Decision Transformer: Reinforcement Learning via Sequence Modeling

Ben Kabongo
Master MVA, ENS Paris-Saclay
ben.kabongo_buzangu@ens-paris-saclay.fr

Abstract

The paper presents a new framework that abstracts reinforcement learning (RL) as a sequence modeling problem. This takes advantage of the simplicity and scalability of the Transformer architecture and associated advances in language modeling.
Decision Transformer is an architecture that transforms the RL problem into conditional sequence modeling. Unlike previous RL approaches that fit value functions or compute policy gradients, Decision Transformer simply produces optimal actions by relying on a causally masked Transformer.
By conditioning an autoregressive model on desired return, past states and actions, Decision Transformer can generate future actions that achieve the desired return.
Decision Transformer meets or exceeds the performance of state-of-the-art model-free offline LR databases on Atari, OpenAI Gym and Key-to-Door tasks.

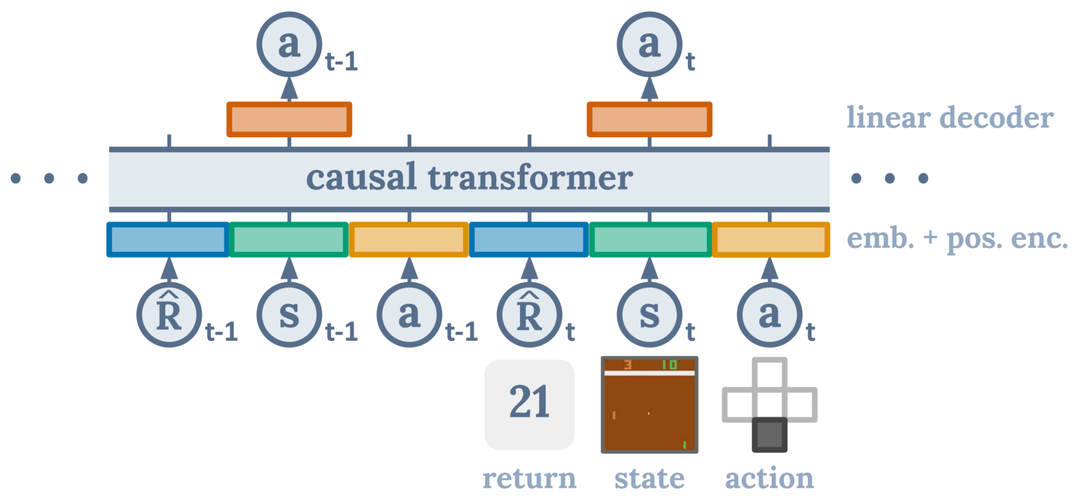

Figure 1: Decision Transformer architecture

Algorithm 1 Decision Transformer Pseudocode (for continuous actions)

```
# R, s, a, t: returns-to-go, states, actions, or 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

# main model
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

    # predict action
    return pred_a(a_hidden)

# 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:  # autoregressive generation/sampling
    # 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
```

Preliminaries

Offline reinforcement learning

A **Markov decision process** (MDP) is a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, with:

- $\mathcal{S}$ **state space**: set of states,
- $\mathcal{A}$ **action space**: set of actions,
- $\mathcal{P}$ **transition function**: deterministic $(\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S})$ or probabilistic $(\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1])$ function of the effects of the agent's actions on the environment.
- $\mathcal{R}$ **reward function**: reward perceived by the agent following actions performed and transitions made within the environment. We can have a deterministic reward function that depends on the current state s and the action performed a given by $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

A **policy** is a deterministic $(\pi : \mathcal{S} \to \mathcal{A})$ or probabilistic $(\pi : \mathcal{S} \times \mathcal{A} \to [0, 1])$ function that indicates the choice of actions for each state. A **trajectory** is a collection of sequences of states, actions and rewards over time: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T, a_T, r_T)$. The goal in reinforcement learning is to learn a policy which maximizes the expected return $\mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$ in an MDP. In offline reinforcement learning, instead of obtaining data through interactions with the environment, we only have access to a limited, fixed dataset of arbitrary policy trajectories.

Transformers

Transformers have become an essential architecture for efficiently modeling sequential data. These models consist of stacked self-attention layers with residual connections. Each self-attention layer receives n embeddings $x_{i\,i=1}^{n}$ corresponding to unique input tokens, and outputs n embeddings $z_{i\,i=1}^{n}$. The i-th token is mapped via linear transformations to a key k_i, query q_i, and value v_i. The i-th output of the self-attention layer is given by weighting the values v_j by the normalized dot product between the query q_i and other keys k_j: $z_j = \sum_{j=1}^{n} \text{softmax}(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n})_j \dot{v}_j$

Methods

Trajectory representation. $\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \cdots, \hat{R}_T, s_T, a_T)$. In the Decision Transformer model, for a given time t, the sum of future rewards (return-to-go) obtained is used instead of the immediate reward. $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. At test time, we can specify the desired performance (e.g. 1 for success or 0 for failure).
Architecture. The last K time steps are passed as input to the decision transformer, for a total of 3K tokens (one for each modality: return-to-go, state or action). To obtain token embeddings, we learn a linear layer for each modality, which projects the raw inputs into the embedding dimension, followed by layer normalization. The tokens are then processed by a GPT model, which predicts future action tokens through autoregressive modeling.

Experimental results

Figure 2: Results comparing Decision Transformer (DT) to TD learning (CQL) and behavior cloning across Atari, OpenAI Gym, and Minigrid. On a diverse set of tasks, Decision Transformer performs comparably or better than traditional approaches. Performance is measured by normalized episode return.

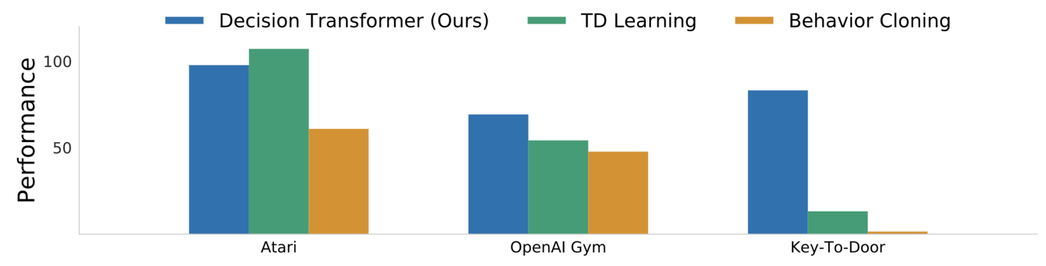

Table 1: Player-normalized scores for the Atari DQN replay 1% dataset. We show the mean and variance for 3 seeds. Best average scores are highlighted in bold. The decision transformer (DT) performs comparably to CQL in 3 out of 4 games, and outperforms the other baselines in most games.

Game	DT (Ours)	CQL	QR-DQN	REM	BC
Breakout	**267.5 ± 97.5**	211.1	17.1	8.9	138.9 ± 61.7
Qbert	15.4 ± 11.4	**104.2**	0.0	0.0	17.3 ± 14.7
Pong	106.1 ± 8.1	**111.9**	18.0	0.5	85.2 ± 20.0
Seaquest	**2.5 ± 0.4**	1.7	0.4	0.7	2.1 ± 0.3

Table 2: Table2: Results for D4RL datasets. We show the mean and variance for three seeds. Decision Transformer (DT) outperforms conventional RL algorithms on almost all tasks.

Dataset	Environment	DT (Ours)	10%BC	25%BC	40%BC	100%BC	CQL
Medium	HalfCheetah	42.6 ± 0.1	42.9	43.0	43.1	43.1	**44.4**
Medium	Hopper	**67.6 ± 1.0**	65.9	65.2	65.3	63.9	58.0
Medium	Walker	74.0 ± 1.4	78.8	**80.9**	78.8	77.3	79.2
Medium	Reacher	51.2 ± 3.4	51.0	48.9	58.2	**58.4**	26.0
Medium-Replay	HalfCheetah	36.6 ± 0.8	40.8	40.9	41.1	4.3	**46.2**
Medium-Replay	Hopper	**82.7 ± 7.0**	70.6	58.6	31.0	27.6	48.6
Medium-Replay	Walker	66.6 ± 3.0	**70.4**	67.8	67.2	36.9	26.7
Medium-Replay	Reacher	18.0 ± 2.4	**33.1**	16.2	10.7	5.4	19.0
Average		56.1	**56.7**	52.7	49.4	39.5	43.5

References

[1] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.