Active Learning - Plasma Surrogate Model

José Filipe Afonso 15/04/2025

Introduction and Problem Formulation

Creating efficient surrogate models that modulate the output of difficult-to-compute physical simulators that model the surface plasma-surface interactions is highly desirable.

So far, this approach has been widely explored from a **supervised learning** perspective. A dataset is collected using a design filling algorithm like the Latin hypercube sampling. Then, as a posterior and independent task, a model is created to predict the dataset as closely as possible (given that we define the loss criteria). In this second stage, different architectures and hyperparameters are tested and fine-tuned.

This work pretends to follow a different direction. Instead of treating the sample collection and the surrogate model developing as two independent tasks, we pretend to combine them, following a perspective of **active learning**. By taking advantage of the uncertainties and expected changes in the surrogate model, we hope to enhance its accuracy and make it more sample-efficient without changing the previously defined model architectures. Indeed, this last task has been studied in a different work, which is out of scope here. However, future development could involve applying the methods developed here to enhance *offline* models.

In a different context, the active learning approach has been successfully applied to other systems and tasks, such as Lewis et al. *A Sequential Algorithm for Training Text Classifiers*, which motivates the present study.

Additionally, it would be beneficial if the dataset we collected improved future offline models, as this could facilitate the learning process.

Mathematical Formalization in an RL Framework

The problem of choosing the next samples (actions) in an active learning context can be framed as a sequential decision-making process as follows:

State Space ($\mathcal{S}$)

The state s_t at time t represents the current "knowledge" about the the input space as encoded by the uncertainty function.

$$s_t = \{u_t(x_1), u_t(x_2), \ldots, u_t(x_n), \phi_t\} \tag{1}$$

where $\{x_1, \ldots, x_n\}$ are representative points and ϕ_t corresponds to the global uncertainty (e.g. average of all representative points)

Action Space ($\mathcal{A}$)

An action a_t involves selecting a subset of new sample points $\{x_{a_1}, x_{a_2}, \ldots, x_{a_k}\}$ from a candidate pool. These are the new points to be queried (or simulated) in order to gather additional information data.

$$a_t \subset \mathrm{X}_{candidates} \tag{2}$$

where constraints like ensuring coverage (e.g. maintining diversity in the set a_t) may be imposed.

Transition Dynamics ($\mathcal{P}$)

The transiton from s_t to s_{t+1} is governed by the acquisition of new data and subsequent update of the predictive model. The update effects the uncertainty function:

$$s_{t+1} = f(s_t, a_t, o_t) \tag{3}$$

Where o_t represents the observed outcomes (e.g. simulator calls) at the chosen points a_t and f denotes the update of the surrogate model.

Reward Function ($\mathcal{R}$)

The reward r_t is designed to capture the value of reducing degree of uncertainty or the increment of the expected improvement. For each point x in the candidate pool (or the list that composes a state) is computed the reward or also called the acquisition function $A(\cdot)$

$$r_t = A(\cdot) \tag{4}$$

The $A(\cdot)$ considered function will be widely explained in a following section

Objective

The goal is to select a sequence of action(s) (i.e. sample points) that maximizes the expected cumlative reward over a horizon T:

$$\max_{\{a_t\}} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right] \tag{5}$$

However, in the case of our system considering $T > 1$ implies to simulate the evolution of the ensemble, which quickly becomes intractable. As a result, we simplify the objective and adopt a myopic and greedy strategy given as follows:

$$T = 1 \quad \max_{a_t} A(a_t) \tag{6}$$

So, at each step, we pick the action that maximizes $A(\cdot)$, without having in mind the long term effects. In this case, our approach is more related to a model-based RL algorithm with a one-step planning rollout

Problem seen in a probabilistic view

State - Candidate Pool - Prior Sampling

The definition of the state described previously corresponds to a belief over the input space given by the **prior** distribution $p(x)$.

We use the space the space-filling design to sample a set of candidate points:

$$\{x_1, x_2, \dots, x_N\} \sim p(x) \tag{7}$$

If we further add a Gaussian perturbation (jitter) to each sample:

$$\begin{aligned} x_i' &= x_i + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_j^2 I) \\ x_i' &\sim \overline{p}(x) = (p * \mathcal{N}(0, \sigma_j^2 I))(x) \end{aligned} \tag{8}$$

The jitter amount σ_j can be scheduled and represents a change in the prior belief.

Action and Rewards - Select the samples from the candiadate pool

We observe the data $D = \{x_i, y_i\}$ from the physical simulator, we train the ensemble of nets that provide a predictive distribution for a new input x:

$$p(f(x)|x, D) = \mathcal{N}(\mu(x), \sigma^2(x) I) \tag{9}$$

The acquisition function $A(x)$ allows to select the points of the candidate pool that maximize a giving score. As a result, the **posterior** for selecting candidates, given a state s_t, can be written as:

$$p(x|D) \propto p(x) l(D|x) \propto p(x) A(x) \tag{10}$$

Acquisiton (Reward) function definition

The proposed acquisition function pretends select the most *informative* samples for training the neural ensemble. This function simultaneously accounts for three critical criteria: the model's uncertainty about the ouput, $U(x)$, the expected change in the model's parameters (gradient update magnitude), $G(x)$, and the coverage of the candidate sample with respect to the current data distribution, $\rho(x)$. They are defined as follows:

- **Uncertainty measure** $U(x)$:

  This metric quantifies the ensemble's uncertainty. In our implementation, it is computed as the standard deviation of the predictions $\{y_i\}_{i=1}^{M}$ provided by M ensemble members:

$$U(x) = \text{std}\left(\{y_i\}_{i=1}^{M}\right) \tag{11}$$

-

- **Gradient Update Estimate** $G(x)$:

  This term quantifies the expected magnitude of the update of the model parameters if the point x was added to the training set. For each ensemble member i with parameters θ and loss function $L(\cdot, \cdot)$, let

$$g_i(x, y) = \nabla_{\theta_i} L(f_i(x; \theta_i), y) \tag{12}$$

  be the gradient vector when. x is associated with a target y. Since y is unknown at the query time, we define an expected gradient norm over the predictive distribution $p(y|x)$:

$$\begin{aligned} G(x) &= \frac{1}{N} \sum_{i=1}^{M} \int ||g_i(x, y)||^2 p(y|x) dy \\ &\approx \frac{1}{N} \sum_{i=1}^{M} ||g_i(x, \hat{y}_i)||^2 \quad \text{where} \;\; \hat{y}_i \sim \mathcal{N}(\mu_i(x), \sigma_i^2(x)) \end{aligned} \tag{13}$$

- **Coverage Measure** $\rho(x)$:

  To favor samples that are representative of regions with high density, we define the measure $\rho(x)$ that corresponds to a function that increases with the representativeness of x:

$$\rho(x) = 1/\min(d(x, D)) \tag{14}$$

To combine these heterogeneous metrics into a unified acquisition function, we first normalize each component so that they take values in the interval $[0, 1]$. So, we define the normalized version $\tilde{F}(x)$ via *min-max* normalization over the set of candidates X:

$$\tilde{F}(x) = \frac{F(x) - \min_{x' \in \mathrm{X}} F(x')}{\max_{x' \in \mathrm{X}} F(x') - \min_{x' \in \mathrm{X}} F(x')} \tag{15}$$

and so the acquisition function $A(x)$ is given by:

$$A(x) = \alpha\tilde{U}(x) + \beta\tilde{G}(x) + \gamma\tilde{\rho}(x) \tag{16}$$

with the weights satifying: $\alpha, \beta, \gamma \geq 0$ and $\alpha + \beta + \gamma = 1$

In this way the Acquisition function $A(x)$ is designed to select candidate samples that are simultaneously:

- **Uncertain**: The ensemble exhibits a high variance in its predictions for x;
- **Informative**: The expected gradient update (and thereby potential improvement) is high
- **Representative**: The sample x lies in an underexplored region of the input space.

Pipeline and Model Architecture

The complete pipeline is available on GitHub [https://github.com/joseAf28/PlasmaActiveLearning]. For experimental purposes, we initially use synthetic data generated from a known function $F : \mathbb{R}^3 \to \mathbb{R}^{10}$ (defined in *PhysicalModel.py*) to validate the framework. The surrogate model comprises an ensemble of neural networks with the following hyperparameter configuration:

```
config = {
    'ensemble_size': 10,
    'input_size': 3,
    'hidden_size': 256,     # For each ensemble's neural net
    'output_size': 10,
    'num_epochs': 70,       # For each state
    'batch_size': 32,
    'learning_rate': 1e-2,
    'candidate_pool_size': 150, # Candidate Pool size
    'n_samples': 200,       # Initial buffer size
    'subset_frac': 0.7,     # Memory buffer fraction to mitigate catastrophic
forgetting
    'n_actions': 15,        # Number of samples taken at each action
    'lambda_vec': np.array([0.25, 0.7, 0.05])  # Weights for the acquisition
function: (α, β, γ)
}
```

This configuration represents a trade-off between model complexity and computational feasibility.

Experimental Results and Comparison

Baseline Models

- **Baseline 1:**
  A neural network with an identical architecture and training regimen as each ensemble member. Its dataset is generated using traditional Latin hypercube sampling.
- **Baseline 2:**
  A neural network trained on data collected by the ensemble predictor.

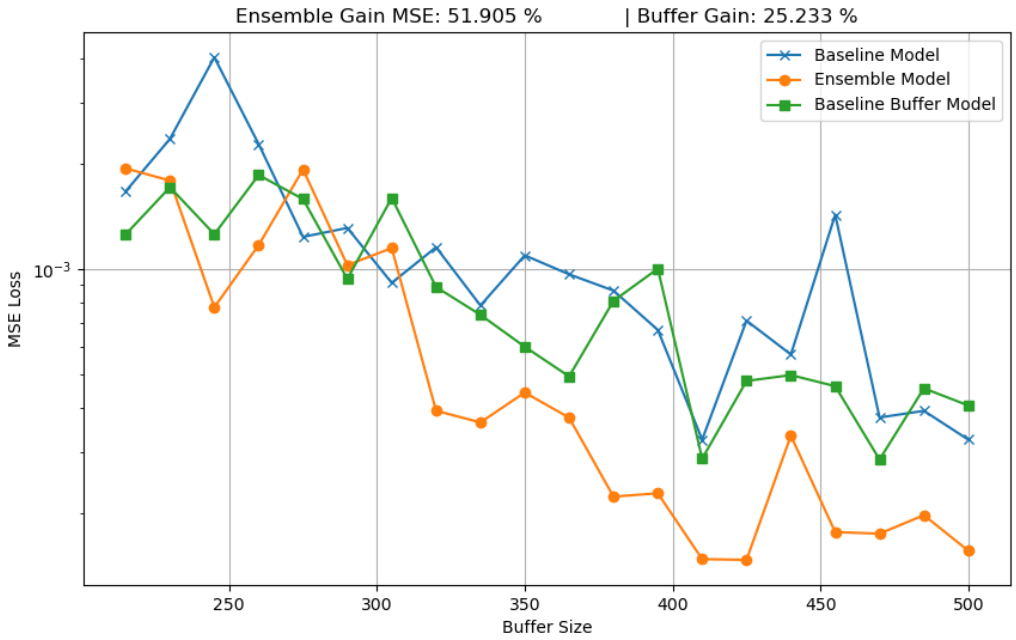
Performamce Analysis

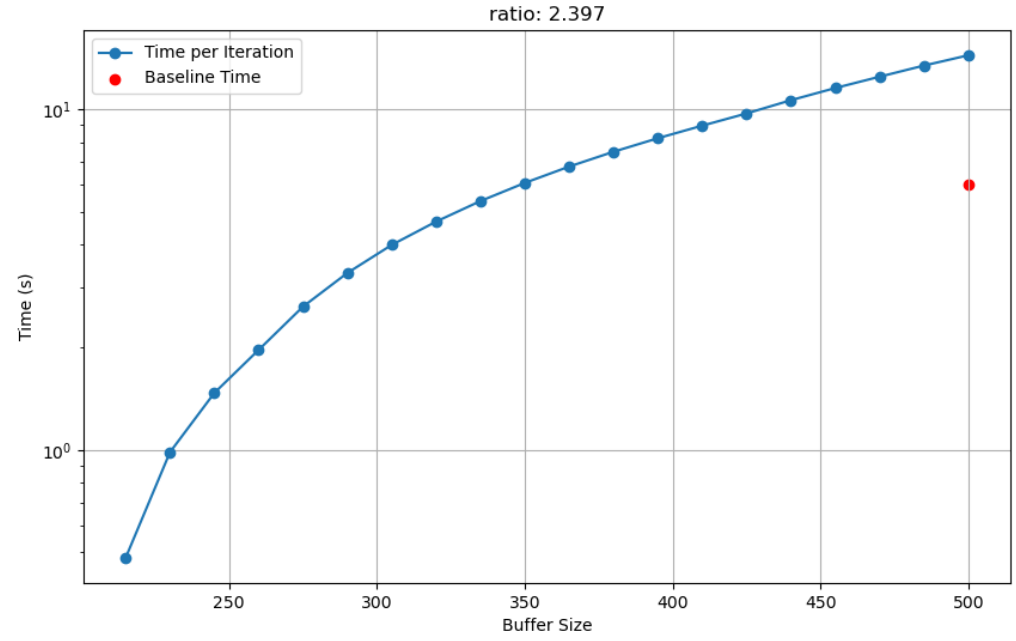
Our experiments reveal that the ensemble predictor, enhanced through active learning, consistently outperforms the baseline models. Key observations include:

Mean Squared Error (MSE) Improvement:
The ensemble model exhibits significant MSE reductions, achieving gains up to 52% compared to the baseline predictor when evaluated at 500 buffer size.

Computational Overhead:
While the ensemble model requires approximately 2.4 times the computational effort compared to the baseline model (with a 500-sample buffer), the training times remain tractable—the baseline model requires less than 10 seconds to train.

Insights on Data Buffering:
Although the ensemble predictor benefits from using its internally collected dataset, a similar gain is not observed when a new model (with independently generated weights) is trained on the buffered data. This discrepancy highlights the importance of ensemble-specific weight initialization and learning dynamics, which merit further investigation.





Discussion and Next Steps

Our active learning framework illustrates a strategy for improving surrogate modeling in simulation tasks. The integration of uncertainty quantification, expected gradient updates, and coverage metrics into the acquisition function aids in efficiently directing the sampling process. Future work may involve:

- **Integration with the LoKI Simulator:**
  Exploring the pipeline with the physical simulator LoKI to validate performance on real-world data.
- **Hyperparameter Fine-Tuning:**
  Refining the model hyperparameters and acquisition function weights to further enhance performance and sample efficiency.
- **Extension to Offline Models:**
  Investigating whether the active learning gains can be transferred to improve offline models, potentially through a two-stage training process.

References

Shin et al., *Neural Bootstrapper*

Gramacy. *Gaussian process modelig, design and optimization for applied sciences*

Osband et al., *Deep Exploration via Bootstrapped DQN*

Lewis et al., *A Sequential Algorithm for Training Text Classifers*

Sergey Levine , Lecture 11 and 13 - CS 285 RAIL

Smith, Freddie Bickford, et al. *Prediction-Oriented Bayesian Active Learning*. 17 Apr. 2023. arXiv:2304.08550, https://arxiv.org/abs/2304.08550.