A Survey on Transformers in Reinforcement Learning, TLMR 2023

Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye

Xincao Xu

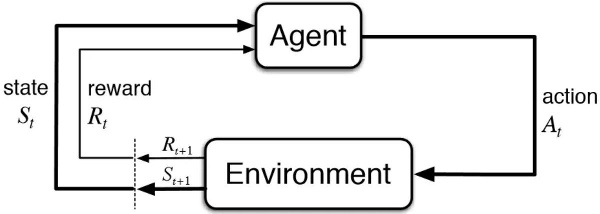
Shenzhen Institute for Advanced Study, UESTC

March 28, 2024

Reinforcement Learning (RL)

A mathematical formalism for ***sequential decision-making*** has provided by RL.

- *Environment*: Physical world in which the agent operates
- *State*: Current situation of the agent
- *Reward*: Feedback from the environment
- *Policy*: Method to map agents state to actions
- *Value*: Future reward that an agent would receive by taking an action in a particular state
- *Workflow*: $S_t \rightarrow$ ***Agent*** $\rightarrow A_t \rightarrow$ ***Environment*** $\rightarrow R_{t+1} + S_{t+1}$

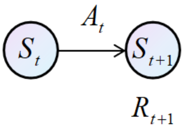


Markov Decision Process (MDP)

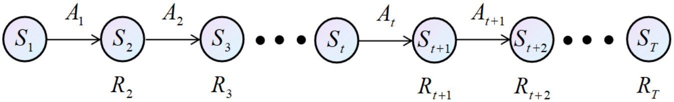
- Random variable: X, Y
- Stochastic process: $S_1, S_2, \cdots, S_{|T|}$
- Markov process: Stochastic process with ***Markov Property***

$$P(S_{t+1} \mid S_t, S_{t-1}, \cdots, S_1) = P(S_{t+1} \mid S_t) \tag{1}$$

- Markov decision process: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$
  - The state-space $\mathcal{S}$
  - The action-space $\mathcal{A}$
  - The state-transition probabilities $\mathcal{P}$
  - The reward function $\mathcal{R}$



Dynamics of MDP



- A trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$
- Dynamics function

$$p\left(s', r \mid s, a\right) \triangleq \Pr\left\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right\} \tag{2}$$

- State-transition probabilities

$$\begin{aligned} p\left(s' \mid s, a\right) &\triangleq \Pr\left\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\right\} \\ &= \sum_{r \in \mathcal{R}} p\left(s', r \mid s, a\right) \end{aligned} \tag{3}$$

Reward and Returns

- Reward function

$$R\left(s', s, a\right) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in \mathcal{R}} r p\left(s', r \mid s, a\right)}{p\left(s' \mid s, a\right)} \tag{4}$$

- Reward of taking action a in state s

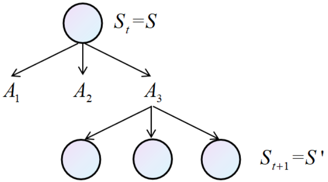
$$R(s, a) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p\left(s', r \mid s, a\right) \tag{5}$$

- Returns

$$G_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{6}$$

Discount factor $\gamma \in [0, 1]$

Policy and Goal



- Policy

$$\pi(a \mid s) \triangleq \Pr\left\{A_t = a \mid s_t = s\right\} \tag{7}$$

- Value function

$$v_\pi(s) \triangleq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \tag{8}$$

Goal

The agent seeks to formulate a policy π guiding its actions within an environment, with the objective of maximizing cumulative rewards.

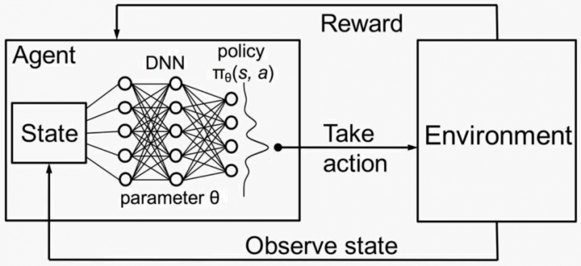
$$\max J(\pi) = \max \mathbb{E}_{\pi, P, \rho_0}\left[\sum_t \gamma^t r\left(s_t, a_t\right)\right] \tag{9}$$

where ρ_0 is distribution of initial states

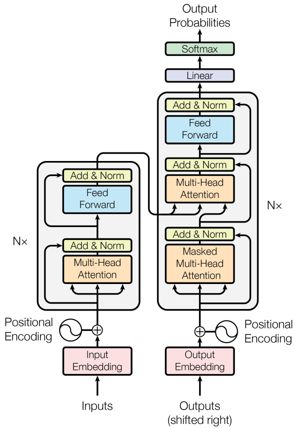
Deep Reinforcement Learning

Objective: Learning to represent states and learning to act
DRL: Deep neural networks (e.g., CNN, RNN, et al.) + RL

- To efficiently handle high-dimensional state and action spaces
- Enabling more complex and flexible decision-making processes
- Problem: Sample efficiency
- Solution: Introduce inductive biases into the DRL framework
- Choice of function approximator architectures, e.g., the parameterization of DNN



Transformer



Most effective and scalable neural networks to model sequential data

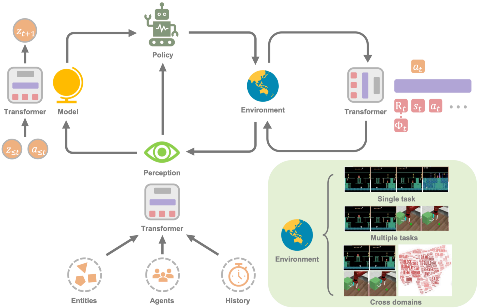
- To incorporate ***self-attention mechanism***
- Capturing dependencies within long sequences in an efficient manner
- Outperforming CNN and RNN across a wide range of supervised learning tasks

Question

Can we use Transformers to tackle the problems (i.e., learning to represent states, and learning to act) in RL?

1. Background
2. **Combination of Transformers and RL**
3. Future Perspectives
4. Insights

Roles of Transformers in RL



1) A representation/dynamic model
   - Representing the state
   - Capturing multi-step temporal dependencies to deal with the partial observability issue
2) A sequential decision-maker
   - Generating sequential decisions
   - Generalizing to multiple tasks and domains

Challenges & Taxonomy

Unique Challenges

- From the aspect of RL
- Ever-changing policy $\rightarrow$ Unstable training data $\rightarrow$ Non-stationarity for learning a Transformer
- Highly sensitive to network architectures
- From the aspect of Transformer
- Suffer from high computational and memory costs $\rightarrow$ Expensive
- Need a much larger amount of training data $\rightarrow$ Exacerbate the sample efficiency problem of RL

Four Classes

- Representation learning
- Model learning
- Sequential decision-making
- Generalist agents

Transformers for Representation Learning

Key idea

One natural usage of Transformers is to use it as ***a sequence encoder***. In fact, various sequences in RL tasks require processing, such as local per-timestep sequence, multi-agent sequence, temporal sequence, and so on.

- To process complex information from a variable number of entities scattered in the agents observation

$$\text{Emb} = \text{Transformer}\left(e_1, \cdots, e_i, \cdots\right) \tag{10}$$

where e_i represents the agents observation on entity i

- To process temporal sequence

$$\mathrm{Emb}_{0:t} = \text{Transformer}\left(o_0, \cdots, o_t\right) \tag{11}$$

where o_t represents the agents observation at timestep t and $\mathrm{Emb}_{0:t}$ represents the embedding of historical observations from initial observation to current observation

Transformers for Model Learning

Key idea

Transformer architecture serves as ***the backbone of the world model*** in model-based RL. Distinct from the prediction conditioned on single-step observation and action, Transformer enables the world model to predict transition conditioned on historical information.

- A world model conditioned on history consists of an observation encoder to capture abstract information

$$z_t \sim P_{\mathrm{enc}}\left(z_t \mid o_t\right) \tag{12}$$

where z_t represents the latent embedding of observation o_t, and and P_{enc} denote observation encoder

- A transition model to learn the transition in latent space

$$\hat{z}_{t+1}, \hat{r}_{t+1}, \hat{\gamma}_{t+1} \sim P_{\text{trans}}\left(\hat{z}_{t+1}, \hat{r}_{t+1}, \hat{\gamma}_{t+1} \mid z_{\leq t}, a_{\leq t}\right) \tag{13}$$

where P_{trans} denote transition model

Transformers for Sequential Decision-Making

Key idea

RL can be viewed as a conditional sequence modeling problem generating a sequence of actions that can yield high returns.

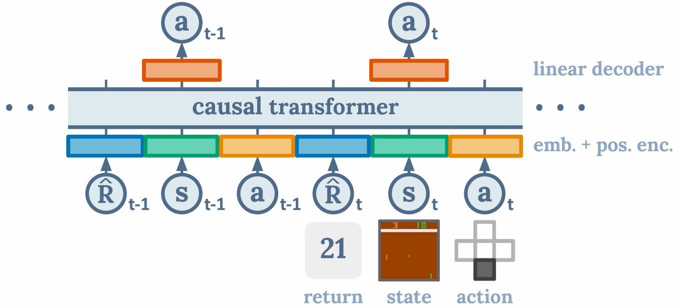
- Challenge: The non-stationarity during the training process
- Offline RL: Training a Transformer model on offline data
- Decision Transformer (DT) first applies this idea by modeling RL as an autoregressive generation problem to produce the desired trajectory:

$$\tau = \left(\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \ldots, \hat{R}_T, s_T, a_T\right) \tag{14}$$

where $\hat{R}_t = \sum_{t'=t}^{T} r\left(s_{t'}, a_{t'}\right)$ is the return-to-go. By conditioning on proper target return values at the first timestep, DT can generate desired actions without explicit TD learning or dynamic programming.

Decision Transformer: Reinforcement Learning via Sequence Modeling

Lili Chen, et al. UC Berkeley, Facebook AI Research, Google Brain, NeurIPS'21



Decision Transformer architecture

- States, actions, and returns are fed into modality-specific linear embeddings
- A positional episodic timestep encoding is added
- Tokens are fed into a GPT architecture
- Actions are predicted autoregressively using a causal self-attention mask

Method

Trajectory Representation

- Enabling transformers to learn meaningful patterns
- To generate actions based on future desired returns, rather than past rewards
- Feeding the model with the returns-to-go $\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right) \tag{15}$$

Architecture

- Feeding the last K timesteps with a total of $3K$ tokens (one for each modality: return-to-go, state, or action)
- A linear layer for each modality is used to obtain token embeddings
- For environments with visual inputs, the state is fed into a convolutional encoder instead of a linear layer

Method

Architecture

- An embedding for each timestep is learned and added to each token as one timestep corresponds to three tokens
- The tokens are then processed by a GPT model
- Predicting future action tokens via autoregressive modeling

Training

- Given a dataset of offline trajectories
- Sampling minibatches of sequence length K from the dataset
- The prediction head corresponding to the input token s_t is trained to predict a_t
- Cross-entropy loss for discrete actions
- Mean-squared error for continuous actions
- The losses for each timestep are averaged

Algorithm (Part 1)

Algorithm 1: Decision Transformer Pseudocode (for continuous actions) - Part 1

```
# R, s, a, t: returns-to-go, states, actions, and timesteps
# transformer: transformer with causal masking (GPT)
# embed_s, embed_a, embed_R: linear embedding layers
# embed_t: learned episode positional embedding
# pred_a: linear action prediction layer
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t) # per-timestep (note: not per-token)
    s_embedding = embed_s(s) + pos_embedding
    a_embedding = embed_a(a) + pos_embedding
    R_embedding = embed_R(R) + pos_embedding
    # interleave tokens as (R_1, s_1, a_1, ··· , R_K, s_K)
    input_embeds = stack(R_embedding , s_embedding , a_embedding)
    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)
    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions
    return pred_a(a_hidden) # predict action
```

Algorithm (Part 2)

Algorithm 2: Decision Transformer Pseudocode (for continuous actions) - Part 2

```
# training loop
for (R, s, a, t) in dataloader: # dims: (batch_size, K, dim)
    a_preds = DecisionTransformer(R, s, a, t)
    loss = mean((a_preds - a)**2) # L2 loss for continuous actions
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
# evaluation loop
target_return = 1 # for instance, expert-level return
R, s, a, t, done = [target_return], [env.reset()], [], [1], False
while not done:        # sample next action
    action = DecisionTransformer(R, s, a, t)[-1] # for cts actions
    new_s, r, done, _ = env.step(action)
    # append new tokens to sequence
    R = R+ [R[-1] - r] # decrement returns-to-go with reward
    s, a, t = s+ [new_s], a + [action], t + [len(R)]
    R, s, a, t = R[-K:], ... # only keep context length of K
```

Transformers for Generalist Agents

Key idea

To enable a generalist agent to solve multiple tasks or problems, as in the CV and NLP fields.

- On one hand, some research draws inspiration from pre-training methodologies applied in computer vision (CV) and natural language processing (NLP), attempting to ***abstract a general policy*** from large-scale multi-task datasets
- On the other hand, the effectiveness of prompting for adapting to new tasks has been demonstrated in numerous prior studies in NLP. Building on this concept, several works focus on ***harnessing prompting techniques for DT-based methods*** to facilitate rapid adaptation
- Beyond generalizing to multiple tasks, Transformer is also a powerful universal model to unify a range of domains related to sequential decision-making

Bridging Online and Offline Learning via Transformers

To train a well-performed online Decision Transformer

- Utilizing Transformers to capture dependencies in decision sequence and to abstract policy is mainly inseparable from the support of considerable offline demonstrations
- Not easy to obtain expert data in certain tasks
- While a substantial amount of less related data holds potential valuable information
- Some environments are open-ended (e.g., Minecraft)
- The policy has to continually adjust to deal with unseen tasks during online interaction
- Bridging online learning and offline learning via transferring the generalization capabilities

Combining Reinforcement Learning and (Self-) Supervised Learning

Unified training approach for RL and Transformer

- The training methods involve both RL and (self-) supervised learning
- When trained under the conventional RL framework, the Transformer architecture is usually unstable for optimization
- RL algorithms inherently require imposing various constraints to avoid the deadly triad issues, i.e., Bootstrapping, Function Approximation, Off-Policy-Learning
- Potentially exacerbate the training difficulty of the Transformer architecture
- The (self-) supervised learning paradigm, while improving the optimization of the Transformer structure
- Significantly constrains the performance of the policy based on the quality of the experience/demonstration data

Transformer Structure Tailored for Decision-making Problems

Enhancing network structures for decision-making problems

- The Transformer structures in the current DT-based methods are mainly vanilla Transformer
- Originally designed for the text sequence and may not fit the nature of decision-making problems
- Is it appropriate to adopt the vanilla self-attention mechanism for trajectory sequences?
- Whether different elements in the decision sequence or different parts of the same elements need to be distinguished in position embedding?
- Transformer is a structure with huge computational cost and high memory occupation

Towards More Generalist Agents with Transformers

Performing multiple and cross-domain tasks

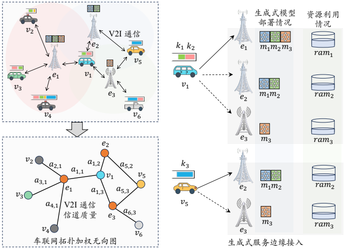
- Allow multiple modalities (e.g., image, video, text, and speech) to be processed
- Using similar processing blocks
- Demonstrating excellent scalability to large-capacity networks and huge datasets
- Generalizing to unseen tasks without strong assumptions
- A general world model

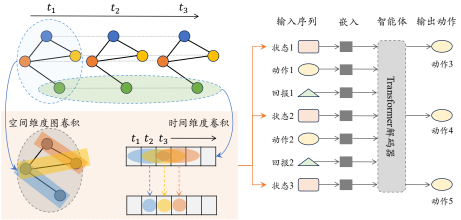
Connections to Other Research Trends

Using RL to benefit Transformer training

- Model language/dialogue generation tasks with offline RL setting and learn generation policy
- Reinforcement Learning from Human Feedback learns a reward model and uses RL algorithms to finetune Transformer for aligning language models with human intent

DT tailored to the Generative Model Edge Caching and Scheduling Problem





Improvement Direction

1. Single Agent $\rightarrow$ Multi-Agent
2. The improvement of Transformer Structure

Question

1. How to improve the data quality of the offline dataset?
2. The environment may not be too complex.

Q&A

Thanks!