Sequential Decision Modelling using Structured State Spaces

Ana Vitoria Rodrigues Lima
Harvard University
anavitoria_rodrigueslima@
@fas.harvard.edu

Geoffrey Liu
Harvard University
geoffreyliu@fas.harvard.edu

Christopher Croft
Harvard University
ccroft@g.harvard.edu

Abstract

Credit assignment in Reinforcement Learning contends with the ability of an agent to determine the true source of returns from a trajectory of states and actions. This is known to be a challenging problem, especially as the length of an episode increases. A novel approach to Reinforcement Learning proposed and implemented by Chen et. al. in their work "Decision Transformer: Reinforcement Learning via Sequence Modelling" approaches this problem by using a transformer as a sequence model. By combining the power of the autoregressive transformer architecture with "Upside Down" RL, in which we condition on desired reward, Decision Transformers are capable of attaining state-of-the-art model-free performance on several common RL benchmarks. Given the success of this combination of ideas, attempted to extend this concept by retaining the "Upside-Down" RL approach, and replacing the sequence model with a promising new architecture, the Structured State Space Sequence Model (S4). This is a novel architecture which allows for efficiently modeling sequences much longer than standard transformers. Preliminary experiments from this project indicate that S4 is able to obtain higher rewards compared to Decision Transformers.

1 Introduction

The intention of this project is to explore the applicability of sequence models to the problem of credit assignment in Reinforcement Learning. A policy must be able to recognize which action(s) influenced future rewards if it is to learn an optimal sequence of actions for performing a task. There are several settings in which this turns out to be very challenging, specifically when rewards are sparse and when the environment is highly stochastic. Decision Transformers attempt to tackle this from a sequence modeling perspective. The idea is that the Transformer architecture is inherently capable of performing credit assignment through self-attention, which allows them to learn which actions were critical even amidst sparse or noisy rewards. Still, this work and others are limited by the information propagation capabilities of the underlying sequence models. Our model exchanges the transformer-based sequence model in the Decision Transformer framework with an sequence model that uses structured state spaces, which we call S4DM (S4 decision model). Our preliminary experiments indicate that S4DM achieves higher rewards and outperforms the Decision Transformer baseline on three MuJoCo environments, which indicates that S4 may be a better sequence model for this task.

2 Background

2.1 Decision Transformer

Decision Transformers are an abstraction of the Reinforcement Learning paradigm into a sequence modeling problem [1]. Intuitively this seems like a natural fit, as traditional RL policies can be "rolled out" within an environment to generate trajectories, which are simply sequential tuples of state, action, and reward. The idea here is that optimal future actions which maximize return can be generated by a causal transformer conditioned on a target reward, and past states and actions. Key here is both the novel application of sequential models as well as the idea of conditioning on desired reward. The intention is that instead of generating actions based on past rewards, the model will instead do so based on *desired* future returns. To incorporate this into the Decision Transformer framework, desired reward-to-go is calculated as $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. At test time, starting from a target return of ρ, we subtract the reward obtained in step t as $\rho' = \rho - r_t$ and feed in ρ' as the target reward-to-go in the next timestep. We can formalize a trajectory at train time, represented as a sequence as:

$$\tau = [\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, ..., \hat{R}_T, s_T, a_T]$$

Each of reward-to-go, states, and actions are projected into an embedding space by separate learned linear layers. A positional embedding is also learned to represent the current timestep, and this embedding is added to each token. In the published implementation, these tokens are then processed by GPT-2 to predict future actions auto-regressively. In the case of continuous actions, the loss is simply calculated as the mean squared error between prediction actions and actual actions:

$$\frac{1}{T}\sum_{t'=t}^{T}(\hat{a} - a)^2$$

Action predictions are retrieved from the transformer hidden states through a linear action prediction layer. This leads to the resulting algorithms which are presented using PyTorch-like pseudocode.

Algorithm 1 Decision Transformer

Input: R=reward, s=state, a=action, t=timestep

function DECISIONTRANSFORMER(R, s, a, t)
    $p \leftarrow \text{embed}_t(t)$
    $R \leftarrow \text{embed}_R(R) + p$
    $s \leftarrow \text{embed}_s(s) + p$
    $a \leftarrow \text{embed}_a(a) + p$
    $input \leftarrow \text{stack}(R, s, a)$
    $hidden\_states \leftarrow \text{GPT2}(input).\text{last\_hidden\_layer}$
    $a\_hidden \leftarrow \text{unstack}(hidden\_states).actions$
    return $pred\_a(a\_hidden)$
end function

Algorithm 2 Decision Transformer: training

for $(R, s, a, t) \in data$ **do**
    $optimizer.zero\_grad$
    $a\_preds = \text{DECISIONTRANSFORMER}(R, s, a, t)$
    $\text{loss} = \text{MSE}(a\_preds, a)$
    $loss.backward$
    $optimizer.step$
end for

In comparison to previous methods, Decision Transformer takes a different perspective on how to generate actions - both in terms of architecturally as well as conceptually. It is common in traditional

Algorithm 3 Decision Transformer: evaluation

Initialize: R_{target}, s, a, t, done
while not done **do**
  $next\_action \leftarrow$ DECISIONTRANSFORMER(R_{target}, s, a, t)
  $new\_obs, r, done \leftarrow env.step(next\_action)$
  $R \leftarrow R - r$
  $s, a, t \leftarrow [s] + [new\_obs], [a] + [action], t + 1$
  keep most recent K timesteps
end while

reinforcement learning to parameterize a policy, actor or critic as a sequential model (e.g. RNN, LSTM, GRU, etc...), which has a similar intention in mind, which is to enable a form of memory to aid in sequential decision-making. Though these approaches are not reimagining the very approach to performing RL as Decision Transformer does, they are simply making a decision as to how they learn within the existing framework. DT is explicitly taking a new approach to generating actions in a manner that directly leverages the strengths of the transformer architecture in conditional sequential modeling.

2.1.1 Transformers

The Transformer model was introduced in the landmark paper "Attention Is All You Need" by Vaswani et. al. in 2017 [7]. Whereas previous sequential models relied on recurrences and/or convolutions, the Transformer did away with all of that in favor of attention. Here, an encoder maps a sequence of tokens to a continuous sequence of representations **z**, and given **z**, a decoder generates an output sequence of symbols. The model is auto-regressive, and so at each timestep, it uses the previously generated symbols in generating the next. A key component of this architecture is the concept of stacked attention. An attention function is essentially a mapping from a query and a set of key-value pairs to some output, the purpose of which is to capture the relationship between components of the input. In the context of language modeling, this works well, as it can explicitly determine how words in an input sentence relate to each other, which is critical in understanding language. This same mechanism is why Transformers are a natural place to explore for applications in RL, as we are interested in how actions and rewards relate to each other, specifically in the context of the credit assignment problem.

2.1.2 "Upside Down" Reinforcement Learning [5]

In traditional reinforcement learning, we learn to predict the discounted future rewards, given previous actions and states, and then learn to transform the predicted rewards into actions through a policy. Upside Down RL (UDRL) skips the intermediary step of predicting a reward, rather it takes a reward as an input. UDRL takes the state, *desired reward*, and *desired time horizon* to achieve the reward as the input, and learns to output an action. This changes the goal of the learning from maximizing returns in expectation, to following *commands* of desired rewards and time horizons. Simply, the UDRL machine takes in commands in the form of "*get so much reward within so much time*", and learns an action consistent with that command.

The innovation in UDRL is turning the problem into a supervised learning problem. The supervised learning problem takes three inputs: *state, desired reward, desired time horizon*, and outputs an action. This leads to two key properties in UDRL. First, the cumulative reward is an input to the agent rather than a prediction as in value-based RL. Second, the learning is based on optimizing a true supervised learning objective, as compared to traditional RL algorithms where the targets are non-stationary.

Decision Transformer's uses a very similar concept to UDRL with two minor changes. First, DT takes in previous states and actions which changes the reinforcement learning problem into a sequence prediction problem i.e. given a history of states and actions, and a reward command, what is the best action. Second, the time-horizons are fixed to be length 1.

2.2 Structured State Space Sequence Model (S4)

[3] tackles the known problem of Long Range dependencies, LRDs. Previous conventional models such as RNNs, CNNs and Transformers have specific variants for capturing long dependencies, however they still struggle with 10 000 or more steps being that they struggle from vanishing gradients. This paper suggests a new method, the S4, which is based on state space models (SSMs), a concept borrowed from the field of control theory:

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) \end{aligned} \tag{1}$$

Given a known and accepted benchmark, the Long Range Arena LRA benchmark [6], S4 is as fast as several baselines and also outperforms those based on the LRA benchmark.

In [3] they put in place a Linear State Space Layer (LSSL [4]) that gets the benefits of RNN models and the benefits of how state space models (SSMs) can address the problem of long range dependencies. Since LSSL has prohibitive computational requirements, they introduce a modification to the concept of the state space model: the structured state space, S4, [3]. The structured state space lies on the fact that they modify the matrices $\mathbf{A}$ by decomposing them as the sum of a low-rang and skew-symmetric term - specifically, they equip S4 with a particular modification of the state matrix $\mathbf{A}$ based on the HiPPO kernel [2]. Thanks to this, S4 is fast as several baselines and also outperforms those based on the LRA benchmark.

The HiPPO kernel to used to define the matrix $\mathbf{A}$ from (1), they discretize this in order to get a sequence to sequence mapping - this enables the state equation to be a recurrence in x_t, enabling the discretized SSM to be computed as a typical RNN. Ultimately, the architecture resulting from these is a layer of a deep neural network, the S4 layer. The core of the S4 layer is the HiPPO kernel which captures a structured state space, along with a feed forward network that consists of nonlinear activations and a linear transformation. What this S4 layer defines is a sequence-to-sequence mapping, with the familiar shape of (batch size, sequence length, hidden dimensions) - similar to sequence models like Transformers and RNNs.

3 Structured State Space Sequential Decision Model (S4DM)

The aim of our project is to implement Decision Transformer [1] by replacing the usage of a model that uses transformer self attention layers in Algorithm 1 with an architecture that uses S4 layers.

Given that [3] shows promising results, such as S4 substantially closing the gap to Transformers for both generative tasks and classification tasks, we plan to use this architecture in our RL context with the hypothesis that S4 will handle longer ranges of trajectories. Meaning, we expect that this new architecture use ‘remember’ longer parts of the input trajectories while generating a new sequence of actions, thanks to the usage of the HiPPO kernel which compresses history within the model. The increased "memory" of this model should give it a distinct advantage in assigning credit to critical actions in a trajectory, allowing it to achieve better performance.

4 Experiments

4.1 Environments

In total, we aim to benchmark results in four environments. The first three environments, HalfCheetah, Walker2D and Hopper are coming from the OpenAI MuJoCo platform. The last and fourth environment is a directed graph, whose dataset will be a random walk - this has been done by [1] and we are trying to replicate this as well.

Our experiments will use three MuJoCo environments, HalfCheetah, Walker2D and Hopper, which is a simple simulated robot designed to mimic motor tasks for humans and animals. The HalfCheetah environment consists of a two-dimensional two-legged figure that goal is to run forwards and has 17

dimensional state space and an action dimension of 6. The Walker2D environment has 17 dimensional state space and action dimension of 6, with a goal of walking forwards without falling. The Hopper environment has an 11 dimensional state space and a 3 dimensional action space, with a goal of hopping forward without falling. [1] All these environments have an unbounded state space, however the each dimension of the action space is bounded by $[-1, 1]$ and represent the torque applied to the joints of the robot.

Since we are learning off an offline environment, we use a dataset of trajectories generated by different agents. We use a dataset of trajectories from sub-optimal agents, we refer to these datasets as,

- Medium: 1 million timesteps generated by a "medium" policy that achieves approximately one-third the score of an expert policy.
- Medium-Replay: the replay buffer of an agent trained to the performance of a medium policy (approximately 25k-400k timesteps in our environments).

In total we have six datasets from the combination consisting of medium and medium-replay trajectories for three environments.

4.2 Experiment details

The experiments we implemented have been to run are to implement 'our new' S4DM for Medium and Medium-replay trajectories in the three aforementioned environments.

Due to the stochasticity that a deep learning architecture from its initialization, we run each experiment three times with separate seeds and report the mean standard error of the evaluations. For each agent, we train on 1000 trajectories per iteration, and evaluate on ten (10) episodes after each iteration to assess the agent's performance in that environment. We compare results between the S4DM model, with both a low and high target reward, against the baseline Decision Transformer and a Behavioral Cloning model in our results. Recall that the Decision Transformer framework utilizes Upside-Down RL, and so we evaluate performance against a "high" target reward, as well as a "low" target (equal to half the high target) to compare how target returns affect performance.

4.3 Implementation details

To implement S4DM in the most similar way to baseline DT model, by modifying line 7 of Algorithm 1. Instead of using GPT-2 as the sequence modelling neural network architecture, we replace this with an $S4$ sequence model. Each S4 layer within the S4 model consists of a kernel (i.e. the HiPPO Kernel) and a feed-forward layer (i.e. linear layer plus activation and dropout) with a skip connection. The kernel block can viewed similarly to the masked self-attention block in the transformer decoder within GPT-2. The inputs to the S4 model is exactly the same as the inputs to GPT-2 as in Algorithm 1. The last hidden layer of the S4 model is taken to be output after passing through all the S4 layers.[2].

The S4 model has many hyperparameters choices, which include the dimension of the kernel state space [3], the maximum sequence length, number of channels, whether the kernel is bi-directional or unidirectional, the activation function after the kernel outputs, and the dropout. Similar to the number of decoder blocks within GPT-2, another choice in the S4 Model is the number of S4 layers.

The dimension of the kernel state space, maximum sequence length control, and the number of channels control the dimensions of the kernel within the S4 block. Some choices of these hyperparameters were clear, such as setting the maximum sequence length to be 1024, larger than the maximum sequence length in our data, and the number of channels is set to be 1 as the (state, action, reward) tuple inputs in DECISIONTRANSFORMER are stacked and flattened into a 1-D vector in line

[1] https://www.gymlibrary.ml/environments/mujoco/hopper/

[2] Note: in a regular S4 model, the last fully connected layer is trained, however in decision transformer the outputs of the last hidden layer for both S4 and GPT2 are used directly

[3] Note: not to be confused with the environment state space

6 of 1. The dropout rate is set to be equal to 0.2 (default), however we note that this is a tuneable hyperparameter.

The kernel being unidirectional is critical to the implementation, as we do not want information from future states and actions to leak into the sequence model. The activation function was chosen to be GELU (Gaussian error Linear Unit) activation which is standard for GPT-2 and [3] original implementation of S4 on other tasks. The other hyperparameters, such as the size of the kernel state space, number of S4 layers are selected via grid search, detailed in the next section.

5 Results

5.1 S4 Sensitivity to Hyperparameters tuning

Since S4 is a new model, it is hard to know the optimal hyper-parameter settings to use for this task. Therefore, we run a small grid search over the learning rate, the size of the kernel state space, and number of S4 layers to see which combination provides the highest maximum performance and the highest performance at the final training iteration. A table of results can be found in Table 3

From this, we decide on using as environment Walker2D Medium Replay because this is the most difficult environment to perform well in our experiments. It is interesting to note that on average utilizing the higher number of S4 layers and of kernel dimension, the higher the maximum achievable reward. The best hyperparameters we found were a learning rate of 6.1×10^{-4}, a S4 kernel size of 256 dimensions, and using 16 S4 layers in our S4DM model.

We do not believe this is is an exhaustive experiment, however due to limited compute power we could only spend limited resources searching for optimal hyperparameters, as we have run this grid search for about 24 hours.

5.2 Comparison of performance of S4 vs DT

Dataset	Environment	Target Max	Target Mean	S4DM results	DT results	BC results
Medium	Half-Cheetah	5309.38	4770.33	**5134.56**	4961.148	4437.66
Medium	Hopper	3222.36	1422.06	**2438.59**	1707.403	1893.73
Medium	Walker2D	4226.94	2852.09	**3787.78**	3249.886	3355.53
Medium-Replay	Half-Cheetah	4985.14	3093.29	**4654.11**	3873.734	-0.95
Medium-Replay	Hopper	3192.93	467.30	**2686.72**	1460.075	1039.16
Medium-Replay	Walker2D	4132.00	682.70	**3450.24**	2402.301	1010.17

Table 1: Our S4DM model outperforms both Decision Transformer and Behavior cloning across all environments and datasets with respect the the maximum reward obtained across iterations. All these experiments have been run for a total of 3 times to ensure accuracy of results. These results are the average of these three runs per iterations, and the max of these averages. The Target Max and Mean is the maximum and mean rewards across the trajectories in the dataset

Dataset	Environment	S4DM	DT	BC
Medium	Half-Cheetah	**48.2%**	46.6%	41.6%
Medium	Hopper	**69.3%**	48.3%	53.7%
Medium	Walker2D	**77.0%**	66.1%	68.2%
Medium-Replay	Half-Cheetah	**43.7%**	36.3%	-0.01%
Medium-Replay	Hopper	**76.4%**	41.2%	29.2%
Medium-Replay	Walker2D	**70.1%**	48.8%	20.5%

Table 2: Comparison of S4DM, DT and BC using normalized scores. Metric calculated as $\%\text{expert} = 100 \cdot \frac{\text{model return - random model return}}{\text{expert return - random model return}}$. The expert returns can be found in Table 4

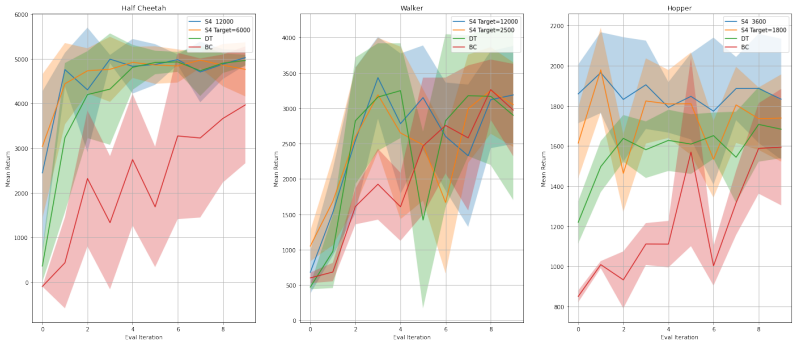

Figure 1: Mean returns for S4 (high and low targets), Decision Transformer and Behavior Cloning trained on medium trajectories.

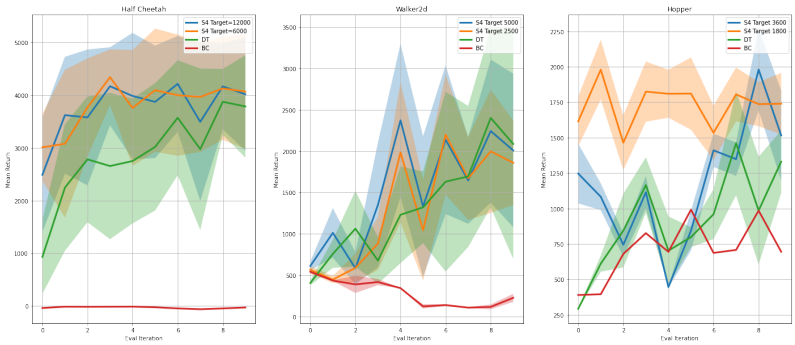

Figure 2: Mean returns for S4 (high and low targets), Decision Transformer and Behavior Cloning trained on "medium" agent replay buffers

In Figure 1 and Figure 2 we immediately observe the high variance in return from S4 and Decision Transformer. Both methods comfortably outperform Behavior Cloning, but when considering the variance, it is hard to definitely determine whether the S4 model is doing better than Decision Transformer. We suspect that the high variance is due to evaluating on so few episodes (10). The decision to not perform more evaluation was due to time and resource constraints. What we are able to determine from these plots is that, at least preliminarily, S4 performs at least as well as DT in these environments. Combining the slight edge we can see in these plots with the greater maximum obtained reward over the all evaluations, we have reason to believe this is an interesting alteration to the DT framework that is worth exploring further.

We believe our results do not fully characterize S4's performance as a decision model. In our experiments, we specifically compared S4 to the environments that the original DT model was trained on. These environments don't play into the strengths of S4DM as the trajectories on the MuJoCo were relatively short and can be easily handled by GPT-2. It would be interesting to compare S4DM and DT on an environment with very long trajectories with lengths greater than 10,000.

6 Conclusion

In this project we explore the applications of sequence models to the problem of credit assignment. Specifically, we explore and implement the Decision Transformer which uses GPT-2 as a sequence model in combination with Upside-Down reinforcement learning. We replace the GPT-2 sequence model with the S4 sequence model to create a new model, S4DM. We then compare the performance of our model to the Decision Transformer and show that we are able to outperform DT across all environments and datasets with respect with the maximum reward obtained across iterations.

For future work, it would important to compare the Decision Transformer and the S4DM model on a wider set of environments, especially environments with long trajectories that Decision Transformer will struggle. Moreover, we do not believe we have selected the optimal hyper-parameters for the S4DM model and further tuning can yield better results.

7 Contributions

Geoffrey Liu

- Implemented the S4 Model that replaced the GPT-2 model in Algorithm 1 and debugged the Cauchy kernel CUDA implementation to train our model faster.
- Helped to incorporate the S4 Model into the Decision Transformer Training Loop and set up the S4DM codebase.
- Debugged the initial training run of the S4 and identified the correct S4 parameters for S4DM to work, such as specifying the unidirectional kernel the S4 layer.
- Helped to get the initial baseline Decision Transformer and Behavior cloning experiments using the authors code.

Chris Croft

- Implemented the Decision Transformer framework that we swapped S4 into
- Helped to incorporate the S4 Model into the Decision Transformer Training Loop
- Debugged training run of S4
- Hyperparameter tuning grid search
- Helped run S4DM experiments
- Downloaded and stored the training datasets for all environments

Ana Vitoria Rodrigues Lima

- Replicated S4 Mnist and Cifar paper results for the first Milestone
- Assisted in the process of incorporating S4 into Decision transformer
- Helped run experiments for S4 and DT and BC. Replicated DT for all environments and datasets over multiple (3) runs
- Contributed in calculation of table values across different runs
- Implemented an efficient way to store the logs of our experiments across environments and models so that not to confuse experiments and runs with each other
- Contributed for a broader understanding of S4 within the group

References

[1] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[2] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Re. Hippo: Recurrent memory with optimal polynomial projections, 2020.

[3] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[4] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state-space layers, 2021.

[5] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards – just map them to actions, 2019.

[6] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers, 2020.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

LR	kernel dim	layers	Max	Last iteration
6.1e-4	256	4	3067.4232	1819.75265
6.1e-4	256	8	3191.243218	2191.465487
6.1e-4	**256**	**16**	**3489.1581625**	**3259.45171**
6.1e-4	128	4	3231.5531	2123.19534
6.1e-4	128	8	3218.0228126	3218.022812
6.1e-4	128	16	3225.85721	1697.52307
9.1e-4	128	4	3000.010026	3000.0100
1e-5	256	4	564.26030	541.91239
1e-5	256	8	967.9459304	247.500470
1e-5	256	16	733.88213	214.218650
1e-5	128	4	598.7867860	563.09206
1e-5	128	8	580.8788	522.0362305
1e-5	128	16	463.10662548	420.4205075
3.1e-4	256	4	3654.690542	2346.19014
3.1e-4	256	8	3250.94064	2794.95862
3.1e-4	256	16	2935.1227	1895.41712
3.1e-4	128	4	2554.7521	1419.6335
3.1e-4	128	8	3205.9525	3083.82110
3.1e-4	128	16	3522.693079	1526.34002

Table 3: Comparison of S4DM performance using different hyperparmeters on Walker2D medium-replay trajectories. The 'Max' column is the maximumm reward over all iterations, and the 'Last iteration' column is the reward achieved at the last iteration of the training.

Agent	Environment	Expert
Expert	Half-Cheetah	10656
Expert	Hopper	3511
Expert	Walker2D	4920

Table 4: Rewards from an expert agent on the three environments

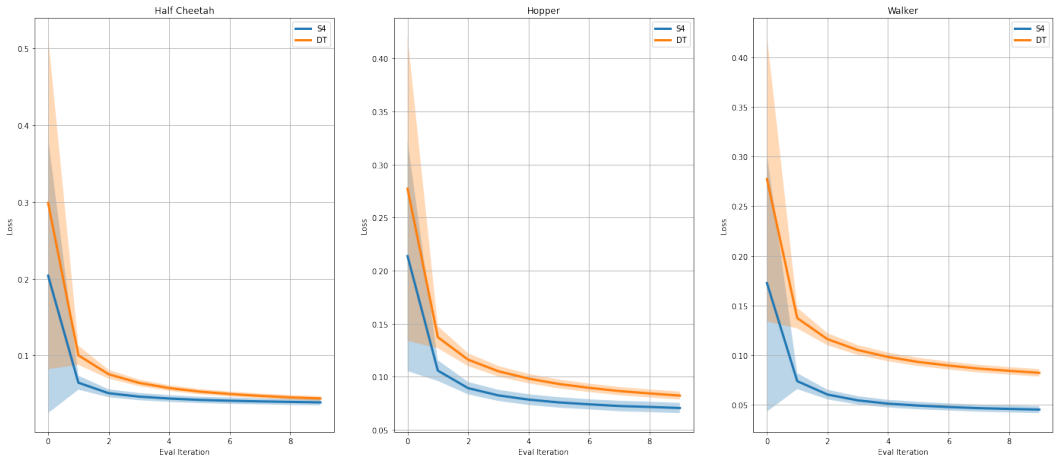

Figure 3: Training loss for S4 and Decision Transformer trained on medium trajectories.

.1 Details of the HiPPO Kernel

[2] frames the problem of memory in LRD as a problem of *online function approximation*. HiPPO, i.e. high-order polynomial projection operators, can be introduced into an RNN and can improve

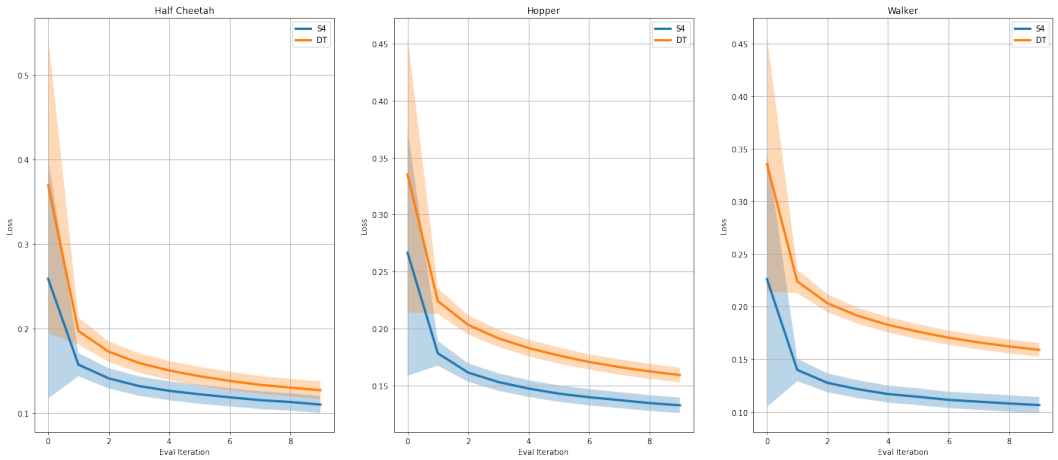

Figure 4: Training loss for S4 and Decision Transformer trained on "medium" agent replay buffers

long-term dependencies. To solve the memory problematic, they phrase a way to learn memory representation through online function approximation with projections. Essentially, since cumulative history cannot be memorized, they compress it and have as objective to maintain this compressed representation of history within a given model. In simpler terms, the combination of a *projection*, which takes a function f in a time range t and maps it to a polynomial g, and of a *coefficient*, that maps that polynomial g to coefficients c of the basis of orthogonal polynomials defined with respect to to a third measure μ, is the HiPPO framework. As formulated by Gu et al: for a function f at every time t there is an optimal projection $g^{(t)}$ of f onto the space of polynomials with respect to a measure $\mu^{(t)}$ weighting the past. For an appropriately chosen basis, those coefficients $c(t)$ represent a compression of the history of f, and satisfies linear dynamics. By discretizing the dynamics, there is an efficient closed-form recurrence, for an online compression of time series $(f_k)_{k \in N}$.