Model-based planning uses dynamics of the environment to plan actions. Compared with model-free reinforcement learning, model-based planning has higher data efficiency and the learned dynamics can be potentially used for other tasks in the environment (Hafner, Lillicrap, Fischer, et al., 2019). However, model-based planning usually requires the learnt dynamics to be highly accurate and efficient (Buesing et al., 2018).

Model-based optimal control algorithms such as iterative Linear Quadratic Regulator (iLQR) have been successfully applied to many science and engineering fields (Li & Todorov, 2003). These control algorithms assume a known system model and relatively low dimensional state representations. Raw sensory data such as images are highly non-linear and has high dimensions and control algorithms are usually not able to be applied directly on images (Watter, Springenberg, Boedecker, & Riedmiller, 2015). Instead of developing optimal control algorithms on raw sensory data, models that learn a low dimensional latent space for optimal control are developed.

Models such as Deep Dynamical Model (DDM), Embed to Control (E2C), Robust Controllable Embedding (RCE) uses variational auto-encoders (VAE) to extract low dimensional feature from images and build transition model in low dimensional space (Banijamali, Shu, Ghavamzadeh, Bui, & Ghodsi, 2018; Wahlström, Schön, & Deisenroth, 2015; Watter et al., 2015). These models are further combined with control algorithms, including adaptive Model Predictive Control (MPC) algorithm, iLQR (Wahlström et al., 2015; Watter et al., 2015). Recent work such as Deep Planning Network (PlaNet) and Dreamer, use a recurrent state-space model (RSSM) as a predictive model in latent space

(Hafner, Lillicrap, Ba, & Norouzi, 2019; Hafner, Lillicrap, Fischer, et al., 2019). PlaNet uses the cross entropy method as planning algorithm and Dreamer agent learns action model and value model (Hafner, Lillicrap, Ba, et al., 2019; Hafner, Lillicrap, Fischer, et al., 2019).

Objective:

Literature Review

LQR

Linear Quadratic Regulator (LQR) is an optimal feedback controller for linear systems in state-space form (Almaged, Khather, Abdulla, & Amjed, 2019). Consider the finite horizon discrete linear time-invariant system dynamics:

The goal is to find a sequence of control input such that the cost

is minimized. is solved by dynamic programming from backward.

MPC

LQR relies on linear dynamics and does not consider any constraints, while some real-world systems that are capable to achieve complex problems are non-linear and some systems have constraints on controls(Amos, Rodriguez, Sacks, Boots, & Zico Kolter, 2018; Li & Todorov, 2003).

Problems with non-convex dynamic functions and cost functions are solved with Model Predictive Control (MPC). MPC includes a group of control algorithms that use a predictive model of a controlled system to determine control actions by solving constrained optimization problems(Schwenzer, Ay, Bergs, & Abel, 2021).

Consider a discrete time nonlinear dynamic system :

, where is the state, is the control at time .

The goal of the optimization problem at each time step is to find sequence of control input such that the cost function

, where and are constraints on states and controls and is a cost function (Amos et al., 2018). MPC applies only the first control action to the system, then repeat prediction and solve the optimization problem for the next time step.

iLQR

Iterative Linear Quadratic Regulator (iLQR) algorithm solves MPC optimization problem by linearizing the non-linear system around a nominal trajectory iteratively and applying a modified LQR method to compute a locally optimal feedback control(Li & Todorov, 2003). Define the value function as: . ILQR algorithm consists of a backward pass and a forward pass in each iteration. In backward pass, the algorithm minimizes with respect to . In forward pass, it computes a new trajectory using the calculated in the backward pass. The iterative algorithm terminates until decrease of cost between two iterations is less than a tolerance or a maximum number of iterations is hit (Tassa, Mansard, & Todorov, 2014). Due to limited numerical precision, regularization is required for iLQR solvers (Tassa, Erez, & Todorov, 2012).

A box-DDP algorithm(Tassa et al., 2014) is proposed to solve MPC with box constraints on :

, where and are lower bound and upper bound of . The control limit is directly considered in the minimization of step.

Based on this box constrained solver, a time and memory efficient approach which uses analytical differentiation is developed to solve box constrained MPC (Amos et al., 2018).

Variational auto-encoder (VAE)

Variational auto-encoder (Kingma & Welling, 2013) uses the auto-encoding variational Bayes algorithm to perform posterior inference and learn model parameters. Consider latent variable and variable ,

VAE can perform efficient 1) approximate maximum likelihood (ML) or maximum a posteriori (MAP) inference for , 2) approximate posterior inference , 3) approximate marginal inference (Kingma & Welling, 2013).

Let be an arbitrary distribution. Kullback-Leibler divergence (KL divergence) between and is defined as follows:

(Doersch, 2016).

VAE approximates the posterior with a recognition model parameterized by a neural network (Kingma & Welling, 2013). The log probability of can be written as:

, where is the variational lower bound (Kingma & Welling, 2013). VAE optimizes the approximate posterior inference by maximizing the lower bound using stochastic gradient (Kingma & Welling, 2013).

A number of models belong to VAE are developed to learn latent dynamics, including E2C, RCE and RSSM.

Embed to control and RCE

E2C (Watter et al., 2015) learns an inference model , a generative model and a transition model following iLQG formulation, where is observation, is latent state and is control. The inference model and transition model follow Gaussian distributions where the mean and variance are parameterized by an encoding neural network and transformation network. The generative model follows a Bernoulli distribution parameterized by a decoding network.

RCE (Banijamali et al., 2018) is developed based on E2C to address two problems of E2C that suffers from: 1) E2C does not achieve optimal lower bound of data likelihood; 2) Future observations are not considered in variational inference. RCE model separates the generative model from variational recognition model and is robust with respect to noises.

Recurrent State Space Model (RSSM), PlaNet and Dreamer

As observations from environment (images) are high dimensional and not efficient for model-based planning, it is assumed that state space models (SSMs) can extract features and predict in a low-dimensional latent space (Buesing et al., 2018). Let be the observation, hidden state, action and reward at time step . Deterministic SSMs (dSSMs) use a deterministic transition function and stochastic SSMs (sSSMs) uses to model uncertainty.

PlaNet (Hafner, Lillicrap, Fischer, et al., 2019) uses RSSM as the predictive model in latent space, which contains both deterministic transition and stochastic transition:

Deterministic state model:

Stochastic state model:

Observation model:

Reward model:

, where is a activation vector for reusing computations before time step . Figure 1 illustrates the structure of RSSM used by PlaNet. The latent state is a combination of both stochastic part and deterministic part.

A picture containing text, clock, watch

Description automatically generated

Given the learnt latent space model, PlaNet considers a partially observable Markov decision process (POMDP):

Transition function:

Observation function:

Reward function:

Policy: ,

and utilizes CEM to plan and maximize the expectation of sum of rewards (Hafner, Lillicrap, Fischer, et al., 2019).

Dreamer (Hafner, Lillicrap, Ba, et al., 2019) also uses RSSM as latent model to learn a representation model , a transition model and a reward model . Besides, Dreamer learns an action model and a value model which can estimate sum of rewards beyond the horizon using the predicted model in latent space.

Methodology

We approach the problem by using the representation model, observation model and transition model from PlaNet (Hafner, Lillicrap, Fischer, et al., 2019) and Dreamer (Hafner, Lillicrap, Ba, et al., 2019) to learn a latent space model and performing optimal control in the latent space using the differentiable MPC solver (Amos et al., 2018).

This section describes the modified world model from Dreamer (Hafner, Lillicrap, Ba, et al., 2019) which learns a latent space model from image observation.

Let denote observation, action, latent state (RSSM state), deterministic state, stochastic state and observation embedding at time step .

RSSM state:

A RSSM state contains four components: mean and variance of stochastic state, stochastic state, and deterministic state.

State model (Observation encoder):

State model is a convolutional neural network that encodes an observation image into a one-dimensional embedding .

Transition model:

Transition model is a RSSM as described in the previous chapter. The deterministic state model is implemented as a recurrent neural network (RNN) and the stochastic model is implemented as a feed-forward network.

The transition model takes in the action and the RSSM state at previous timestep. RNN outputs a deterministic state given the previous action, stochastic state and deterministic state. Feed-forward network outputs the mean and variance of the Gaussian distribution given the deterministic state . A stochastic state is sampled from the Gaussian distribution. The output RSSM state is constructed as a tuple (.

Representation model:

Representation model first computes the prior RSSM state using the transition model. The posterior stochastic model is a feed-forward network that takes in the deterministic state from prior RSSM state and the observation embedding and outputs the mean and variance of the Gaussian distribution. A stochastic state is sampled from the Gaussian distribution. The posterior RSSM state is constructed as a tuple (.

The representation model ouputs the prior RSSM state and the posterior RSSM state .

Observation model (Observation decoder):

Diagram

Description automatically generatedThe observation model takes in the RSSM state feature which is the deterministic state and stochastic state of a RSSM state. The decoder outputs a Gaussian distribution whose mean is an observation computed from a convolutional neural network and the variance is constant.

Model training

As derived in PlaNet (Hafner, Lillicrap, Fischer, et al., 2019), the one-step prediction variational lower bound is

, and it is further generalized into a multi-step prediction lower bound:

. The model trained by maximizing this multi-step prediction lower bound.

Figure 2 illustrates the loss of the model as described above for one-step prediction. is transformed to an embedding through the observation encoder. Then the embedding, previous action and previous RSSM state is feed into the representation model to get a prior RSSM state and a posterior RSSM state. The observation decoder takes in the posterior RSSM state to get observation prediction. The image\_loss computed from the observation and prediction corresponds to and the KL divergence between prior distribution and post distribution corresponds to .

The learnt transition model is able to predict in the latent space, thus we can use MPC to plan in the latent space. This section describes the MPC planner that we used.

A differential MPC library (Amos et al., 2018) provides APIs implemented in PyTorch to solve the optimization problem:

.

Dynamic system:

We use a dynamic function based on the transition model . The transition model takes in a RSSM state, which is a tuple (mean, variance, stochastic state, deterministic state) and an action and output a predicted RSSM state. We drop the mean and variance and use the feature of a RSSM state, concatenation of stochastic state and deterministic state . Thus, , . The RNN inside the transition model outputs . The feed-forward network outputs the meanand variance of the Gaussian distribution given the deterministic state . Then sample a stochastic state from the Gaussian distribution. The output is .

Cost function: ),

Assume that there exists a goal state in the latent space and the target control is a zero vector. . To control the system towards the goal state, we use a quadratic cost function:

, where is a weight vector. This is implemented as a quadratic form in the library as follows:

. Thus the optimization problem is formulated as:

, where , .

The goal state in the latent state is obtained by passing the embedding of the image observation of the system in the goal state to the representation model with zero previous RSSM state and zero action.

The MPC library outputs a sequence of controls given the dynamic function, cost function and initial state. We repeatedly call the MPC library to solve the optimization problem at each time step and use the first outputted control to control the original system to get the next state.

MPC planner

We abstract APIs to the differential MPC library (Amos et al., 2018) into a MPC planner which provides three APIs: , and .

: As the world model parameters change during training, the goal state in the latent space changes. This function updates the cost function parameters for the MPC planner using the updated goal state.

: Clean up the MPC planner.

: Output the first action from the action sequence outputted from the MPC library given the initial state.

Implementation

|  |  |
| --- | --- |
| Algorithm 1: Model training | |
| Input: | |
| S Seed episodes | |
| C Collect interval | |
| B Batch size | |
| L Chunk length | |
| exploration rate | |
|  | |
| 1: | Initialize dataset with random seed episodes |
| 2: | Initialize neural network parameters randomly |
| 3: | Initialize mpc\_planner |
| 4: | **while** not converged **do** |
| 5: | **for** update step s = 1.. **do** |
| 6: | Draw data sequences |
| 7: | Compute loss and update model parameters |
| 8: | Update goal state in mpc\_planner |
| 9: |  |
| 10: | **for** time step t = 1.. **do** |
| 11: | observation\_encoder ( |
| 12: | representation\_model() |
| 13: | random() |
| 14: | **if** **then** |
| 15: | action\_space.random() |
| 16: | **else** |
| 17: | mpc\_planner (get\_feature()) |
| 18: | **end if** |
| 19: |  |
| 20: | } |