Wall Street Tree Search: Risk-Aware Planning for Offline Reinforcement Learning

Dan Elbaz
dan.elbaz@intel.com

Gal Novik
gal.novik@intel.com

Oren Salzman
osalzman@cs.technion.ac.il

Abstract

Offline reinforcement-learning (RL) algorithms learn to make decisions using a given, fixed training dataset without the possibility of additional online data collection. This problem setting is captivating because it holds the promise of utilizing previously collected datasets without any costly or risky interaction with the environment. However, this promise also bears the drawback of this setting. The restricted dataset induces subjective uncertainty because the agent can encounter unfamiliar sequences of states and actions that the training data did not cover. Moreover, inherent system stochasticity further increases uncertainty and aggravates the offline RL problem, preventing the agent from learning an optimal policy. To mitigate the destructive uncertainty effects, we need to balance the aspiration to take reward-maximizing actions with the incurred risk due to incorrect ones. In financial economics, modern portfolio theory (MPT) is a method that risk-averse investors can use to construct diversified portfolios that maximize their returns without unacceptable levels of risk. We integrate MPT into the agent's decision-making process to present a simple-yet-highly-effective risk-aware planning algorithm for offline RL. Our algorithm allows us to systematically account for the *estimated quality* of specific actions and their *estimated risk* due to the uncertainty. We show that our approach can be coupled with the Transformer architecture to yield a state-of-the-art planner for offline RL tasks, maximizing the return while significantly reducing the variance.

1 Introduction and Related Work

RL is concerned with an agent learning how to take actions in an environment to maximize the total reward it obtains. The RL agent typically learns by trial and error, involving online interaction with the environment to collect experiences (Sutton and Barto, 1998). But learning in the real world may be undesirable, as online data acquisition is often costly, time-consuming, or even dangerous. Offline RL aims to bridge the gap between RL algorithms and real-world systems by leveraging an existing dataset or batch to learn how to make decisions in an offline stage without any online interactions with the environment (Levine et al., 2020).

However, learning solely from offline data is a double-edged sword. On the one hand, it enables applications in domains where online exploration is avoided, e.g., in healthcare (Gottesman et al., 2018; Wang et al., 2018; Yu et al., 2019), autonomous driving (Sallab et al., 2017), and recommendation systems (Strehl et al., 2010; Covington et al., 2016). On the other hand, it poses a major algorithmic challenge as we are faced with high levels of uncertainty.

Foundation Models for Decision Making Workshop at Neural Information Processing Systems, 2022.

The offline RL setting often exposes the agent to two types of uncertainty, objective and subjective. The *objective uncertainty* emerges from the stochastic characteristic of an environment. This uncertainty is manifested in the variance of the observed data. However, this is not the only source of uncertainty. In offline RL, we typically want the learned policy to perform better than the policy that collected the data. Consequently, we must execute a different sequence of actions from the sequences stored in the batch. As a result, the agent encounters unfamiliar state-action sequences, which induces *subjective uncertainty* (uncertainty that comes from ignorance due to the limited size of the training batch). This uncertainty can lead to erroneous value estimations, a phenomenon known as *distributional shift* (Kumar et al., 2020), which is one of the central challenges of offline RL.

Algorithms for offline RL typically fall under two categories: model-free and model-based algorithms. Generally, each category addresses the uncertainty (mainly due to the distributional shift) differently. In *model-free* algorithms, the agent learns a policy or value function directly from the dataset. Such algorithms typically address the distribution shift by constraining the learned policy to avoid out-of-distribution behavior that the dataset does not support (Fujimoto et al., 2019; Kumar et al., 2019; Siegel et al., 2020; Wu et al., 2019; Peng et al., 2019) or use uncertainty quantification techniques, such as ensembles, to stabilize Q-functions (Agarwal et al., 2019; Fujimoto et al., 2019; Jaques et al., 2019; Wu et al., 2019).

The second category, *model-based RL* (MBRL) methods, is the approach we take in this work, which is less explored for offline RL. Nonetheless, prior works have demonstrated promising results of MBRL methods, particularly in offline RL (Kidambi et al., 2020; Yu et al., 2020; Janner et al., 2021; Chen et al., 2021). MBRL divides the RL problem into two stages: The first stage is learning an approximate environment model (referred to as the transition or dynamics model) with the data. In the second stage, this model is used for decision making (i.e., planning or policy search), usually via Model-Predictive Control (MPC) (Richalet et al., 1978). An advantage of using MBRL is that we can benefit from a convenient and powerful supervised learning workhorse in the model-learning stage, which allows us to generalize to states outside the support of the batch. However, due to the distribution shift, the model becomes increasingly inaccurate as we get further from the points in the batch. As a result, a planner that tries to obtain the highest possible expected reward under the model without any precautions against model inaccuracy can result in "model-exploitation" (Levine et al., 2020)- a situation where the planner prefers predictions with higher returns than would be obtained from the actual environment, resulting in poor performance. Argenson and Dulac-Arnold (2020) addressed the model-exploitation problem by using ensembles and averaging the reward over all ensemble members. Kidambi et al. (2020) addressed this problem by incorporating pessimism into learned dynamics models. MOPO Yu et al. (2020) also study the effects of uncertainty in model-based approach to offline RL and suggests optimizing a policy using an uncertainty-penalized reward. In contrast, we propose an explicit planning algorithm to account for the uncertainty.

Our approach to offline RL builds on the recent works by (Janner et al., 2021; Chen et al., 2021) that frame RL as a sequence modeling problem and exploit the toolbox of contemporary sequence modeling. At the heart of their approach is a sequence model based on the GPT-3 Transformer decoder architecture (Brown et al., 2020), which they use for learning a distribution over trajectories by jointly modeling the states and actions to provide a bias toward generating in-distribution actions. To plan using the Transformer model (decode its outputs), Janner et al. coupled it with a minimally modified version of beam search (BS) (Lowerre and Reddy, 1976), and replaced the log probabilities of transitions with the expectation of the predicted reward signal to create a search strategy for reward-maximizing behavior. However, decoding methods accounting only for the trajectories with the maximum predicted rewards and ignoring the uncertainty may suffice for large-scale language models, which are trained on millions of high-quality web pages (hence their decoding methods are not designed to be immune to the effects of objective and subjective uncertainty prevalent in offline RL) but might not be the best option in an offline RL setting. We incorporate uncertainty into sequential decision-making to address this issue and propose a new decoding algorithm.

In portfolio optimization theory, investors demand a reward for bearing risk when making investment decisions. Economist Harry Markowitz introduced this risk-expected return relationship in the mean-variance model in a 1952 essay (Markowitz, 1952), for which he was later awarded a Nobel Memorial Prize in Economic Sciences. The mean-variance model weighs the risk against the reward and solves the so-called portfolio-selection problem (formally defined in Sec 3.3), i.e., decides how much wealth to invest in each asset by considering only the expected return and risk, expressed as the variance.

Inspired by modern portfolio theory (MPT), we propose a new risk-aware planning algorithm for offline RL based on the mean-variance model to mitigate the distributional shift problem. As a risk model, we use the Transformer architecture to model a distribution over candidate trajectories and their value estimates. The variance of these value estimates acts as an indicator of the variance of the observed data and as a proxy for the risk indicating a distribution shift. Our planning algorithm is s best-first search algorithm that treats the value associated with each candidate trajectory as an asset. When building the search tree, we treat the limited node-expansion budget as wealth to invest in many different assets (each asset corresponds to a candidate trajectory) and solve a portfolio-optimization problem to determine how much wealth to invest in each of these assets. We then decide on the order of the best solutions (which trajectories to keep) based on the portfolio-optimization problem's solution by randomly sampling trajectories in proportion to their wealth allocation.

We refer to our planner, which we formally introduce in sec 4, as Wall Street Tree Search (WSTS) as an homage to the famous Monte Carlo Tree Search (MCTS) Kocsis and Szepesvári (2006) algorithm and evaluate it in continuous control tasks from the widely studied D4RL offline RL benchmark (Fu et al., 2020). We show, in sec 5, that WSTS matches or exceeds the performance of state-of-the-art (SOTA) offline RL algorithms. At the same time, it is substantially more reliable than the conventional decoding method resulting in significantly reduced variance when used on the same risk model.

2 Problem Definition

A *Markov decision process* (MDP) is a tuple $\mathcal{M} = \{S, A, r, P, \rho_0, \gamma\}$ where S and A are the sets of states and actions, $r : S \times A \to \mathbb{R}$ is the reward function, P is the system's dynamics , which is a conditional probability distribution of the form $P(s_{t+1}|s_t, a_t)$, representing the probability over the next state given the current state and the applied action, ρ_0 defines the initial state distribution and $\gamma \in (0, 1]$ is a scalar discount factor that penalizes future rewards.

A *trajectory* $\upsilon = (s_1, a_1, r_1, s_2, a_2, r_2, ..., s_H, a_H, r_H)$ is a sequence of H states, actions and rewards. Here both the reward and the next state of a given state-action pair adhere to some underlying MDP. A *policy* $\pi : S \to A$ is a mapping between states and actions.

In the *offline RL* problem we are given a static, previously-collected dataset $\mathcal{D}$ of trajectories collected from some (potentially unknown) behavior policy π_B (for example, π_B can be human demonstrations, random robot explorations, or both). The goal is to use $\mathcal{D}$ to learn, in an offline phase, a policy π for the underlying, unknown MDP $\mathcal{M}$, to be executed in an online phase.

3 Algorithmic Background

We take a *model-based* offline RL approach in this work. Such an approach divides the problem into two stages **(S1)** learning an MDP model $\mathcal{M}'$ that approximates $\mathcal{M}$ using the dataset $\mathcal{D}$ and **(S2)** using the learned model $\mathcal{M}'$ to extract the policy π. We will refer to the first and second stage as the *offline model learning* and the *online decoding* stages, respectively and to the entire approach as *model-based RL*. It is important to note that we use the same Transformer architecture proposed by Janner et al. (2021); Chen et al. (2021) to learn an MDP model **(S1)**. Our work focuses on the decoding procedure **(S2)**.

3.1 Model Learning (S1)

As stated, our approach to offline RL builds heavily on recent Model-based offline RL methods (Janner et al., 2021; Chen et al., 2021) using Transformers Vaswani et al. (2017) to learn an MDP model **(S1)**. We describe this approach in this subsection.

3.1.1 Dataset

Each transition in the original trajectory υ is augmented with a discounted reward-to-go estimate (namely, $R_t = \sum_{t'=t}^{H} \gamma^{t'-t} r'_t$) to obtain an *augmented trajectory* $\mathbf{y}$ consisting of a sequence of H states, actions, rewards, and reward-to-go. Here, it is important to note that the estimated reward-to-go is computed using the training data and estimates the return obtained by following the behavior policy π_B. In general, it does not necessarily approximate the values of the learned policy. However, since we use it as a heuristic estimation, it suffices for guiding the search in the decoding stage (Janner et al., 2021).

Using a discrete-token architecture (a Transformer) forces tokenization of the augmented trajectory. This is done by discretizing each dimension independently in the event of continuous inputs. Assuming N-dimensional states, M-dimensional actions, and scalar reward and reward-to-go, $\mathbf{y}$ is turned into a sequence of length $T = H \cdot (N + M + 2)$, where every dimension of the trajectory is a token (subscripts on all tokens denote timestep, and superscripts on states and actions denote dimension.):

$$\mathbf{y} = (\ldots, s_t^1, s_t^2, \ldots, s_t^N, a_t^1, a_t^2, \ldots, a_t^M, r_t, R_t, s_{t+1}^1 \ldots)$$

3.1.2 Notation

The elements of the vector $\mathbf{y}$ (a trajectory) are denoted using an ordered set of components, y_t , $\mathbf{y} = (y_1, y_2, \ldots, y_T)$. Each y_t is an element of $\mathcal{V}$, the set of output tokens. $\mathbf{y}_{<t}$ denotes a trajectory from the first time step up to $t - 1$, $\mathcal{Y}$ is the set of all valid output trajectories, x is the initial state, sampled from ρ_0 (the initial state distribution of the MDP $\mathcal{M}$). Finally, $\circ$ denotes a concatenation operator.

3.1.3 Model

A *Transformer* Vaswani et al. (2017) is a sequence-transduction model whose network architecture relies solely on attention mechanisms Bahdanau et al. (2015) which allows for high parallelization and was shown to be highly effective in natural language processing (NLP) tasks Devlin et al. (2019); Brown et al. (2020). We aim to use the Transformer to select an action sequence to be executed in the environment.

The model (parametrized via the network's weights θ) predicts the probability distribution over an output space of possible trajectories $\mathbf{y}$ given an initial state x, which is factorized as:

$$P_\theta(\mathbf{y}|x) = \prod_{t=1}^{T} P_\theta(y_t|\mathbf{y}_{<t}, x)$$

Where $P_\theta(y_t|\mathbf{y}_{<t}, x)$ typically defines a multinomial classification model. In such case, assuming a per-dimension vocabulary size of $\mathcal{V}$, the network's output layer consists of logits over a vocabulary of size $\mathcal{V}$. Namely, for a discrete output variable y_t (which represents a dimension of a state, action, reward, or reward-to-go estimate), the outputs correspond to a mapping: $y_t^i \mapsto p_i$ for $i \in \{1, \ldots \mathcal{V}\}$ where p_i is the probability of the next token to be y_t^i.

As we use the original network as-is, we refer the reader to the original paper for the architecture's details.

3.2 Online Decoding (S2)

To decode a sequence from a trained model, we use the model's autoregressive nature to predict a single token, y_t, at each step. Given an initial state x, we generate tokens sequentially using some heuristic-based algorithms. The core algorithm providing the foundation of our planning techniques is beam search (BS) Lowerre and Reddy (1976). Here we describe BS as a meta-algorithm. As we will see in Sec 4, this will significantly simplify the presentation of our planning algorithm WSTS. Accordingly we present in Alg. 1 BS as an algorithm that takes in additional two functions as inputs- a scoring function `score` and a filter function `filter`, which are called only after all candidate trajectories are collected. At each iteration, BS expands all currently-considered sequences and creates new candidate sequences. These candidate sequences are evaluated according to the `score` function. Subsequently, BS uses these scores as inputs to the `filter` function and decides which $\mathcal{B}$ sequences to keep for the next iteration.

Different instantiations of these functions correspond to different search algorithms that may be used during online decoding. For example, in NLP and imitation learning, the overarching objective is to find the most-probable sequence under the model at inference times (commonly known as maximum a posteriori, or MAP, decoding Meister et al. (2020)). This corresponds to solving the following optimization problem:

$$y^* = \arg\max_{\mathbf{y} \in \mathcal{Y}} \log P_\theta(\mathbf{y}|x). \tag{1}$$

Algorithm 1 Beam Search

Input: start state x, scoring function `score()`, filter function `filter()`, sequence model $P_\theta(y_t|\mathbf{y}_{<t}, x)$
Parameters: beam width $\mathcal{B}$, planning horizon H
Output: Approx. single best trajectory.

```
C_0 ← {x}
for t ← 1 to H do
  C ← {}
  for all y_<t ∈ C_{t−1} do
    // Autoregressively simulate transitions
    (s_t, a_t, r_t, R_t) ~ P_θ(y_t | y_<t, x)
    C ← C ∪ {(y_<t ∘ (s_t, a_t, r_t, R_t))}
  w ← score(C)
  C_t ← filter(C, B, w)
return C.max()
```

In this case, the scoring function is $\texttt{score}(\mathbf{y}_{<t} \circ y_t) = \log P_\theta(y_t|\mathbf{y}_{<t})$. and the filter function often randomly picks the $\mathcal{B}$ most likely next tokens according to their score (i.e., the conditional probability distribution). This corresponds to the top-K (Fan et al., 2018) sampling, which is a beam-search variant. In the context of offline RL, Janner et al. (2021) used a different choice of these functions and modified BS to decode trajectories that achieve the maximum cumulative rewards. For this case, the log probabilities of transitions are replaced by the log probability of the predicted reward signal. Consequently, the scoring function is the expected cumulative reward added to the reward-to-go estimate while the filter function remains unchanged. As we will see shortly (sec. 4), our approach, WSTS, accounts not only for the expected rewards but also for the risk. We will explain WSTS using a particular choice of these functions.

3.3 Portfolio Optimization

In finance, a *portfolio* is defined as a combination of financial assets, each typically associated with some expected reward and risk. To form the portfolio, given such N different risky assets and wealth w, we need to decide how much wealth to invest in each asset, i.e., determine the vector of weights on assets 1 to N: $w = (w_1, w_2, ..., w_N)$. The *portfolio optimization* problem corresponds to selecting the best portfolio out of the set of all portfolios being considered, according to some objective. These objectives typically balance expected reward and risk allowing investors a principled way to maximize return while bounding risk. Modern portfolio theory (Markowitz, 1952) or mean-variance analysis solves the portfolio selection problem by taking only the mean and variance of the portfolio into consideration, weighing the risk, expressed as variance, against the expected return.

Expected utility theory estimates the utility of action when the outcome is uncertain. It takes into account that individuals may be risk-averse (Tversky and Kahneman, 1992), meaning they tend to prefer outcomes with low uncertainty to those with high uncertainty. The expected utility maximization is consistent with mean-variance analysis if the utility function is quadratic or if the asset returns are normally distributed (Levy and Markowitz, 1979). In such a case, the constrained portfolio-optimization problem can be formulated as a utility-maximization problem. Specifically, given some risk-aversion parameter δ, assets mean and variance vectors μ and Σ, the portfolio-optimization problem is solved by computing a weight vector w dictating the relative amount to invest in each asset by solving the following optimization problem:

$$\max_{w} \quad w^T\mu - \frac{\delta}{2} w^T \Sigma w. \tag{2}$$

4 Method: Wall Street Tree Search (WSTS)

In this section, we present "Wall Street Tree Search" (WSTS)—our approach for online risk-aware decoding. We expand beam search with portfolio optimization for sequential decision-making under uncertainty. Like portfolio optimization, our risk-aware planning algorithm is about budget allocation to different assets. However, the budget is not money but the computational effort in our setting. In our variation of beam search, we allocate the limited amount of node-expansion budget to determine which $\mathcal{B}$ trajectories we keep at each time step. To this end, we use portfolio optimization to weigh trajectories according to their "risk" (which is uncertainty in our setting), expected return, and our risk-aversion parameter (we discuss the implications of this parameter in Sec. 5).

We start by describing how we compute the mean vector and covariance matrix that will be used in Eq. 2. We then continue to detail how WSTS uses portfolio optimization to instantiate the generic `score` and `filter` functions described in Sec. 3.2.

4.0.1 Mean Vector and Covariance Matrix

Recall that to solve the portfolio-optimization problem (Eq. 2), in addition to the risk-aversion paramaeter δ, we require (i) a vector of expected returns, μ, and (ii) a covariance matrix Σ of the assets. To estimate these quantities, we use the Transformer decoder sequence model trained in the model-learning stage (Sec. 3.1). Namely, we use the variance computed from the Transformer's output as a measure of predictive uncertainty and as a proxy indicating the amount of distribution shift. This approach is taken due to Desai and Durrett (2020) which show that Transformer-based models are well-calibrated when trained with temperature scaling (Guo et al., 2017) as is done in our setting.

To start, recall that the Transformer is a multinomial classification model which at inference time autoregressively outputs a probability p_i of the subsequent trajectory token y_t^i at step t, conditioned on the preceding trajectory tokens $\mathbf{y}_{<t}$. Hence, by conditioning on each of the partial trajectories, the mean $E[y_t]$ and variance $\mathrm{Var}[y_t]$ of each output random variable y_t are simply $E[y_t] = \sum_{i=1}^{\mathcal{V}} p_i y_t^i$ and $\mathrm{Var}[y_t] = \sum_{i=1}^{\mathcal{V}} p_i \cdot (y_t^i - \mu)^2$, respectively.

Now, using these values to predict the vector of expected returns, μ and the covariance matrix Σ for all the candidate solutions on each time step requires more care. The longer the effective planning horizon, the more error the model inaccuracy introduces. Thus, in contrast to the common approach in RL where immediate rewards are incentived over long-term rewards via a discount factor $\gamma \in [0, 1]$ which exponentially scales down the rewards after each step, in our case we want to account for potential effect of compounding future risks. Specifically, in addition to scaling down the mean, we also scale up the variance, informing our downstream planner that the uncertainty increases for future predictions. Consequently, the means vector, μ, is an N-dimensional vector, consisting of the discounted means of the cumulative reward plus the reward-to-go estimate associated with each of the N candidate trajectories. The covariance matrix, Σ, is an $N \times N$ diagonal matrix consisting of the discounted variances of the cumulative reward plus reward-to-go estimate associated with each candidate trajectory along its main diagonal. Namely, μ_j and σ_j which are the jth entry in μ and Σ, respectively are defined as:

$$\mu_j = \sum_{t=1}^{T-1} \gamma^t \mathrm{E}[\mathrm{r_t^i}] + \gamma^T \mathrm{E}[\mathrm{\hat{R}_T^i}],$$

$$\sigma_j^2 = \sum_{t=1}^{T-1} \gamma^{-2t} \mathrm{Var}[\mathrm{r_t}] + \gamma^{-2T} \mathrm{Var}[\mathrm{\hat{R}_T}].$$

4.0.2 `score`

function: Given the mean vector μ, and the covariance matrix Σ, we compute a score for each of the N candidate trajectories available at time step t by solving a portfolio-optimization problem via Eq. 2 with the input risk-aversion parameter, δ. As stated, the intuition and implications of the value of δ are discussed in Sec. 5.

The output of the `score` function is the solution to the portfolio optimization problem, namely the wealth coefficients $w_1, \ldots, w_N$, representing the proportion of the our computational time (money) to invest in exploring each trajectory (asset).

4.0.3 `filter`

function: Given the coefficients $w_1, \ldots, w_N$ computed using the score function, our filter function samples $\mathcal{B}$ trajectories (with repetition) where the jth trajectory is sampled with probability w_j. Note that a trajectory may be sampled more than once. Thus, our practical beam width is smaller than $\mathcal{B}$ (but never bigger). This biases the search toward more-promising trajectories which helps mitigate the effect of aleatoric uncertainty inherent in the randomness of the observed data.

5 Experiments and Results

In our experiments, we aim to study the following questions:

Q1 How does WSTS perform compared to prior approaches?

Q2 How to choose the risk aversion parameter δ?

Q3 How does the risk aversion parameter δ affects the performance?

Our experimental evaluation focuses on continuous control tasks from the D4RL benchmark (Fu et al., 2020), namely the Gym-MuJoCo tasks: Walker2d, HalfCheetah, and Hopper. The different dataset settings are described below:

1. The "medium" dataset is generated by collecting 1 million samples from a partially-trained policy.
2. The "medium-replay" dataset records all samples in the replay buffer observed during training until the policy reaches the "medium" level of performance.
3. The "medium-expert" dataset is generated by mixing 1 million samples generated by the medium policy concatenated with 1 million samples generated by an expert policy.

We use MPC approach with WSTS (Sec. 3), interleaving planning and execution. We implemented WSTS using the same code base provided by Janner et al. (2021) and used the trained network weights and the same hyperparameters that the original paper's authors provided. As we use the original network as-is, we refer the reader to the original paper for the details. The only modification we do is alter the overarching planning algorithm that decodes the Transformer's outputs. [1] We use a portfolio optimization software package (Martin, 2021) to solve the portfolio optimization problem. We ran all tests on NVIDIA Tesla V100 GPUs.

5.0.1 (Q1) How does WSTS perform compared to prior approaches?

To answer this question, we compare our approach against five other prior SOTA methods spanning other approaches of offline RL: CQL (Kumar et al., 2020), MOPO (Yu et al., 2020), MBOP (Argenson and Dulac-Arnold, 2020), DT (Chen et al., 2021) and TT (Janner et al., 2021). WSTS performs on par with or better than all prior methods (Table 1). Moreover, WSTS consistently more stable than BS, reducing the variance considerably. To illustrate this, we compare WSTS to the modified beam search version used in the original Trajectory Transformer (TT) paper (Janner et al., 2021). We decode a trained Transformer model provided by the authors of TT using both algorithms and present the results in Fig. 1

5.0.2 (Q2) How to choose the risk aversion parameter δ?

Offline RL usually requires expensive online rollouts for hyperparameters search (for example, Janner et al. (2021) used six hyperparameters), which makes hyperparameters undesirable. However, our algorithm is robust when using diversified δ values, as can be seen Fig. 2, and we can outperform BS using a wide range of delta values. Moreover, in our case, we can get a hint on how to set the risk aversion parameter's value properly by observing the connection between the risk aversion parameter and the data batch: scilicet, the batch size (relative to the environment complexity), and the data quality. Generally, the more expert the behavior policy that generated the batch, the less risk-averse we need to be, which is expected since we have more trust in the behavior policy. The batch size also plays a role when setting the risk aversion parameter's value. Increasing the batch size reduces the subjective uncertainty, which can reduce the risk aversion parameter.

These observations are concise with imitation learning. In the extreme case of expert demonstrations with unlimited data, the optimal strategy would be to select a single action with the maximum

[1] Code will be made publicly available upon paper publicatin

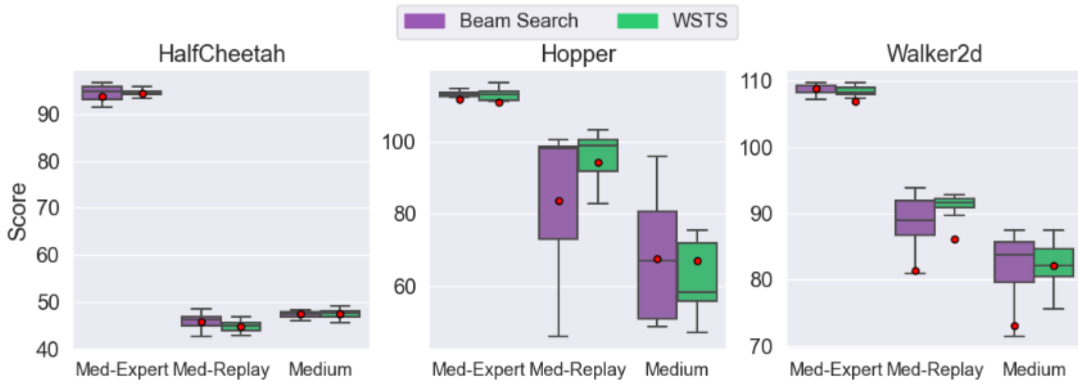

Figure 1: Variants correspond to the mean and standard error over 15 random seeds. The box extends from Q1 to Q3 quartile values of the results, with a line at the median (Q2) and a red dot on the mean. Whiskers restricted to a maximum of 1.5 times the interquartile range. Results of the Trajectory Transformer (Beam Search) are reconstructed from the code base provided by the original authors.

Dataset	**Environment**	**CQL**	**MOPO**	**MBOP**	**DT**	**TT (BS)**	**WSTS**	δ
Med-Replay	HalfCheetah	45.5	53.1	42.3	36.6	41.9 ± 2.5	44.8 ± 0.3	1.0
Med-Replay	Hopper	95.0	67.5	12.4	82.7	91.5 ± 3.6	94.2 ± 2.6	2.0
Med-Replay	Walker2d	77.2	39.0	9.7	66.6	82.6 ± 6.9	86.1 ± 3.8	2.0
Medium	HalfCheetah	44.0	42.3	44.6	42.6	46.9 ± 0.4	47.6 ± 0.2	0.1
Medium	Hopper	58.5	28.0	48.8	67.6	67.4 ± 2.9	67.0 ± 4.7	0.5
Medium	Walker2d	72.5	17.8	41.0	74.0	79.0 ± 2.8	82.1 ± 0.8	1.0
Med-Expert	HalfCheetah	91.6	63.3	105.0	86.8	95.0 ± 0.2	94.4 ± 0.3	0.1
Med-Expert	Hopper	105.4	23.7	55.1	107.6	110.0 ± 2.7	111.0 ± 1.6	0.1
Med-Expert	Walker2d	108.8	44.6	70.2	108.1	101.9 ± 6.8	107.0 ± 1.5	0.1
Average		77.6	42.14	47.8	74.7	78.9	81.6	

Table 1: (Offline RL) Results for WSTS variants correspond to the mean and standard error over 15 random seeds. Results for the other algorithms are taken from the original papers.

expected reward. Such an approach is equivalent to zeroing the risk aversion parameter, which narrows the beam width $\mathcal{V}$ to one.

In our experiments, we can get affirmation for this observation when examining the comparison between BS and WSTS in the medium-expert dataset (The left boxplot in each image in Fig. 1). Recall the medium-expert dataset was generated by a well-trained behavior policy. In addition, this dataset contains twice as many samples as other datasets. For such a dataset, reward-maximizing behavior (that does not account for the risk) might be a good option. Indeed WSTS does not provide a significant improvement over BS on this dataset.

5.0.3 (Q3) How does the risk aversion parameter δ affects the performance?

This question is relevant for scenarios where system engineers can select the amount of risk aversion. Such a selection generally depends on the application and system domain. Fig. 2 presents a kernel density estimate plot that shows that by decreasing the risk aversion parameter, it is possible to achieve higher results if we are willing to take the risk of failing.

6 Conclusion and Future Work

We introduced WSTS, a planning algorithm that suggests a constructed way to control the risk, and demonstrated it on an offline model-based RL setting. Our planning algorithm leverages the capacity of the learned model to generalize to states outside the static batch support. Still, it is cautious when drifting to states where the model can't give a confident prediction based on the offline dataset. For

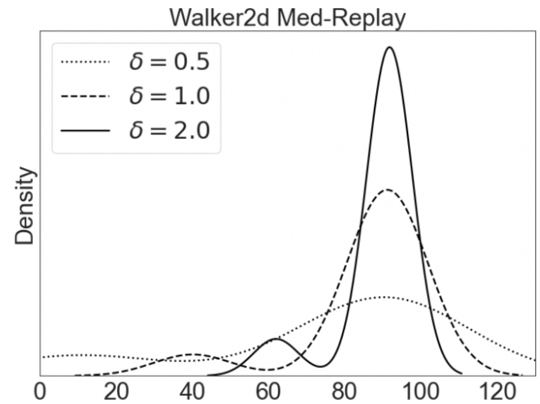

Figure 2: A kernel density estimate plot, presenting WSTS sensitivity to the risk aversion parameter δ. Results showed for Walker2d medium-replay dataset over ten random seeds for each risk aversion value.

such states, our planner weighs this risk vs. the expected return by incorporating modern portfolio theory into sequential decision-making. To this end, WSTS introduces a risk aversion parameter, which can be tuned, allowing the system designer to reflect its preferences about outcomes with low uncertainty explicitly but possibly lower return vs. outcomes with high uncertainty, even if the return can be higher.

Our experiments show that WSTS outperforms state-of-the-art offline RL methods in the D4RL standard benchmark. Moreover, it is consistently more stable, reducing the variance considerably. The broad definition of regularization is a statistical procedure that gives more stable estimates. Hence, we can consider WSTS as a search regularizer, keeping the search roughly within reasonable bounds.

In this work, we rely on temperature scaling for calibration. However, a question remains whether different Bayesian formulations of deep learning can potentially improve predictive uncertainty quantification can further improve our results. This question is left for future research.

A different potential research path to improve our algorithm is to improve the portfolio selection. One way to improve the portfolio selection is by accounting for the dependency between trajectories when solving the portfolio optimization by filling in the corresponding entries in the covariance matrix. Another way to improve our algorithm is using better portfolio optimization methods which have come a long way from Markowitz (1952) seminal work that introduced the mean-variance risk management framework. Using such methods, we can revisit candidate trajectories' selection order (**score** function). For example, Markowitz's model intrinsically assumes that the portfolio returns follow a normal distribution, which enables measuring the risk by its standard deviation. However, the returns do not follow a normal distribution in our case. In such a case, this model may have undesirable properties. Moreover, we are more interested in the risk of incurred loss than the variability of returns measured by standard deviation. Current approaches address this issue by formulating risk management in terms of Value-at-Risk (VaR) (Markowitz, 1959) or conditional VaR (CVaR) (Rockafellar and Uryasev, 2002) to measure and control the level of risk exposure. In future work, we hope to explore these directions.

Sutton and Barto (1998) Levine et al. (2020)

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. 2019. doi: 10.48550/ARXIV.1907.04543. URL `https://arxiv.org/abs/1907.04543`.

Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. In *International Conference on Learning Representations (ICLR)*, 2020.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1877–1901, 2020.

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, 2021. URL `https://arxiv.org/abs/2106.01345`.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *ACM Conference on Recommender Systems*, pages 191–198, 2016.

Shrey Desai and Greg Durrett. Calibration of pre-trained transformers. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 295–302, 2020.

Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT)*, pages 4171–4186, 2019.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Association for Computational Linguistics (ACL)*, pages 889–898, 2018.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020. URL `https://arxiv.org/abs/2004.07219`.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning (ICML)*, pages 2052–2062, 2019.

Omer Gottesman, Fredrik Johansson, Joshua Meier, Jack Dent, Donghun Lee, Srivatsan Srinivasan, Linying Zhang, Yi Ding, David Wihl, Xuefeng Peng, Jiayu Yao, Isaac Lage, Christopher Mosch, Li-wei H. Lehman, Matthieu Komorowski, Matthieu Komorowski, Aldo Faisal, Leo Anthony Celi, David Sontag, and Finale Doshi-Velez. Evaluating reinforcement learning algorithms in observational health settings, 2018. URL `https://arxiv.org/abs/1805.12298`.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, pages 1321–1330, 2017.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem, 2021. URL `https://arxiv.org/abs/2106.02039`.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog, 2019. URL `https://arxiv.org/abs/1907.00456`.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel : Model-based offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 21810–21823, 2020.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293, 2006.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11761–11771, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1179–1191, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. URL `https://arxiv.org/abs/2005.01643`.

H. Levy and H. M. Markowitz. Approximating expected utility by a function of mean and variance. *American Economic Review*, 69(3):308–317, 1979.

B. Lowerre and R. Reddy. The harpy speech recognition system: Performance with large vocabularies. *The Journal of the Acoustical Society of America*, 60(S1):S10–S11, 1976.

Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

Harry M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Yale University Press, 1959.

Robert Martin. Pyportfolioopt: Portfolio optimization in python. *Journal of Open Source Software*, 6(61):3066, 2021.

Clara Meister, Ryan Cotterell, and Tim Vieira. If beam search is the answer, what was the question? In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 2173–2185, 2020.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL `https://arxiv.org/abs/1910.00177`.

J. Richalet, A. Rault, J. L. Testud, and J. Papon. Model predictive heuristic control. applications to industrial processes. *Automatica*, 14(5):413–428, 1978.

R.Tyrrell Rockafellar and Stanislav Uryasev. Conditional value-at-risk for general loss distributions. *Journal of Banking Finance*, 26(7):1443–1471, 2002.

Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 19:70–76, 2017.

Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Michael Neunert, Thomas Lampe, Roland Hafner, and Martin Riedmiller. Keep doing what worked : Behavioral cloning priors for fully offline learning. In *International Conference on Learning Representations (ICLR)*, 2020.

Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. Learning from logged implicit exploration data. *Advances in Neural Information Processing Systems*, 23, 2010.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, 5(4):297–323, 1992.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, page 6000–6010, 2017.

Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 2447–2456, 2018.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning, 2019. URL `https://arxiv.org/abs/1911.11361`.

Chao Yu, Guoqi Ren, and Jiming Liu. Deep inverse reinforcement learning for sepsis treatment. In *International Conference on Healthcare Informatics (ICHI)*, pages 1–3, 2019.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 14129–14142, 2020.