음 ($\sum q_i \leq Q$)
    - 방문 제약: 모든 고객은 정확히 한 대의 트럭에 의해 한 번만 방문됨
    - 경로 제약: 모든 트럭은 센터에서 출발하여 센터로 복귀해야 함



Job-Shop Scheduling

- Input
  - 작업 및 기계 (N, M): 처리해야 할 N개의 작업과 M개의 기계
  - 작업 명세 (Job Specification):
    - 기술 순서: 각 작업이 거쳐야 하는 기계의 정해진 순서 (Constraint)
      - ✓ ex) 작업 A: 기계 1 → 기계 3 → 기계 2
    - 소요 시간 (p_{ij}): 작업 j가 기계 i에서 처리되는 데 걸리는 시간

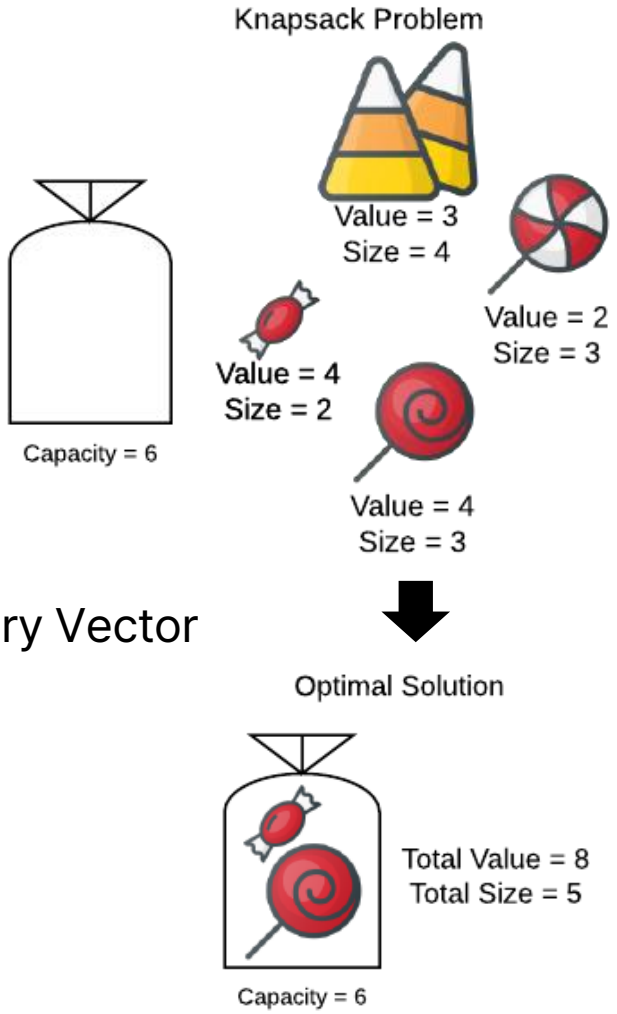
- Output
  - 의사결정 변수 (S_{ij}): 모든 작업의 각 공정별 시작 시각
  - 기계별 작업 순서: 각 기계가 어떤 순서로 작업을 처리할지 결정
    - ex) 기계 1: 작업 B → 작업 A → 작업 C 순서로 처리



- Objective
  - Minimize: C_{max} (Makespan, 총 완료 시간의 최소화)
  - Subject to:
    - 선행 관계 제약: 작업 내의 앞 단계가 끝나야 다음 단계를 시작 가능
    - 자원 제약: 한 기계는 한 번에 하나의 작업만 처리 가능

Knapsack Problem

- Input
  - 가방 용량 (C): 담을 수 있는 최대 무게 한계치
  - 물건 리스트: N개의 물건 각각에 대한 속성
    - 가치 (v_i): 선택 시 얻을 수 있는 보상
    - 무게 (w_i): 선택 시 소모되는 비용



- Output
  - 의사결정 변수 (x): 각 물건의 선택 여부를 나타내는 Binary Vector
    - $x = [x_1, x_2, \dots, x_N]$ (단, $x_i \in \{0, 1\}$)
    - ex) $[1, 0, 1, 0, 0]$ (1번, 3번 물건 선택)

- Objective
  - Maximize: $\sum v_i x_i$ (총 가치의 최대화)
  - Subject to: $\sum w_i x_i \leq C$ (총 무게는 용량 C 이하)

Why is Combinatorial Optimization Hard?

- [C1] Discreteness
  - 변수가 실수라면 미분이나 Simplex 알고리즘으로 빠르게 최적해 도달 가능
    - by Linear Programming (선형 계획법)
  - 하지만 0과 1로만 딱 떨어져야 한다는 제약 때문에, 사실상 모든 조합을 따져봐야 함
    - by Integer Programming (정수 계획법)

- [C2] Combinatorial Explosion
  - Search Space: 정답을 찾기 위해 탐색해야 하는 경우의 수가 데이터(N)에 따라 폭발적으로 증가
  - Complexity Growth:
    - 2^N (Selection): Knapsack 문제 ($N=30 \rightarrow 10$억 개)
    - $N!$ (Sequence): TSP 문제 ($N=20 \rightarrow 243$경 개, 2.4×10^{18})
  - The Reality: 슈퍼컴퓨터로도 전수 조사(Brute Force)가 불가능한 NP-Hard 난이도

Traditional Approaches: Exact Solver

✓ Guaranteeing Optimality at the Cost of Time

- General-Purposed Solver
  - ex) Gurobi, CPLEX, Google OR-Tools
- Problem-Specific Solver
  - ex) Concorde

Traditional Approaches: Heuristic Solver

✓ Fast Approximations, but Limited Flexibility

- General-Purposed Solver
  - ex) GA (Genetic Algorithm)

- Problem-Specific Solver
  - ex) LKH (Lin-Kernighan-Helsgaun)

CONTENT

- RNN Architectures
- Combinatorial Optimization
- **Pointer Network**
- GFlowNet

Related Works

- [2015][NIPS][-] Pointer Networks (cite:4521)
  - Google Brain, Department of Mathematics, UC Berkeley
  - https://arxiv.org/pdf/1506.03134

Why Pointer Networks?

✓ Guaranteeing Optimality

- Exact Solver (e.g., Concorde, Gurobi)
  - 계산 시간이 너무 느림 (문제 크기에 따라 지수적 증가)

✓ Fast Approximations, but Limited Flexibility

- Heuristic Solver (e.g., LKH, OR-Tools)
  - 상대적으로 빠름
  - 문제별로 수작업 규칙 설계 필요 (유연성 부족)
  - GPU 병렬 처리 어려움
  - 새로운 문제마다 알고리즘 재설계 필요

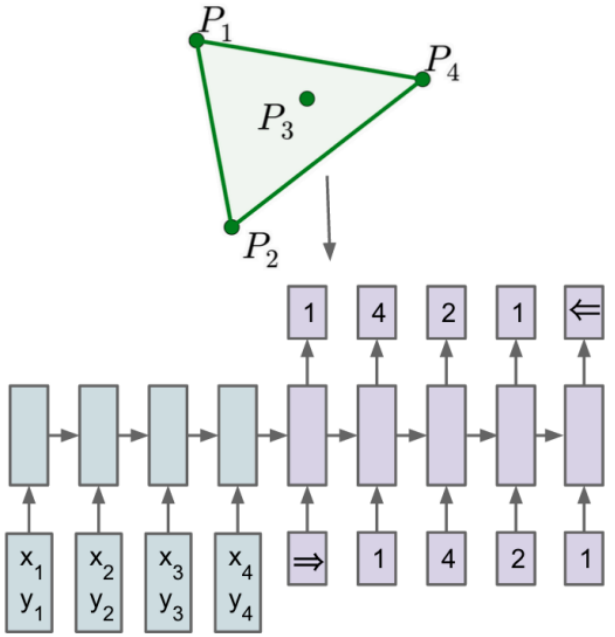
- Pointer Network:

"GPU로 빠르게 병렬 처리가 가능하면서,
데이터만 주면 알아서 규칙을 배우는 유연한 모델은 없을까?"

What is Pointer Networks?

- Sequence-to-Sequence
  - 일반적인 분류 모델: 출력 개수(Class)가 고정되어 있음
  - 출력 차원이 고정되어 있어서 문제 크기가 바뀌면 모델을 다시 만들어야 함

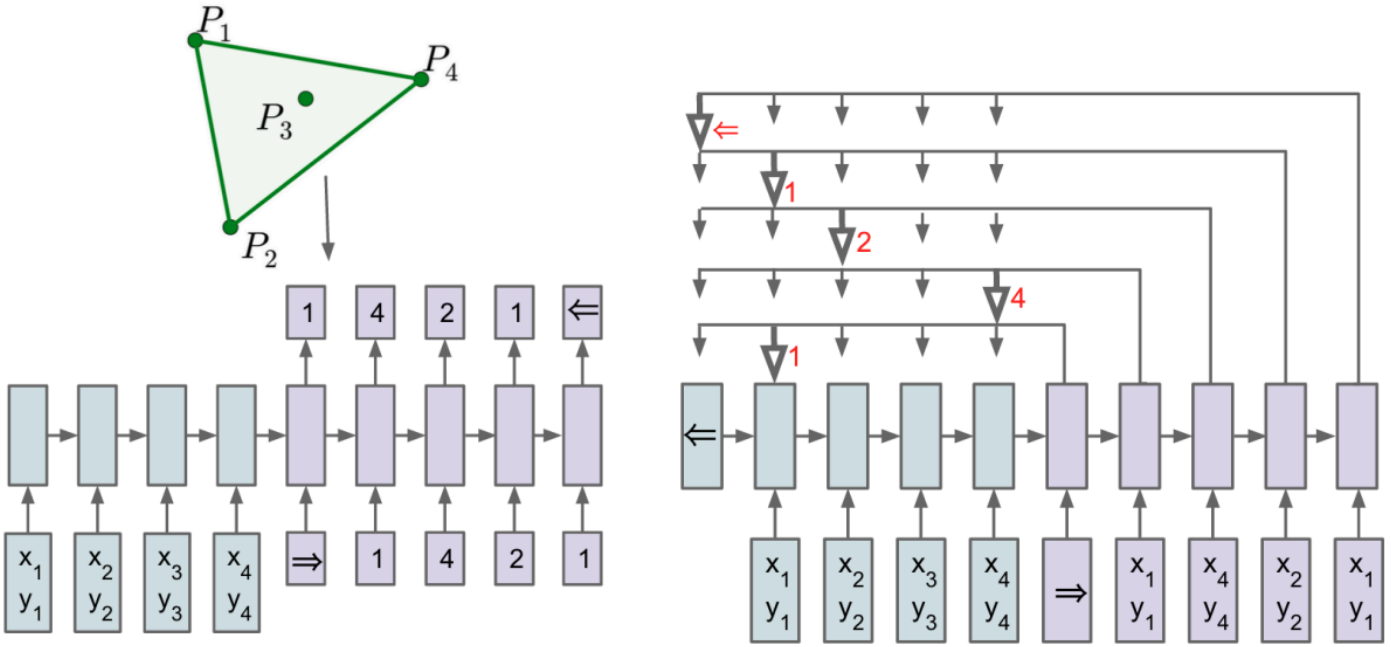
```
# TSP 10개 도시
vocab_size = 10
model_10 = Seq2Seq(vocab_size=10)  # 10개 출력

# TSP 100개 도시
vocab_size = 100
model_100 = Seq2Seq(vocab_size=100)  # 100개 출력
# 재학습 필요!
```



(a) Sequence-to-Sequence

What is Pointer Networks?

- Pointer Network
  - 가변적인 입력/출력 사이즈를 유연하게 처리할 수 있음
  - 입력된 시퀀스의 원소 중 하나를 '가리켜서(Point)' 출력으로 가져오는 구조



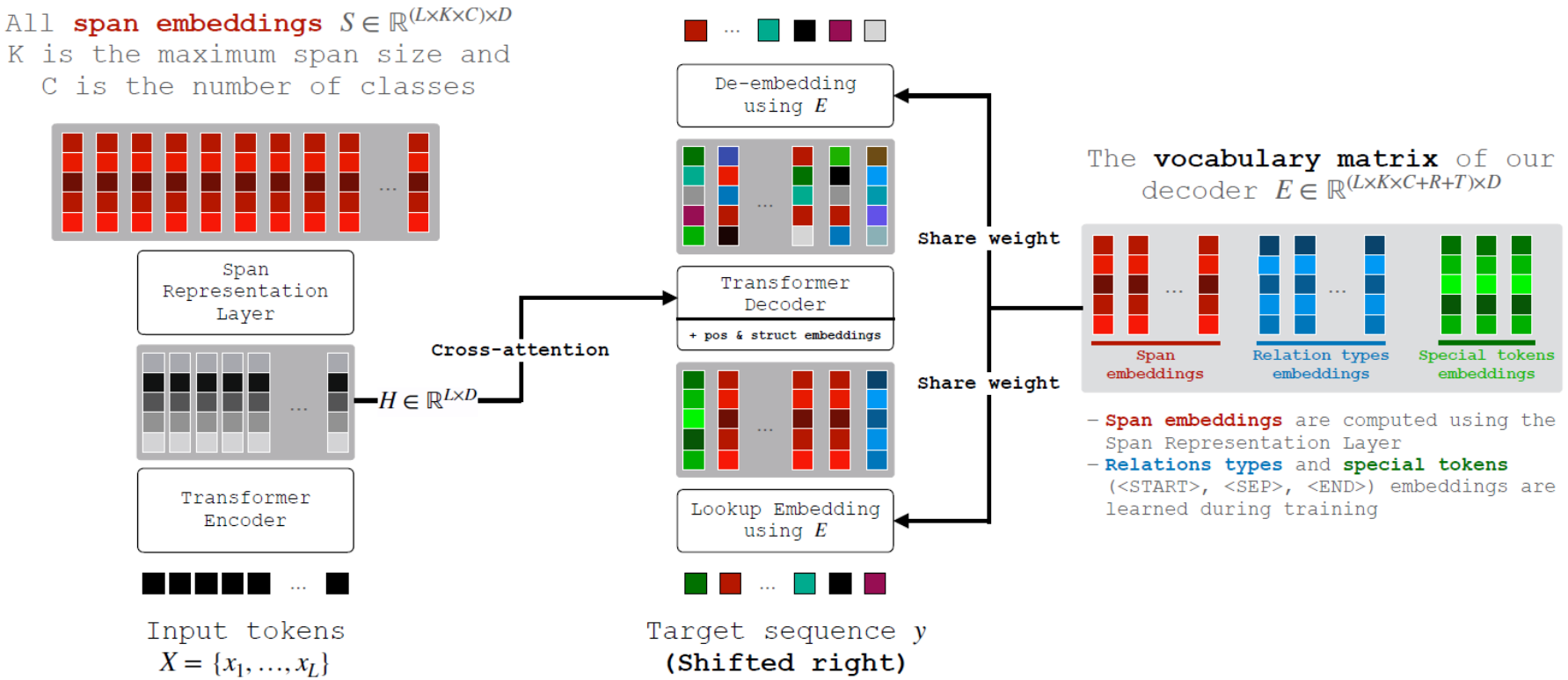
(a) Sequence-to-Sequence

(b) Ptr-Net

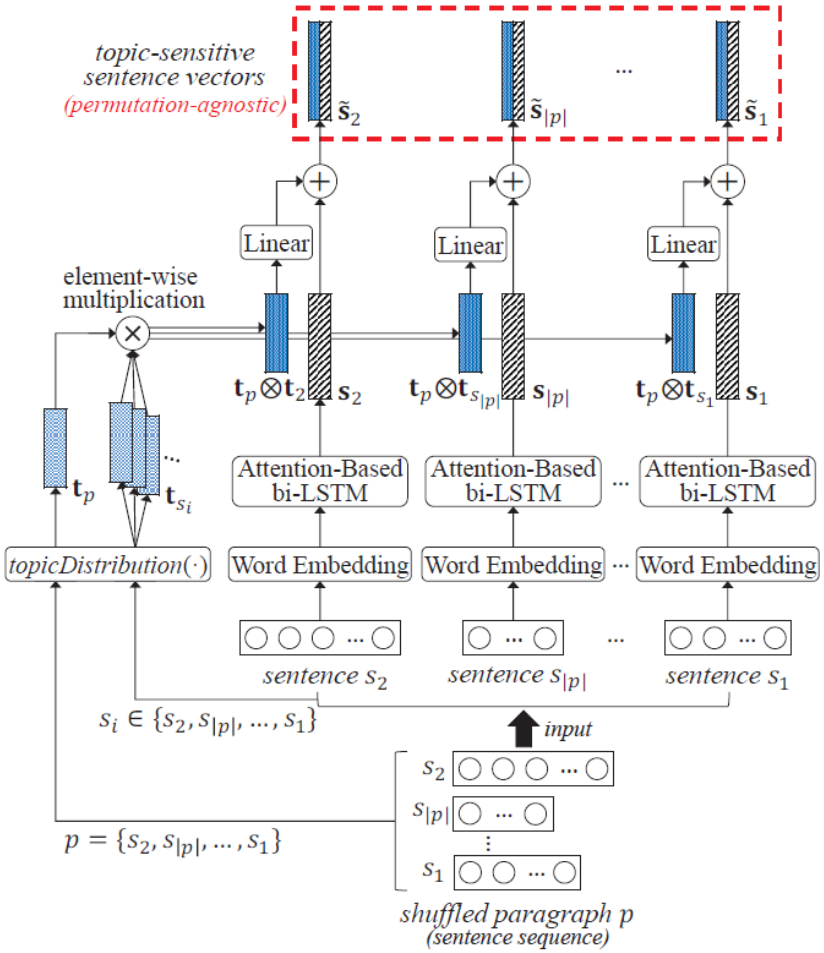
(EX) Sorting Alphabet Sequence

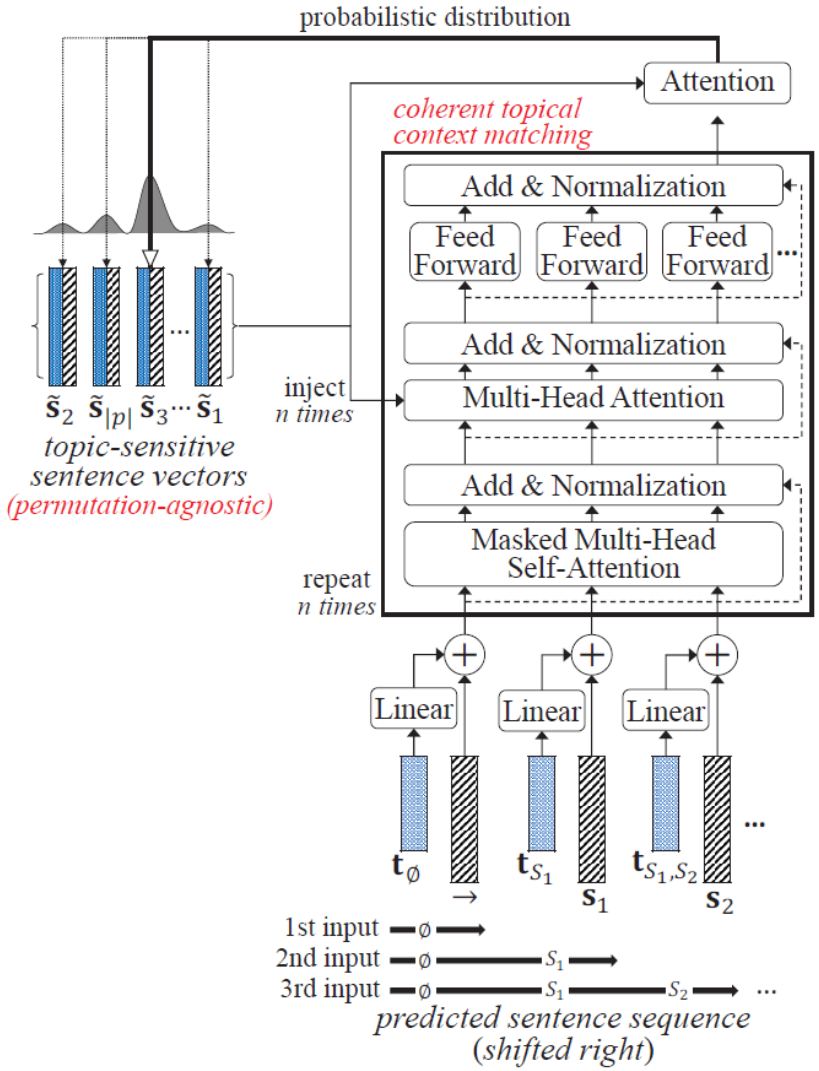
- https://colab.research.google.com/drive/1-vAqvgakOg3VKX3yI3F5OcbqcpImjyiK?usp=sharing

(EX) ATG (AAAI'24)



(EX) TGCM (EMNLP'19)





Limitations of Pointer Networks

- Pointer Networks
  - 장점: 학습이 매우 빠르고 안정적
  - 단점:
    - 데이터 생성 비용: 데이터가 많으면, 정답 Label을 만드는 외부 Solver가 너무 느려짐
    - 성능에 한계가 존재함
      - ✓ 모델은 solver가 제공한 정답을 모방할 뿐, 새로운 해법을 발견하지 못함
        -> From Supervised Learning to Reinforcement Learning

- From Supervised Learning to Reinforcement Learning
  - 장점: 직접 탐색하여 이론적 최적해를 찾을 수 있음
  - 단점:
    - 만약 정답 후보가 여러 개라면? RL은 최적해 하나에만 집중함
      -> GFlowNet: 최적해 탐색(Maximization)에서 분포 학습(Sampling)으로 확장

CONTENT

- RNN Architectures
- Combinatorial Optimization
- Pointer Network
- **GFlowNet**

Related Works

- [2021][NIPS][GFlowNet] Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation (cite:516)
  - Mila, McGill University, Université de Montréal, DeepMind, Microsoft
  - https://yoshuabengio.org/2022/03/05/generative-flow-networks/
  - https://github.com/GFNOrg/gflownet

- [2022][UAI][-] Bayesian Structure Learning with Generative Flow Networks (cite:197)

- [2023][JMLR][GFlowNet] GFlowNet Foundations (cite:374)
  - Mila, Université de Montréal
    Canadian Institute for Advanced Research (CIFAR)

Motivation

Sampling Diverse Structured Objects

- 분자/단백질 설계에서 주로 쓰임
  - 다양한 좋은 후보들이 필요하기 때문에,
    "**보상** 비례 다양성"이 분자 설계(**생성**)에 이상적임
  - 생성한 후보들을 **점수화**할 객관적 지표 존재함
    - 결합 친화도, 합성 가능성, 독성 등

- 그 외에도, 일반적인 조합 최적화 문제에 활용될 수 있음
  - ex) 도로망, 전력망, 통신망 설계
  - ex) 여러 최적 토폴로지 탐색
  - ex) 그래프 생성

- 반면, 텍스트 생성에서는"하나의 최고 답변 " 을 선호함
  - 또한, 보상 함수 정의 어려움

Related Keywords

- Generative Flow Networks (GFlowNets)
  - A framework for **sampling diverse structured objects**

- A New Research Direction at the Intersection of
  - Reinforcement Learning: "어떤 행동을 해야 기대 보상이 최대가 되는가?"
    - 보상 함수를 정의하고 → **기대 누적 보상을 최대화**하는 정책을 학습

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi}[R(\tau)]$$

  - Deep generative Models: "데이터 분포 $p(x)$를 어떻게 모델링하고 샘플링할까?"
    - 주어진 데이터셋의 분포 전체를 직접적으로 **모방**하는 모델

$$\min D(p_{\text{model}}(x), p_{\text{data}}(x))$$

  - Energy-based Probabilistic Modelling: "무엇이 좋은 상태인가?"
    - 데이터들을 **점수화**해서 간접적으로 확률분포를 모델링
    - 판단이 아니라 상대적 가능도(density)를 정의
      - ✓ 이진 분류가 아닌 연속적인 확률 형태

$$p(x) = \frac{1}{Z} e^{-E(x)}$$

Introduction

- Existing Approaches
  - standard return maximization tends to converge to **a single return-maximizing sequence**

$$\max_{\pi} \mathbb{E}_{\tau \sim \pi}[R(\tau)]$$

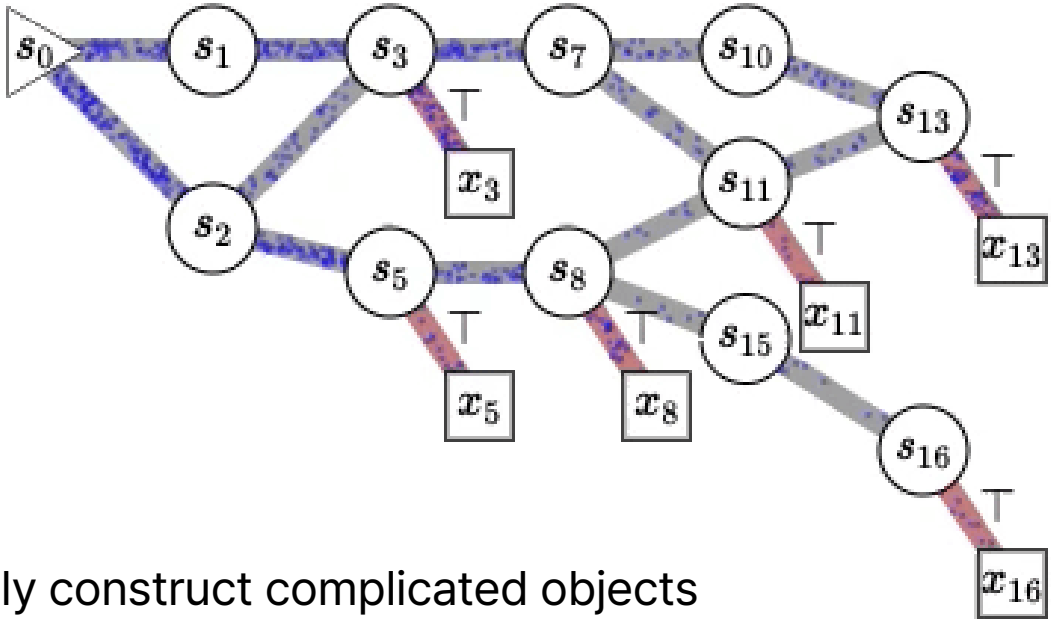
    - ex) Policy Gradient (REINFORCE, PPO), Q-learning, Actor-Critic

- Learning a stochastic policy for generating an object (like a molecular graph) from a sequence of actions
  - we would like to sample a **diverse set of high-return solutions**
  - the probability of generating an object is **proportional** to a given positive reward for that object

$$\pi(x) \approx \frac{R(x)}{Z} = \frac{R(x)}{\sum_{x' \in \mathcal{X}} R(x')}$$

GFlowNet

- We think about the flow of unnormalized probabilities, similar to the amount of water flowing from an initial state
  - trajectories correspond to all the possible sequences of actions
    - actions that determine state transitions



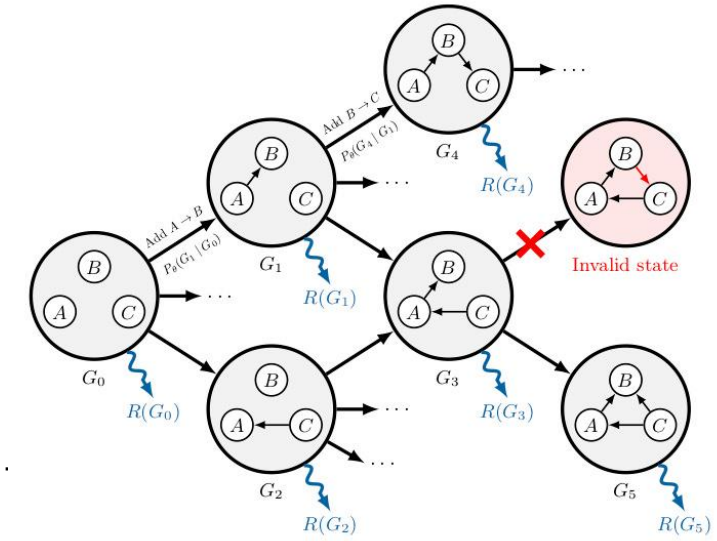
square nodes with red transitions correspond to terminal states

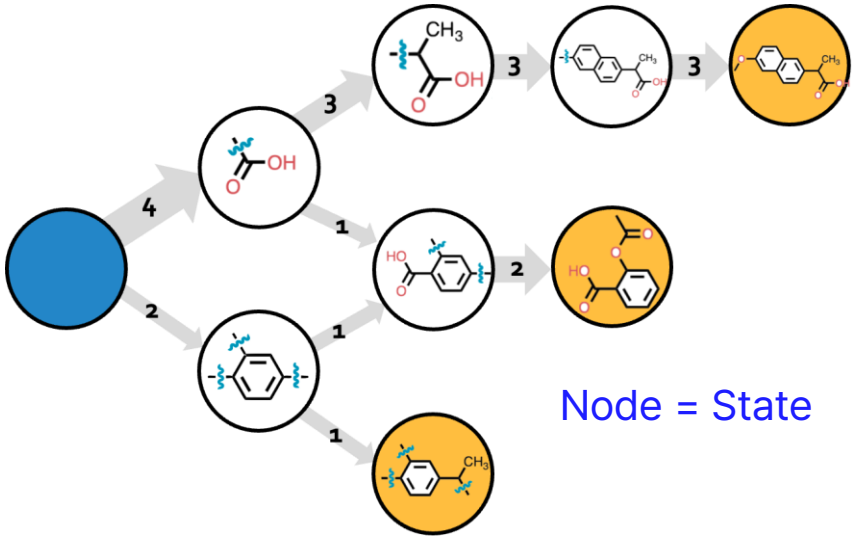
  - sequentially construct complicated objects
    - ex) molecular graphs, causal graphs

https://yoshuabengio.org/2022/03/05/generative-flow-networks/

Problem Definition

- DAG (Directed Acyclic Graph) 구조
  - States: $S = \{s0, s1, s2, s3, s4, \dots\}$
  - Actions: $A = \{a1, a2, a3, a4, a5, a7, \dots\}$
  - Transition: $T(s, a) \rightarrow s'$ (deterministic)
- Reward Function
  - R(s): S → $\mathbb{R}$+ / R(s_4) > 0 (terminal state의 보상) / R(s) = 0 (non-terminal states)
- Goal: Flow equations을 만족하는 $F(s, a)$: $S \times A \rightarrow \mathbb{R}+$ 찾기

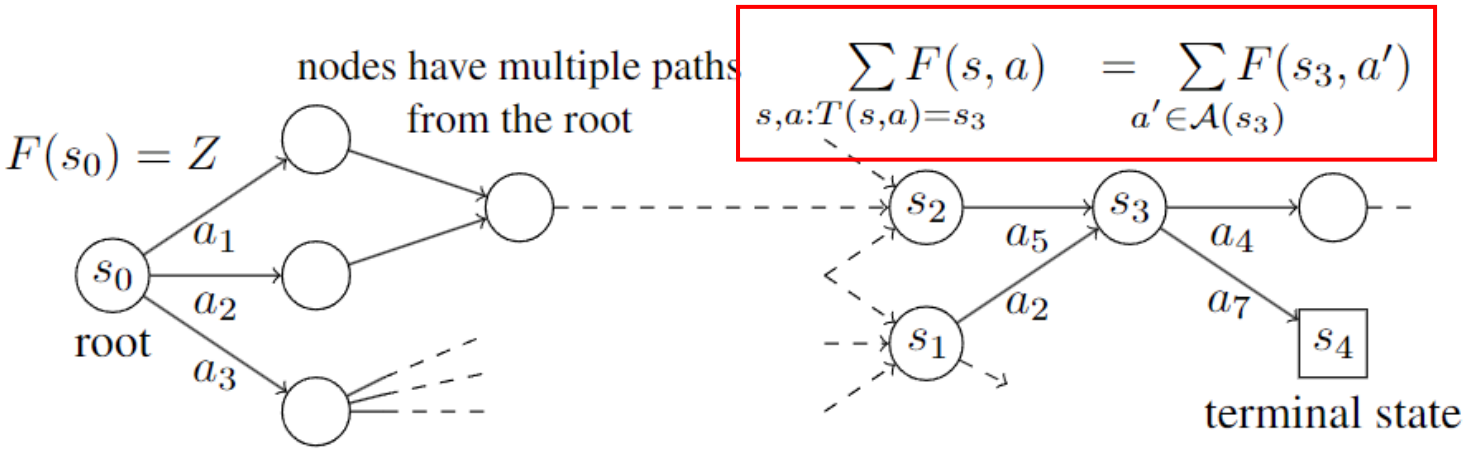




GFlowNet

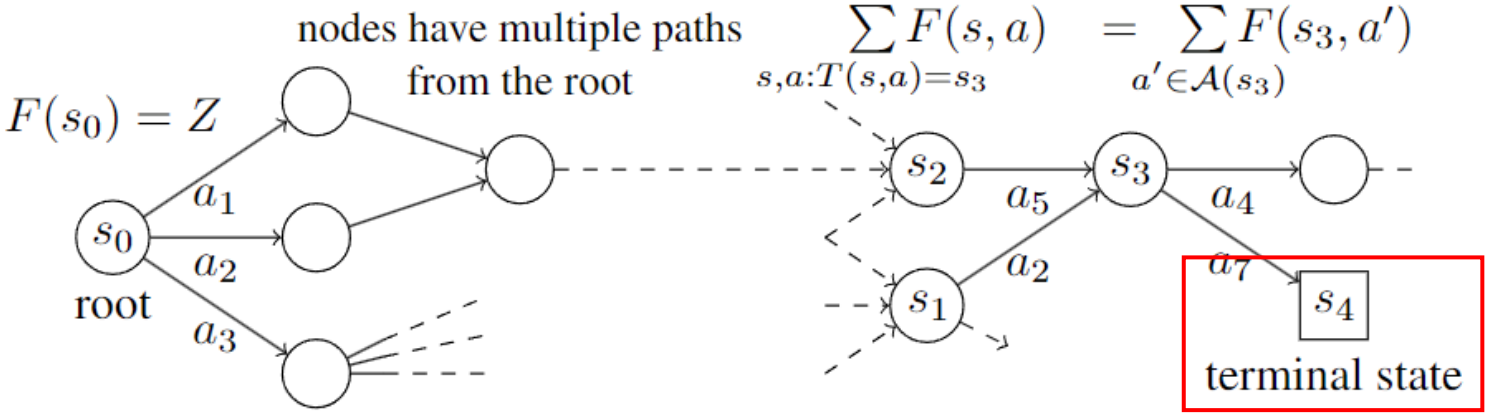
- A Flow Network MDP
  - episodes start at source s_0 with flow Z (there are no cycles)
  - The goal is to estimate $F(s,a)$ such that the flow equations are satisfied for all states: for each node, incoming flow equals outgoing flow
    - s에서 action a를 선택하는 flow: $F(s,a) = F(s) \times policy\ \pi(a|s)$
    - s에서 a를 선택할 확률: $\pi(a|s) = F(s,a)/F(s)$

각 intermediate state에 들어가는 inflow의 양과 나오는 outflow의 양은 항상 같음



GFlowNet

- A Flow Network MDP
  - s가 terminal state라면 are sinks with out-flow $R(s)$
    - $R(s) = F(s)$ 가 해당 output의 보상이며 확률과 동일함



terminal state로 들어가는 inflow의 양은 각 terminal state 의 object가 가지는 score(혹은 reward) $R(s)$에 비례

$$\sum_{s,a:T(s,a)=s'} F(s,a) = R(s') + \sum_{a' \in \mathcal{A}(s')} F(s',a').$$

terminal state

Training 방법1: Flow Matching

- score에 비례하는 샘플을 생성하기 위해 flow의 양을 정확하게 추정하는 방향으로 학습을 진행
  - parameterize한 F_θ가 다음 식을 만족하도록 학습 (Flow $F(s,a)$ 값을 직접 모델링)

```
class FlowNetwork(nn.Module):
    def __init__(self, state_dim, action_dim,
                 hidden=128):
        super().__init__()

        self.state_encoder = nn.Sequential(
            nn.Linear(state_dim, hidden),
            nn.ReLU(),
            nn.Linear(hidden, hidden),
            nn.ReLU()
        )

        self.flow_head = nn.Sequential(
            nn.Linear(hidden, action_dim),
            nn.Softplus()  # F(s,a) > 0 보장
        )

        self.log_Z = nn.Parameter(torch.zeros(1))
```

```
def forward(self, state):
        """Returns: F(s,a) for all actions"""
        h = self.state_encoder(state)
        flows = self.flow_head(h)
        return flows

    def get_flow(self, state, action):
        """특정 (s,a)의 flow"""
        flows = self.forward(state)
        return flows[action]

    def get_state_flow(self, state):
        """F(s) = Σ F(s,a)"""
        flows = self.forward(state)
        return flows.sum()
```

Training 방법1: Flow Matching

- parameterize한 F_θ가 다음 식을 만족하도록 학습
  - 각 개별 state에서 들어오는 flow와 나가는 flow의 균형을 맞추는 방법

$$\min || \sum_{s' \in Child(s)} F_\theta(s \to s') - R(s') - \sum_{s \in Par(s')} F_\theta(s' \to s) ||$$

```
def flow_matching_loss(flow_net, state_batch, env):
    """모든 state에서 flow conservation 만족하도록"""

    total_loss = 0

    for state in state_batch:
        # === 1. Inflow 계산 ===
        inflow = 0
        parents = env.get_parents(state)

        for parent_state, action in parents:
            inflow += flow_net.get_flow(parent_state, action)

        # === 2. Outflow 계산 ===
        if env.is_terminal(state):
            outflow = torch.tensor(env.reward(state))
        else:
            outflow = flow_net.get_state_flow(state)
```

```
        # === 3. Conservation Loss ===
        loss = (inflow - outflow) ** 2
        total_loss += loss

    return total_loss / len(state_batch)
```

Training 방법2: Trajectory Balance

- Flow Matching 대비 장점
  - 하나의 trajectory당 하나의 loss term만 필요함
    - 모든 intermediate state에서 flow 보존 법칙을 만족해야 함

```
# Flow F(s,a) 값을 직접 모델링
flows = flow_net(state)  # → [F(s,a₀), F(s,a₁), ...]
```

  - trajectory의 확률 분포을 직접 최적화

```
# Policy P(a|s)를 모델링 (확률 분포)
logits = policy_net(state)  # → logitsprobs =
softmax(logits)             # → [P(a₀|s), P(a₁|s), ...]
```

  - 중간 상태마다 복잡한 flow 계산이 불필요

Training 방법2: Trajectory Balance

- trajectory의 확률 분포을 직접 최적화

```
class GFlowNet(nn.Module):
    def __init__(self, state_dim, action_dim, hidden=128):
        super().__init__()

        self.policy = nn.Sequential(
            nn.Linear(state_dim, hidden),
            nn.ReLU(),
            nn.Linear(hidden, action_dim)
        )

        self.log_Z = nn.Parameter(torch.zeros(1))

    def forward(self, state):
        """Returns: action logits"""
        return self.policy(state)

    def sample(self, state):
        logits = self.forward(state)
        return Categorical(logits=logits).sample()
```

```
def tb_loss(gfn, trajectories):
    losses = []

    for states, actions, reward in trajectories:
        # Forward: Σ log P(a|s)
        log_pf = sum(
            F.log_softmax(gfn(s), -1)[a]
            for s, a in zip(states[:-1], actions)
        )

        # Backward: uniform
        log_pb = -torch.log(torch.tensor(len(states)
- 1.0))

        # TB objective
        loss = (log_pf + gfn.log_Z - log_pb -
torch.log(reward)) ** 2
        losses.append(loss)

    return torch.stack(losses).mean()
```

Advanced Optimization

- Advanced Training 방법들
  - Detailed Balance (DB) – 2022
  - Sub-Trajectory Balance (SubTB) – 2022
  - Forward Looking GFlowNet – 2023
  - …

- RNN/Transformer 확장
  - RNN/LSTM/Transformer: Long-range dependency
  - GNN: Graph 구조 대응

Reference

- https://process-mining.tistory.com/216
- https://www.iro.umontreal.ca/~slacoste/teaching/ift6269/A23/notes/lecture23_slides.pdf
- https://www.neuroai.science/p/gflownets-sampling-on-sets-and-graphs
- https://milayb.notion.site/The-GFlowNet-Tutorial-95434ef0e2d94c24aab90e69b30be9b3
- https://yoshuabengio.org/2022/03/05/generative-flow-networks/