Do Not Learn If You Trust Yourself: Efficient Resource-Constrained Online Learning Using Uncertainty

Bachelor's Project Thesis

Michal Tešnar, s4740556, m.tesnar@student.rug.nl,
Internal Supervisor: Dr Matias Valdenegro-Toro, m.a.valdenegro.toro@rug.nl,
External Supervisor: Dr.-Ing. Bilal Wehbe, bilal.wehbe@dfki.de

Abstract: Machine learning proves effective in constructing dynamics models from data, especially for underwater vehicles. Continuous refinement of these models using incoming data streams, however, often requires storage of an overwhelming amount of redundant data. This thesis investigates the use of uncertainty in the selection of data points to rehearse in online learning when storage capacity is constrained. The models are learned using an ensemble of multilayer perceptrons as they perform well at predicting epistemic uncertainty. We present three novel approaches: the *Threshold* method, which excludes samples with uncertainty below a specified threshold, the *Greedy* method, designed to maximize uncertainty among the stored points, and *Threshold-Greedy*, which combines the previous two approaches. The methods are assessed on data collected by an underwater vehicle Dagon. Comparison with baselines reveals that the *Threshold* exhibits enhanced stability throughout the learning process and also yields a final model with the lowest testing loss. We also conducted detailed analyses on the impact of model parameters and storage size on the performance of the models.

1 Introduction

In recent years, machine learning techniques have become popular for obtaining information from streams of data, thereby replacing manual data aggregation. In robotics, machine learning can be applied to learn a dynamics model of the controlled vehicle. The dynamics model describes the behavior of the vehicle over time, based on its states and inputs. Instead of constructing the equations that govern the model, the relation can be inferred from the recorded states of the vehicle during its operation (Wehbe et al., 2019). This can be done by *direct modeling* which aims to create a model that predicts the next state given the current state of the model (Thuruthel et al., 2017).

In underwater robotics, this allows us to directly learn complex models without sacrificing performance due to any simplifying assumptions. Moreover, we can avoid expensive computations of *Navier-Stokes* equations that describe the motion of fluids. Underwater dynamics present a highly non-linear task, hence flexible modeling approaches need to be used. Neural networks are found to perform the best in these settings (Wehbe, 2020). The obtained model can later be applied in the control of *autonomous underwater vehicles* (AUVs), for example, in a technique called model predictive control, which uses the dynamics model to predict the future states of the vehicle to find the most optimal action at each time to achieve its goal (Henson, 1998).

What is more, the underwater environment is very challenging. The density of the surrounding liquid can change, and the robot's body can wear over time, or parts of sea fauna might get attached to it (Wehbe, 2020). Furthermore, the vehicle might perform new actions or its thrusters might break down. To achieve optimal control in this changing environment, it is desirable to be able to continuously update the dynamics model of the vehicle during its operation based on the newly collected data. At the same time, the model needs to preserve its past knowledge. Simply retraining the learner on new data incoming might lead to loss of previous knowledge, commonly referred to as *catastrophic forgetting* (French, 1999). We want to achieve what is commonly referred to as *incremental online learning*, the process of adapting the model without losing previously learned information (Part & Lemon, 2017).

One hypothetical solution for the stated problem is to continuously store all collected data. To update the model, we could retrain it from scratch using all the stored data. This would preserve all the acquired knowledge from previous data points. However, in real-life scenarios, AUVs are deployed underwater for long periods, and they collect massive amounts of data from their sensors. This makes it virtually impossible to store all collected data and train on it, assuming the finite resources of AUVs.

What is more, many of the collected data points inevitably hold a lot of redundancy, as the AUV might be performing the same maneuver repeatedly in a similar environment. This means, that updating the model with information from every collected point may not be beneficial. What is more, the computation costs associated with these steps could be eliminated, thereby saving precious resources for the AUV.

Given the imposed constraint on the amount of data as well as the undesirability of training on every collected data point, the aim is to obtain the most informative subset of our dataset. By maintaining this dataset, the model can on each training train also on the most informative samples that were collected previously. This is usually referred to as *rehearsal*, as the model learns on samples it has already seen (Verwimp et al., 2021). This subset has a limited size and has to be collected online (i.e. during the operation of the vehicle. It should contain points that allow us to train the best-performing model. This naturally leads us to ask two questions: "Which newly collected data point should we add to our storage and train our model on"? and "If we need to remove a data point, which one should we remove?" For this, we need to assess whether a point is valuable for the model. If we can determine whether the information contained in a point has already been learned, we can decide to not store and process the incoming point.

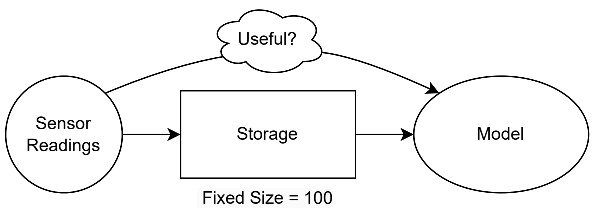


Figure 1.1: Problem addressed in this work. Gathered data contains redundancy. Restricting the size of the storage forces us to investigate whether information from collected data points was already learned and for which stored point we should substitute it.

This idea is visualized in Figure 1.1.

For these purposes, one can use the concept of uncertainty to assess the collected samples. We can equip our learner with an uncertainty quantification method to be able to quantify the uncertainty of the predictions (Valdenegro-Toro & Mori, 2022). In literature, two types of uncertainty are distinguished, *aleatoric*, which describes the inherent uncertainty of the collected data due to noise, and *epistemic*, which relates to the uncertainty of the model's prediction on the data (Hüllermeier & Waegeman, 2021). In regression settings, the uncertainty of a model can be interpreted as the variance of a model around a certain prediction mean, or as a confidence interval of the prediction.

In our task, knowledge of epistemic uncertainty can be exploited, as we can use it as a determining criterion for whether a point is informative for the model. Intuitively, uncertainty gives us useful information about the sample: the samples the learner is the most uncertain carry the most information for the learner to learn from. This is the key to the concept of *active learning*: models can achieve greater accuracy on smaller amounts of data if allowed to choose which data to train on (Settles, 2009). Hence, we can decide whether the sample is worth storing and training on. We can let that depend on the quantified uncertainty of the prediction on that point. Furthermore, by using uncertainty prediction on the currently stored samples we can decide which data to discard. We can choose to discard the least uncertain samples.

What is more, equipping the model with uncertainty quantification can be used for other purposes. It can be used to give meaning to the prediction of the output of the model in deployment. In control, for example, we can use the uncertainty on prediction output as an indicator of the confidence of the vehicle in the current manipulation scenario. Based on this, we can let the vehicle slow down if the model is uncertain, to approach less-know scenarios at lower speeds.

In this work, we would like to investigate the above-motivated application of uncertainty quantification in incremental online learning. In particular, we want to answer the question "Can uncertainty estimation be used to increase efficiency and stability in online incremental learning of regression tasks in settings with constrained resources?" In this context, by resources we mean the number of data points that the model has to hold in its memory and the number of times the model has to backpropagate gradients of those.

The contribution of this work lies in the investigation of the use of active learning in complex regression settings. If the approach turns out to be successful, the methods could potentially be used in reducing the computational and storage load in robotics applications where machine learning models are trained in an online fashion.

To answer our research question, we will compare the performance of models in an online incremental learning procedure. The models with uncertainty quantification capabilities will be compared to baseline models that select their training data randomly.

In this paper, we will first discuss the background Section 2. Then in Section 3, we will in detail introduce the online learning methods, both baselines and the approaches in which we will exploit uncertainty. In Section 4, we discuss the implementation details as well as the evaluation metrics of the models. The results are analyzed in Section 5, which are further analyzed in Section 6, based on which conclusions are drawn in Section 7.

2 Related Works

Learning models of dynamics and kinematics is certainly not a new task. The fact that learning a model from collected sensor data is more feasible than explicitly defining the model has been long recognized and heavily researched in many different scenarios. There have been successes in non-parametric learning, as well as semi-parametric approaches, which use non-parametric models informed by prior knowledge of the system Thuruthel et al. (2017). For supervised learning, labeled data is usually expensive, which motivated the investigation of active learning in this context (Cohn et al., 1996; Dasgupta, 2004).

Generally, there are two types of learning dynamics models. First is a global approach, which tries to approximate one function across the whole feature space. The popular choices for this include *gaussian regression process* (Williams & Rasmussen, 1995), *support vector machine regression* (Schölkopf et al., 2000), and *variational Bayes for mixture models* (Ghahramani & Beal, 1999). Secondly, in an incremental local approach, one can incrementally fit the data locally by the multitude of functions (Vijayakumar et al., 2005). In this work, we choose to focus on the global methods. The reason we do not use local approaches is that we cannot create an uncertainty measurement on such a model. Furthermore, global methods are state of the art in other areas of research, therefore our findings can be further applied to various other areas.

The paradigm of incremental online active learning has not yet been heavily explored in regression settings. It has been widely explored in the realm of classification, to tackle problems such as concept drift (Rožanec et al., 2022; Lu et al., 2016; Dasgupta et al., 2005; Sculley, 2007; Feng et al., 2016; He et al., 2020). In regression settings, it has been explored in a simple setting of linear regression (Chen et al., 2022). This work will expand this approach to a more complex regression task. Identification of a dynamics model of AUV is highly non-linear and many dimensional. This allows us to explore the applicability of these methods in more complex settings. Furthermore, the methods will be tested on real-world collected data, which will show how the methods perform with natural aleatoric certainty generated by the noise in the sensors.

We also must note the existence of the problem of updating the knowledge if the model gets outdated is usually referred to as *adaptive learning* (Loeffel, 2017). In that case, our estimated model could have low uncertainty about its prediction, yet it would still need to change. We decide to ignore this issue, as it is outside of the scope of this work.

3 Proposed Method

To answer the research questions, we need to implement uncertainty into the selection procedure of data in online learning and compare the performance of that model to a baseline model that does not use such a technique. If the augmented model proves to outperform the baseline model, then we can positively answer our research question.

Before we do that, we will first describe the specification of the learning task that we are tackling. Then we will in detail describe incremental online learning approaches: both baseline approaches as well as uncertainty-augmented methods.

3.1 Model Learning

The aim is to learn a dynamics model from collected data. We will do this by directly relating the inputs and the outputs, without explicitly establishing the equations, nor estimating the parameters of any presumed model. This work is an extension of Wehbe & Krell (2017), hence in this part as well as later, we will closely follow the derivations given in section II. and III. of the paper.

The position and orientation of vehicles in three-dimensional spaces can be described using 6 degrees of freedom. The forward direction is referred to as *surge*, the sideways direction is *sway*, and the upwards and downwards as *heave*. Additionally, to describe the rotation of the body in space, we denote *roll* the side-to-side direction of rotation, *pitch* up-down, and *yaw* the left-right direction of rotation. For learning the dynamics model, we are interested in the velocity in these degrees of freedom, and how they change relative to the given input commands. This means that we are interested in predicting the acceleration in the directions. Once we have those, we can later integrate the model to obtain the actual position in space based on the input sequence. To summarize, the models that we are going to consider in the upcoming sections learn to predict the acceleration in the considered directions given the velocities in those directions as well as the thruster activations.

3.2 Online Learning Methods

In this section, we describe used techniques for learning online incrementally from data. We will start by describing baseline approaches and then we will follow with the techniques that exploit uncertainty.

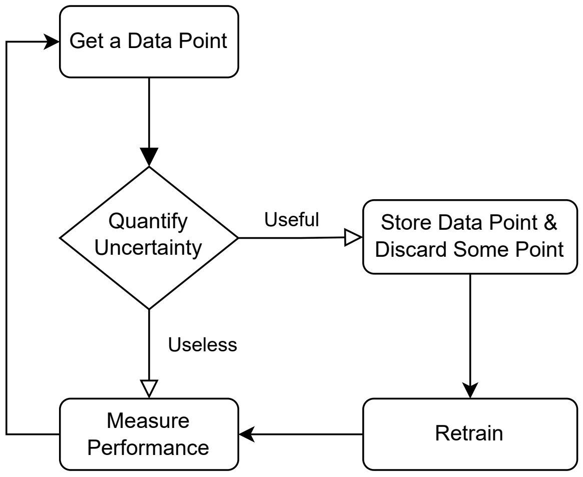


Figure 3.1: Diagram of applied methods. The online learning process is augmented with uncertainty quantification. Based on the estimation we can determine the usefulness of a point. If we consider it useful, we can choose to store it and train on it. Otherwise, we can skip it.

For each of the models, we assume it is used in the same process. The model gets new data points served one at a time and can learn from them and store them, or it can reject them. This is visualized in Figure 3.1.

All of the later described approaches are formally described in pseudocode in Appendix B and visualized in Appendix C.

3.2.1 Baseline Approaches

Offline: The best performance can be obtained by assuming that all data was already collected in the past and training on all at once. The performance of this model can be assumed to be the optimal one. We will use it to compare the absolute performance of the online models, to see how close they come to learning the optimal solution.

First-In-First-Out (FIFO): We will use the most recent data obtained in the learning process to estimate the model. In each iteration, the model will forget the data point it has seen the latest and add the newest collected point. We expect this method to perform poorly due to catastrophic forgetting, the model will forget previously learned information and it will prefer just learning the data captured in the current part of the dataset.

First-In-Random-Out (FIRO): To have the model maintain a more evenly spread distribution over the dataset, we consider instead discarding a random point from the currently stored set. We expect this model to perform better than FIFO since it will preserve some of its knowledge. However, it will not be able to fit as tightly onto newly presented data.

Random-In-Random-Out (RIRO): The online learning methods augmented with uncertainty aim to reject some points, thereby avoiding (re-)learning redundant information. To implement the same idea in a statistical baseline, this next approach chooses to learn and store an incoming point only with probability p (parameter to be tuned). This model should be more robust, as it will not overfit to locally present information as FIFO. Furthermore, it should obtain a more even distribution over the dataset over a longer timeframe. The lower the parameter p, the longer the model should take to obtain this distribution, but the better it should be in maintaining the distribution over a longer time.

3.2.2 Approaches with Uncertainty Quantification

In this section, we will consider a model which is augmented with uncertainty quantification capability. For each point, the model can run to determine the uncertainty of the model's prediction. This will be exploited to decide whether to include the point in the dataset or not. The model will be retrained each time that a new point is added to the dataset.

Greedy: This method greedily selects the most interesting points to learn from, that is the points that are the most uncertain. When the incoming point is assessed for uncertainty, so will the points stored in the currently preserved dataset. If the incoming point has higher uncertainty than any point in the current set, it will be substituted for this point, before the model is retrained.

Threshold: To avoid re-learning redundant information, this model chooses to avoid points whose uncertainty is low. If the model is certain on a point, there is no need to include it in its data set. The converse is also true if the new point has uncertainty above a certain bound. In practice, we choose a bound t (hyperparameter to be determined). If t is chosen too high, the model will ignore data that is uncertain; if it is too low it will learn on (almost) all points and will present no advantage over FIFO. The point chosen to be discarded from the currently held dataset is selected randomly.

Threshold-Greedy: This approach aims to combine the two previously presented techniques. Each point will be first compared against the uncertainty threshold t (parameter to be tuned). Subsequently, if it needs to be inserted into the dataset, the least informative sample of the dataset will be discarded. This should present a little advantage over the previous Threshold approach, since at each step, the uncertainty in the training set is maximized.

3.3 Uncertainty in Learning

The motivation of this work is based on the presumption, that the most informative samples for the model are those, which are uncertain. Therefore, we expect that the methods that utilize uncertainty to select the samples will outperform the baseline methods. Furthermore, to train a good model, it might be sufficient to sample the space of data points evenly. However, some parts of the feature space are more difficult to learn (are highly non-linear). Those might benefit from having more points sampled there. This could be even better captured by the uncertainty, as the model will remain more uncertain in those regions.

4 Experimental Setup

In this section, we will first discuss the contents of the data set. Then we will discuss the evaluation criteria of proposed methods. We will look in further detail at the machine learning techniques that we will use, including uncertainty quantification. Lastly, we will lay out the organization of the experiment that we are going to perform.

4.1 Dataset

4.1.1 The Dagon Vehicle

The data that we will be using was collected by the autonomous underwater vehicle Dagon, which can be seen in Figure 4.1 and 4.2. Dagon is a vehicle

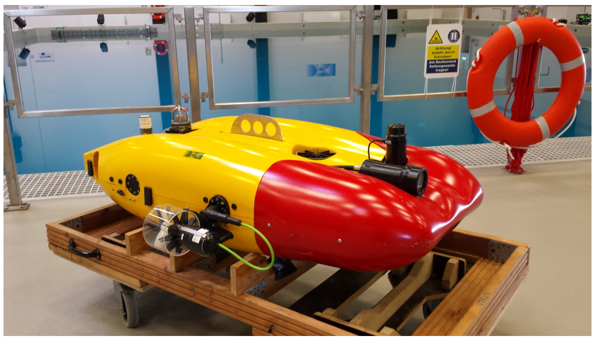


Figure 4.1: The AUV Dagon in a stationary position. Photo from Bande & Wehbe (2021).

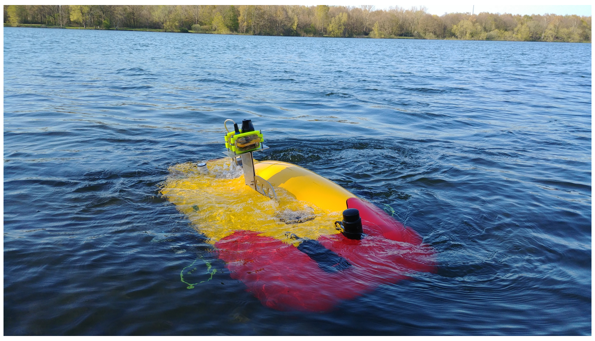


Figure 4.2: The AUV Dagon during operation. Photo courtesy of Bilal Wehbe.

specifically designed for scientific testing and evaluation of algorithms. Its dimensions are 70x60x30cm and it weighs around 50kg. It is designed to withstand diving into depths of 150m. It has 5 thrusters, which can be used to maneuver the vehicle, and can be equipped with a multitude of sensors. Detailed information can be found online in (Hildebrandt & Hilljegerdes, 2010).

4.1.2 Collected Dataset

The dataset was kindly provided by *DFKI**. The Dagon AUV vehicle was driven in the salty water basin of the research center of DFKI. The vehicle was stabilized to drive in the horizontal plane. Therefore, only 3 out of 5 of the thrusters were used to manipulate the vehicle. This means that for our purposes of learning, we will consider only the directions of surge, sway, and yaw and only the data of the relevant thrusters. To obtain all the interactions of the model's dynamics and the thrusters' activations, the thrusters were given sinusoidal input with different periods. During the experiment, the linear and angular velocities of the vehicle were captured. This data was numerically differentiated to obtain acceleration. For further details on how the data was captured refer to Wehbe & Krell (2017).

*Deutsches Forschungszentrum für Künstliche Intelligenz – German Research Center for Artificial Intelligence, see aknowledgements

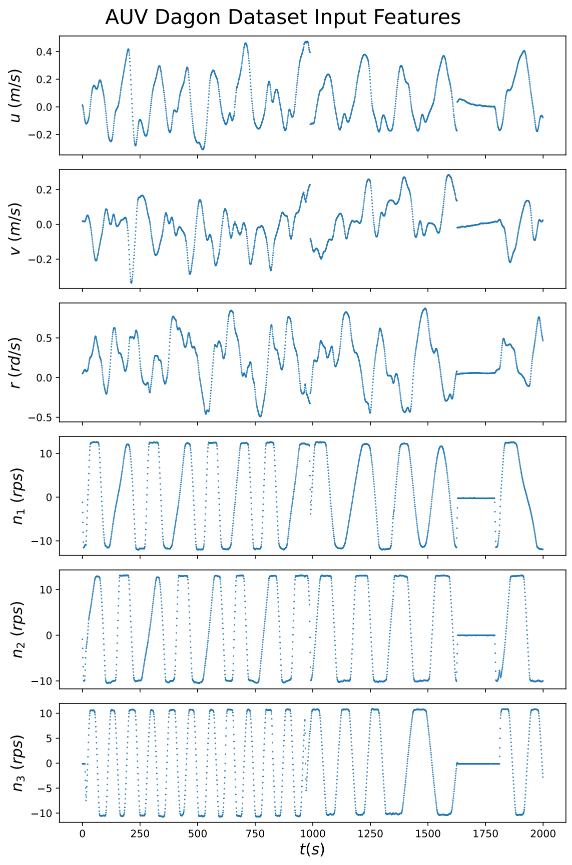


Figure 4.3: Input features of the Dagon dataset. The model was fixed to the horizontal plane, therefore inputs contain surge, sway, and yaw (u, v, r) and the three relevant thrusters (n_1, n_2, n_3). The thrusters were activated with sinusoidal inputs of different frequencies.

The final dataset contains 11566 data points in total. The first 2000 input features, can be seen in Figure 4.3, while the corresponding targets are shown in Figure 4.4.

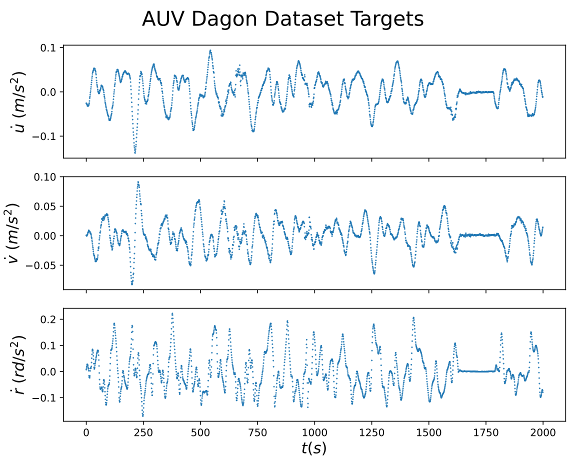


Figure 4.4: Targets of the Dagon dataset. To capture the dynamics model of the vehicle, it is sufficient to learn the acceleration in the desired directions, which are surge, sway, and yaw (u, v, r).

4.1.3 Approximated Function

Following the notation from (Wehbe & Krell, 2017), we can denote the first derivatives of the surge, sway, heave, roll, pitch, and yaw as $\nu = [u\ v\ w\ p\ q\ r] \in \mathbb{R}^6$ and the inputs of the 5 thrusters as $n = [n_1\ n_2\ n_3\ n_4\ n_5] \in \mathbb{R}^5$. Therefore the dynamics of the robot are given by function $\mathcal{F}$:

$$\dot{\nu} = \mathcal{F}(\nu, n) \tag{4.1}$$

As noted before, the robot's motion in the collected dataset was restricted to a horizontal plane, hence the only relevant directions of motion are surge, sway, and yaw. Furthermore, as only 3 thrusters of the robot were actively used, we can leave the other two out of the equation. This setup is visualized in Figure 4.5. Hence, the function we are learning is $\mathcal{F}'$:

$$(\dot{u}, \dot{v}, \dot{r}) = \mathcal{F}'(u, v, r, n_1, n_2, n_3) \tag{4.2}$$

4.1.4 Dataset Split

For tuning the parameters, a validation set was withheld from the dataset. To test the performance of the model a test set was also set apart. As the data from the robot was collected over time, the test set was sampled at evenly spaced intervals throughout the set. The dataset split follows the ratio 60/20/20% for training, validation, and testing sets.

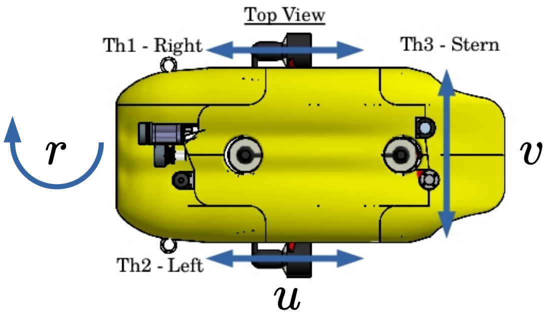


Figure 4.5: The Dagon AUV from the top view and its degrees of freedom and related thrusters. The vehicle was fixed in the horizontal plane. It was controlled in surge, sway, and yaw (u, v, r) and steered by 3 thrusters (n_1, n_2, n_3). Figure adapted from Wehbe & Krell (2017).

4.2 Metrics

In this section, we will discuss the evaluation criteria of the learning process. We will propose metrics to evaluate the performance of the model both at the end of the learning process as well as quantifying its performance over the learning process.

4.2.1 Instantaneous Model Performance

At each point of the learning process, we would like the machine learning model to be as accurate as possible in the testing set. We will use standard approaches for this.

To precisely describe the two metrics, we will first introduce the necessary notation. Denote (X_i, y_i) the pair of inputs and targets of the testing set of size n. Then $\hat{y}_i$ is the prediction of the model on the input X_i. Furthermore, denote $\bar{y} = \frac{1}{N}\sum_{i=1}^{n} y_i$ the mean of the target variables.

The first used metric is *Mean Squared Error* normalized by the number of samples. This metric measures how far on average the prediction is from the ground truth. Furthermore, introducing the second power penalizes the model for being very wrong on some samples. Hence the lower the MSE, the better.

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{4.3}$$

The second metric will be the *Coefficient of Determination*, also known as R^2. It measures the amount of variance in the output of the model related to the model by dividing it by the sum of differences of the corrected variables from the mean (non-normalized variance).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.4}$$

The closer the R^2 is to the value 1, the better. We note that for multivariable regression, the R^2 cannot clearly be defined, as each y_i becomes a vector. Therefore we will consider the mean of R^2 scores.

4.2.2 Performance Over Learning Process

The ideal model converges fast to a solution and remains stable over the whole process of online learning. To capture this in a numerical metric, we use inspiration from (Zinkevich, 2003) and look at the cumulative loss of the model over the learning process. The models that converge fast to a small loss, and remain close to it over the learning process should obtain low cumulative loss at the end of the learning process as well. This can be applied to MSE, however, not to R^2 as negative and positive values would cancel out. It is important to note, that summarizing the information contained in MSE is not lossless. An unstable method might produce the same cumulative MSE as a method that converges slowly. Therefore, the conclusions will have to be drawn by inspecting the development of the MSE over time.

Moreover, a good model should be gradually less uncertain about incoming points as it should learn the whole model and new information should not come in as a surprise. Therefore, we will collect the predicted uncertainty of each incoming point, and we can look at its development over the learning process.

4.2.3 Saved Resources

The motivation for the whole concept of using uncertainty in incremental online learning sparks from the will to increase efficiency. As per usual in machine learning models, forward passes are less expensive than backpropagation. Therefore, our uncertainty-equipped methods aim to save resources by avoiding unnecessary backpropagation, by avoiding training on some points. Therefore, one of the metrics we will look at is the number of skipped points in the training in the sense described in Figure 3.1.

4.3 Machine Learning Techniques

4.3.1 Uncertainty Quantification

For our tasks, we need to estimate epistemic uncertainty, which represents how uncertain the model is on the input data. There are many different uncertainty estimation techniques, however, to estimate epistemic uncertainty, ensembles perform the best (Valdenegro-Toro & Mori, 2022; Valdenegro-Toro, 2021). This method of estimation of uncertainty is very stable. It is also very practical, as it is easy to parallelize in practice (Lakshminarayanan et al., 2017).

4.3.2 Simple Ensemble

The method of ensembles relies on the simple, yet powerful idea of combining multiple models of the same kind to obtain the uncertainty on the output.

Formally, we are aiming to create a regressor, which will be represented by a neural network. We define the dataset $\mathcal{D} = \{(\mathbf{X}_n, y_n)\}_{n=1}^{N}$, with features vectors $\mathbf{X}_n$ and targets y_n. The ensemble M will be composed of models m_i with learnable parameters which are summarized as $\Theta_i \in \mathbb{R}^K$. Each of these sets will be initialized independently, to ensure statistical independence of the ensemble. We define a loss function $\mathcal{L}(\mathbf{x}, y, \Theta_i)$, in our case, using the MSE loss. Hence $\mathcal{L}(\mathbf{x}, y, \Theta_i) = (f_{\Theta_i}(\mathbf{x}) - y)^2$, which is be calculated independently for each of the models m_i. Let us denote all the sets Θ_i as Θ.

Now we define the desired parameters $\hat{\Theta}$, which can be trained using gradient descent using the following rule, once again for each of the models independently.

$$\hat{\Theta}_i \in \underset{\Theta_i \in \mathbb{R}^K}{\arg\min}\, \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}}[\mathcal{L}(\mathbf{x}, y, \Theta_i)] \tag{4.5}$$

To obtain the predictive output of the model on the data point $\mathbf{x}$, denoted $f_\Theta(\mathbf{x})$, we can look at the mean output of the models.

$$f_\Theta(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} f_{\Theta_i}(\mathbf{x}) \tag{4.6}$$

Furthermore, the standard deviation of the outputs of the ensemble on the point $\mathbf{x}$, denoted as $u_\Theta(\mathbf{x})$ can be interpreted as uncertainty.

$$u_\Theta(\mathbf{x}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f_{\Theta_i}(\mathbf{x}) - f_\Theta(\mathbf{x})\right)^2} \tag{4.7}$$

It is noteworthy, that basic ensembles are different from the so-called deep ensemble. The deep ensemble uses models that all have two heads, one that predicts mean and one that predicts variance. This allows us to estimate both aleatoric and epistemic uncertainty. The head that predicts the mean, is supervised by the label of the data and the variance with the negative-log-likelihood Lakshminarayanan et al. (2017). However, as our goal is to only estimate epistemic uncertainty, the simple ensemble is sufficient.

4.3.3 Learning Process

To run the experiments, we used a simple ensemble with 10 estimators. The storage size of the buffer is set to 100. The model can learn for at most 100 epochs at each iteration. Each strategy can use its tuned patience parameter for early stopping. The Adam optimizer was used (Kingma & Ba, 2014). The number of layers, the units in each layer, the learning rate, batch size, and patience were tuned independently for each method. For details in hyperparameter tuning refer to Appendix A.

4.4 Technical Implementation

The experiments were implemented in Python 3 (Van Rossum & Drake, 2009) in the machine learning framework Keras (Chollet et al., 2015) the repository is available here. To estimate uncertainty, the library Keras Uncertainty was used. It provides all necessary utilities for estimating uncertainty on multi-layer perceptron. To perform the resource-demanding experiments, the high-performance cluster Habrók of the University of Groningen was used.

4.5 Experiment Plan

We decided to tune the machine-learning parameters and method-specific parameters separately, as we want to investigate the influence of the method-specific parameters on the behavior of the network. Firstly, we will fix the parameters p in RIRO and t in Threshold and Threshold-Greedy to an experimental value: $p = 0.7$ and $t = 0.02$. We will perform hyperparameter tuning of all machine learning parameters and we will look at the performance of the models. Then we will separately investigate the influence of t and p on the performance of the model. Afterwards, we will pick the most optimal parameters for each method and we will compare the models once again, to reach a better conclusion of the comparative performance. Lastly, whilst keeping the optimal parameters discovered before, we will investigate the influence of the storage size of the models on their performance.

5 Results

To find whether uncertainty-based selection can help us decrease the size of the dataset stored in the buffer, we compared the performance of different models on this task. In this section, we follow the plan set out in Section 4.5. We will first investigate the performance of the different models, with hyperparameters fitted according to the procedure specified in Appendix A. Some of the parameters related to the processes in the online learning models were fixed with arbitrary values. The influence of those will be investigated afterward. Subsequently, we will present a final comparison of results with optimal parameters. Lastly, we will investigate and discuss the role of the size of the buffer on the performance of the models, assuming they all use optimal parameters.

5.1 Basic Hyperparameter Tuning

The results of the performance of the difference models on the testing set are visible in Figure 5.1. For a graph of each metric in detail please refer to Appendix D. On the shared x-axis, we can see the iterations over the training set, and on the y-axis, we can see the metric value, which is calculated over the testing test at each iteration. Each model has its assigned color, which is the same on

Model Name	Points Used	Dataset Usage %
RIRO	4786	70.0%
Greedy	6661	97.4%
Threshold	1357	19.8%
Threshold-Greedy	1172	17.1%

Table 5.1: Resources used by methods after basic hyperparameter tuning. The online learning methods can skip storing and training on points, which makes them save resources. This table shows for each method how many of the points have been used over the learning process over the whole training set of 6840 points.

all plots. The results of MSE and R^2 were truncated at the start of the graph to remove outliers at the beginning of the learning process where the model performs poorly. The bad performance skewed the graphs and made them harder to interpret.

To assess the consumed resources, the number of skipped points in the training set was recorded. The training set consisted of 6840 points and the final number of skipped points for each method that was able to skip points are visible in Table 5.1. The percentages are rounded to one decimal digit after the dot.

5.2 RIRO p Study

Next, we investigated the effect of the parameter p in RIRO. The parameter p represents the probability of accepting a point into the dataset. We evenly sampled the space of possibilities, taking p from the set $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. While noting that $p = 1$, makes the method RIRO the same as method FIFO, and $p = 0$ would accept no points at all. We performed the same experiment as for all previous methods, with varying p.

As our decision metric for the selection of the best parameters in Hyperparameter Tuning (see Appendix A) was cumulative MSE, we will also use this here to assess the parameters. The cumulative MSE over the test set of all the models as a function of p can be seen in Figure 5.2. More detailed results for the R^2 and cumulative MSE are shown in Figure D.2.

Percentile of Uncertainty	Value of t
10%	0.0016
20%	0.0023
30%	0.0036
40%	0.0056
50%	0.0075
60%	0.0096
70%	0.0120
80%	0.0156
90%	0.0228

Table 5.2: Values of t considered for the thresholding methods. The values of t we determined according to the percentile division of uncertainty prediction obtained by baseline methods FIFO and FIRO.

5.3 Threshold t Study

To sample the parameter t for the thresholding methods, we divided uncertainty into percentiles. This was done by using the methods that use all points from the training set, which are FIFO and FIRO. We recorded the uncertainty prediction of each point of the set. Using that, we can divide up the points in the training set into uncertainty level percentile. We did this for both of the methods and took the mean of the percentiles. The values we obtain are visible in Table 5.2. For each of these values of t we performed the experiments once again.

5.3.1 Threshold with Varying t Performance

The cumulative MSE over the test set of all the models depending on the parameter t can be seen in Figure 5.3. More detailed results for the R^2 and cumulative MSE are shown in Figure D.3.

5.3.2 Threshold-Greedy with Varying t Performance

The cumulative MSE over the test set of all the models depending on the parameter t can be seen in Figure 5.4. More detailed results for the R^2 and cumulative MSE are shown in Figure D.4.

5.4 Optimal Parameters

Finally, we can compare the performance of all the models with optimal values of parameters. The selected value of p for RIRO was 0.2, the value of t for

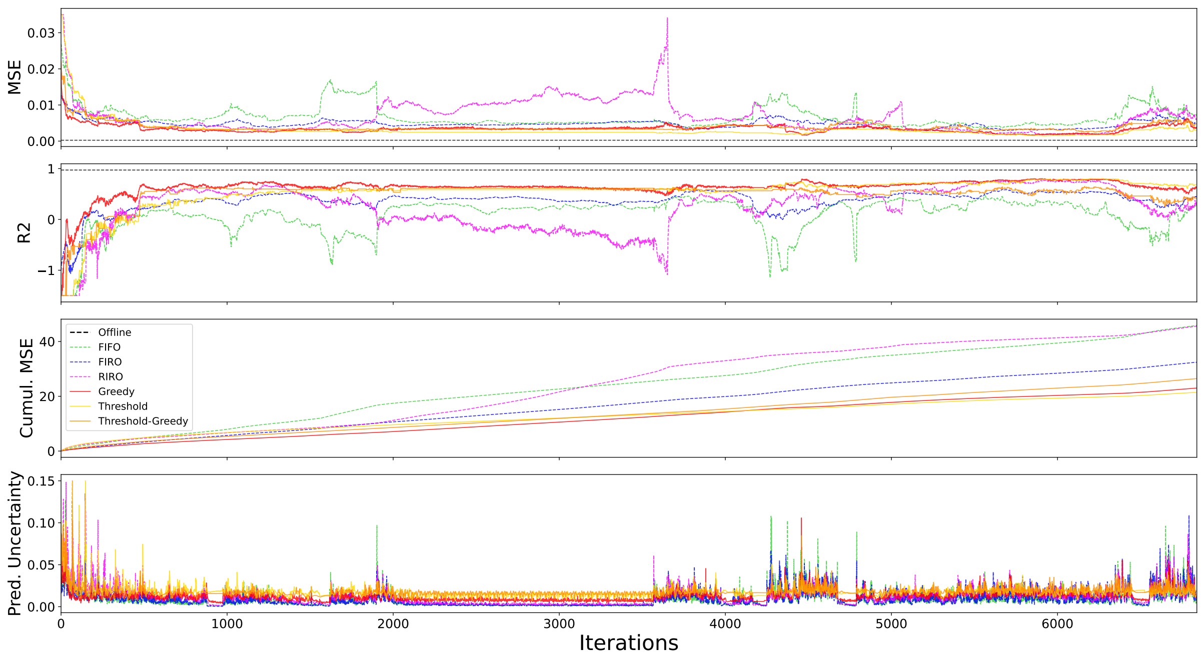


Figure 5.1: Comparison of models after basic hyperparameter tuning. The x-axis denotes the iterations over the training set, and the y-axis the performance of respective metrics on the testing set in each moment. The baseline models are denoted by dotted lines and uncertainty models by full lines. The graphs of MSE, R^2, and predicted uncertainty were truncated to make the graphs more interpretable.

Threshold was 0.0156, and t for Threshold-Greedy was 0.0228. The comparison of the performance over the training set with the metrics recorded over the testing set is shown in Figure 5.5. For a better display of all metrics in detail, please consult Appendix D.

To assess the consumed resources, we once again present the number of skipped points. The training set consisted of 6840 points and the final number of skipped points for each method that was able to skip points are visible in Table 5.3. The percentages are rounded to one decimal digit after the dot.

Model Name	Points Used	Dataset Usage %
RIRO	1397	20.5%
Greedy	6661	97.4%
Threshold	1948	28.5%
Threshold-Greedy	636	9.3%

Table 5.3: Resources used by methods after method-specific parameters have been tuned. The online learning methods can skip storing and training on points, which makes them save resources. This table shows for each method how many of the points have been used over the learning process over the whole training set of 6840 points.

5.5 Buffer Saturation

Lastly, we look into the performance of the methods with a varying size of a buffer. For all of the previous experiments, the buffer size was set to 100 for all of the methods. To produce these results, both optimal machine learning and method-specific parameters were used, that is the same set of parameters which was used to produce the results shown in Figure 5.5.

The buffer size we choose to investigate is from the set $\{10, 20, 50, 100, 200, 400\}$. We look at the cumulative MSE achieved by each of the methods over the learning process given the size of the buffer. The results are presented in Figure 5.6. For a better display of the results, see Appendix D.

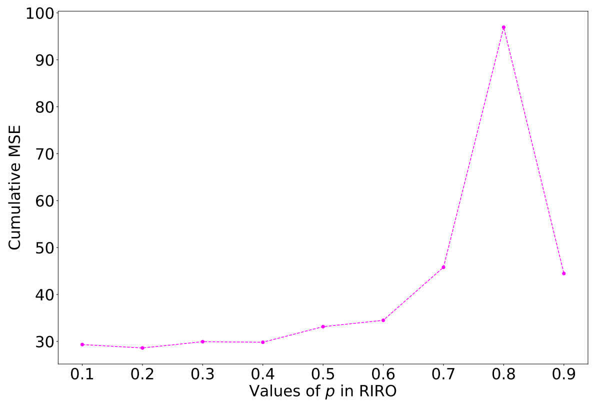


Figure 5.2: Final cumulative MSE of RIRO models with varying parameter p. The parameter p in RIRO regulates the probability of accepting the point into the training set. The graph shows the final cumulative MSE after the learning process as a function of the parameter p.

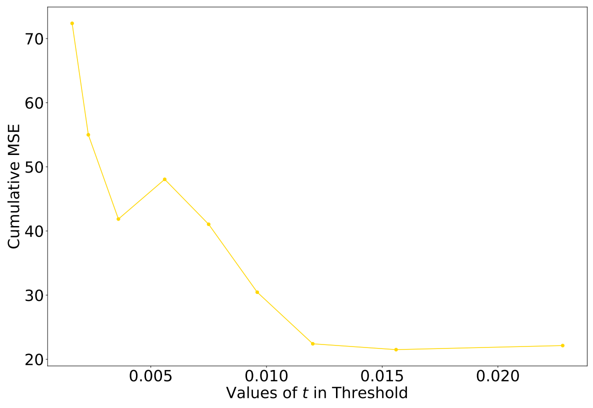


Figure 5.3: Final cumulative MSE of Threshold models with varying parameter t. The parameter t in Threshold determines the necessary level of uncertainty that a point has to reach to be included in the dataset. The graph shows the final cumulative MSE after the learning process as a function of parameter t.

6 Discussion

The aim of this paper was to determine, whether the amount of data can be retained in the online learning process without a loss of performance using uncertainty. We will now discuss the results obtained in Section 5.

6.1 Basic Hyperparameter Tuning

Firstly, we take a look at the results presented in Figure 5.1. Looking at the MSE, we can see that the statistical baselines (FIFO, FIRO, and RIRO) do not obtain a stable model over the whole learning process. The MSE oscillates, which indicates that these methods locally overfit the current data and do not manage to generalize over the whole learning process. What is more, the methods suffer from catastrophic forgetting constantly over the whole learning process. The same picture is sketched by the R^2 metric, where none of the three previously mentioned techniques manage to remain stable over the whole learning process. This projects into the cumulative MSE metric, which tells us that overall FIFO performed the worst, with RIRO being a close contestant. FIRO performs quite a bit better, but still worse than the uncertainty-based methods.

The uncertainty-based methods all manage to regularize for local overfitting on the current data. This can be deduced from the stability of the learning process, as the curves of MSE for Threshold, Greedy, and Threshold-Greedy all stay low during the whole learning process. We can note that Greedy converges the most quickly at the beginning of the training. However, later it is less stable. The Threshold method converges a lot slower at the beginning, with Threshold-Greedy performing somewhere in between the two previously mentioned. However, later in the process, the Threshold method is the most stable and keeps to a good model, while other methods oscillate a lot more. The R^2 once again sketches a similar picture of this. All of these effects are captured in the cumulative MSE, which sets the Threshold to be the best-performing model overall, followed in close succession by Greedy and Threshold-Greedy.

It is also noteworthy to inspect the graph of prediction uncertainty over the learning process. We can see that the statistical baseline methods predict lower uncertainty most of the time, with high peaks from time to time. This can be interpreted as overfitting the local data, as when very novel data appears, the uncertainty jumps up to be high before the model is fitted onto this new data. In contrast, the uncertainty-based models, predict somewhat higher uncertainty over most of the points, however, they do not have such high peaks, which indicates that they manage to generalize and novel

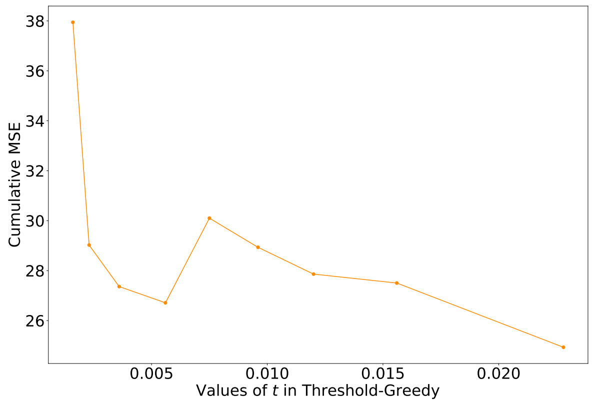


Figure 5.4: Final cumulative MSE of Threshold-Greedy models with varying parameter t. The parameter t in Threshold determines the necessary level of uncertainty that a point has to reach to be included in the dataset. The graph shows the final cumulative MSE after the learning process as a function of the parameter t.

data is only a slightly bit uncertain when it comes to the input.

Lastly, Table 5.1 displays proof that the uncertainty-augmented methods manage to save resources. Notably, the Threshold method has performed the best overall on cumulative MSE, while using only 17.1% of the dataset. From this, we can conclude that uncertainty can be used as a valuable predictor of whether or not a data point should be selected into a dataset, especially in the scenario when storage is scarce and data is expected to have redundancy.

6.2 Additional Parameter Tuning

The performance of RIRO models with different parameters p was compared based on the final cumulative MSE in Figure 5.2. From the results, we can conclude that lower p is more beneficial to final cumulative loss, but as Figure D.2 hints, these models take a lot more time to converge. This exactly follows what we predicted in Section 3. Lower p means that we need a longer time to establish an even distribution of the data over the whole sample space, but archives an even distribution later on. Overall, the value $p = 0.2$ gave us the best performance, hence it was also used later on in other experiments.

Similarly to this finding, Figure 5.3 and Figure 5.4 suggest that higher thresholds guide models to higher performance on the final cumulative MSE. Curiously, for Threshold methods, the difference at the beginning of the learning process is not that stark as shown in Figure D.3. We hypothesize, that this is because, at the beginning of the learning process, all the data has high uncertainty, as it is not well known. Curiously, this is not true for Threshold-Greedy, as shown in D.4, where all the methods perform similarly over the whole learning process. In the end, however, a higher value of t was indeed also beneficial. All in all, the value $t = 0.0228$ brought the best results for Threshold-Greedy and $t = 0.0156$ for the Threshold methods. These were selected for use in the later experiments.

6.3 Optimal Parameters

The results with the optimally tuned p and t are shown in Figure 5.5. All of the same observations as previously discussed with the first results apply to these results now. However, RIRO with tuned p now achieves a lot lower final cumulative MSE, which makes it perform better than FIRO and brings it close to the performance of uncertainty-based methods. The graph of cumulative MSE, however, does not tell the full story in this case. From the development of R^2 and MSE, we can see that by the end of the learning process, the RIRO model has performed better than the Threshold models, achieving lower MSE and higher R^2. We note that over a long period, taking just random points introduces sufficient robustness against local overfitting. However, RIRO was not as stable as uncertainty-based methods during the whole learning process. It could not prevent a sharp decrease in performance in the middle of the learning process. From this we can conclude, that uncertainty is a good selection metric for picking the points, which can help us increase the robustness of the model, however, over long horizons, it will perform no better than random selection.

It is interesting to consider the performance of all the models in comparison with the Offline baseline, which has considered all the points at once. The performance of the baseline has been plotted as a line for MSE and R^2 over the whole process so that we can compare the optimal model we are striving towards. We can conclude that no one of the models

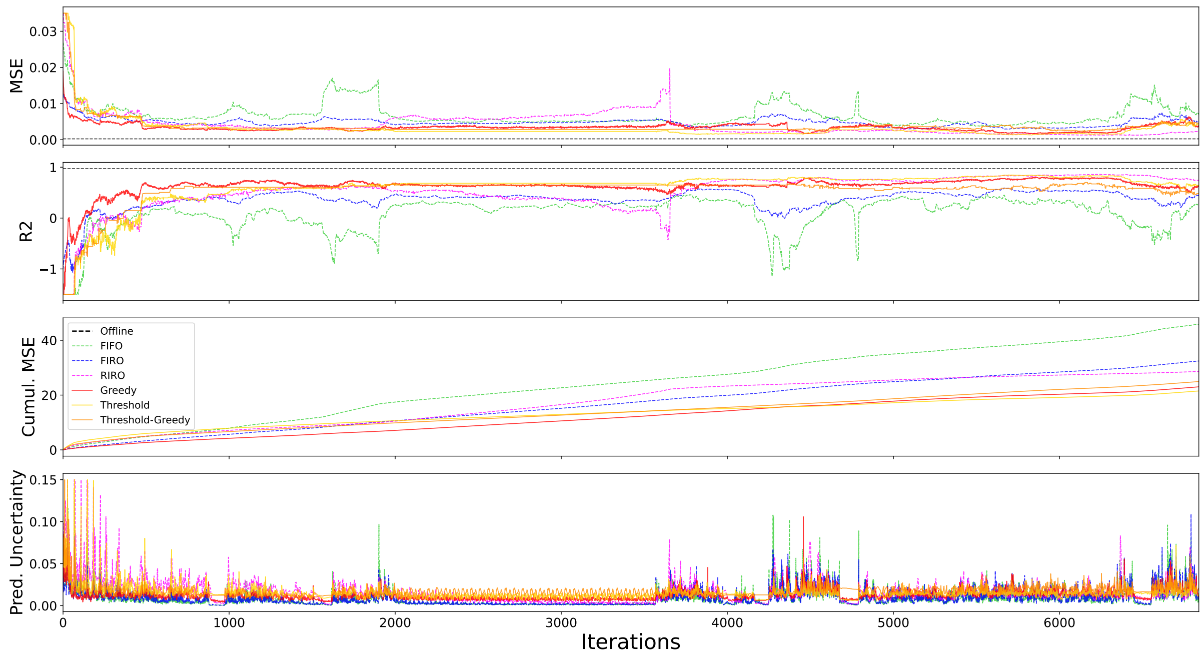


Figure 5.5: Final comparison of models. The x-axis denotes the iterations over the training set, and the y-axis the performance of respective metrics on the testing set in each moment. The baseline models are denoted by dotted lines and uncertainty models by full lines. The graphs of MSE, R^2, and predicted uncertainty were truncated to make the graphs more interpretable.

came close to achieving the same MSE and R^2 as the baseline. RIRO was closest at the end, achieving MSE of 0.001236 and R^2 of 0.859239, however, the Offline baseline obtained MSE equal to 0.000237 and R^2 of 0.971106. From this, we can conclude, that the buffer of size 100 is simply not big enough to contain sufficient data for learning models that can achieve a low loss.

Another proof that uncertainty is a good indicator of which points to skip is the analysis of skipped points in Figure D.5e. As established in the previous paragraph, RIRO with small p achieves the best performance, and we can see that it has been skipping a similar quantity of points, as the uncertainty-based methods. This indicates, that the uncertainty can help us select the quantity of points to be skipped, to increase the robustness of the learning approach.

6.4 Buffer Size Influence

As hypothesized, the performance in terms of cumulative loss changes with the size of the buffer, as shown in Figure 5.6. For all of the models, the bigger the buffer, the smaller the final cumulative loss. However, the gain related to the increase in the size of the buffer is not the same for all the methods. All in all, the uncertainty-based methods perform poorly with a small buffer size, while for the bigger buffer sizes, the performance is getting better and better. The baseline methods, on the other hand, show a more stable performance regardless of the size of the buffer. For all of the methods, the gain in performance is indeed diminishing with the increased buffer size.

7 Conclusion

The question that we tried to answer during this research was: *Can uncertainty estimation be used to increase efficiency and stability in online incremental learning of regression tasks in settings with constrained resources?* We can answer affirmatively: as discussed above, the uncertainty-based methods exhibit greater stability over the learning process and

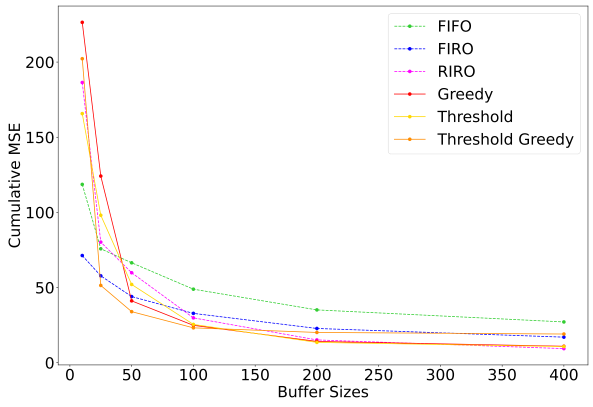


Figure 5.6: Cumulative MSE of each of the methods depending on the size of the buffer. The size of the buffer influences the maximum number of points each of the methods can store at any point during the online learning process. This has an influence on the quality of the model as the more points the model can store, the better the stored points represent the whole distribution of the data. The graph shows the final cumulative MSE after the learning process as a function of the size of the storage buffer.

consume fewer data points than statistical baselines.

All in all, the Threshold method has performed the most stable and delivered the best model in the end. As noted in Table 5.3, the method has used 28.5% of the data. The method that required the least data was the Threshold-Greedy method (9.3%). Although this might seem like a great number of saved resources, in the process of estimation uncertainty, 10 estimators were used and all of them had to be trained independently. This effectively multiplies the amount of used resources by 10. Therefore the going in saved resources is not that big. All in all, we conclude that there is no free lunch (Adam et al., 2019).

7.1 Potential Improvements

There are potential flaws in the study, which could be improved upon. Firstly, one might point out that the statistical baselines, such as FIFO, FIRO, and RIRO, are only regularized by early stopping. It is therefore unexpected, that they did not generalize. In the end, the best-performing methods are the ones that skip a lot of points, preventing (or at least delaying) overfitting. Although the hyperparameter tuning gave each of the methods the possibility of using a smaller network, it could be useful to try tuning $L1$ or $L2$ regularization, or other different regularization techniques.

Moreover, one might argue that fitting the parameters separately in two rounds (firstly the basic machine learning parameters, then the other parameters of the methods) is not fully fair. The parameter for the Threshold that was guessed for the initial hyperparameter tuning was closer to the optimal one than the one chosen for RIRO. This could have negatively influenced the final performance of the RIRO method. It would be optimal, to tune all the parameters at once, refine the search space, and give the Bayesian Optimizer more iterations.

Next up, one could question the techniques used for the estimation of uncertainty. Estimation of epistemic uncertainty has its limits, there are many techniques to do it, which have different accuracy (Valdenegro-Toro & Mori, 2022). It would be best to confirm the results with another uncertainty estimation technique.

7.2 Suggested Research Directions

Although we have performed an extensive comparison of methods. It would be useful to compare the results also with some spatial and temporal heuristic approaches, similar to the ones presented in Wehbe et al. (2017). Achieving equally-spaced spatial distribution of points in the dataset might incur smaller computational costs while inducing the great performance of the model. It would be useful to see, how the uncertainty-based methods would compare against such approaches.

As hinted in the discussion of the results with varying buffer sizes, the increase in buffer size brings diminishing returns. One might consider investigating further, the effect of the buffer size and see how the returns diminish till finding a buffer size that performs indistinguishable from the Offline baseline. We hypothesize that in the case of our experiment, this might be around 1000 points.

Furthermore, more investigation should be done into how information is retained in the network throughout the learning process. From this work, there is no direct insight into how much the networks suffer from catastrophic forgetting and how much they can bring with them over the online learning process. Having a better insight into the mechanics of this might help us leverage information gathered from uncertainty estimation, or any other heuristic for that matter, even better.

Although we have concluded that the threshold method performed the best, the performance of the methods is closely tied to the parameter t. This means, that the parameter must be tuned up front, which might not be very useful in practice. We propose to investigate an adaptive approach, for example, developing the idea of the percentiles based on uncertainty predicted on previously encountered points. Let the model estimate the uncertainties and only accept those in the 90% percentile of uncertainty. Such a technique would remove independence from the tuning procedure.

All in all, the ultimate proof of the usefulness of the suggested methods will be the deployment of the methods in the real environment on an autonomous underwater vehicle. This is the next suggested step towards seeing how the method could be exploited in practice.

8 Acknowledgements

Big thanks to Bilal Wehbe and the *Robotics Innovation Center of Deutsches Forschungszentrum für Künstliche Intelligenz* (DFKI RIC, Germany, 28359 Bremen. Department of Mathematics and Informatics, University of Bremen) for the opportunity to collaborate on this scientific discovery

We thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high-performance computing cluster.

Thanks to RNBStylerz for an amazing playlist that has kept me sane throughout the writing process.

References

Adam, S. P., Alexandropoulos, S.-A. N., Pardalos, P. M., & Vrahatis, M. N. (2019). No free lunch theorem: A review. *Approximation and optimization: Algorithms, complexity and applications*, 57–82.

Bande, M., & Wehbe, B. (2021). Online model adaptation of autonomous underwater vehicles with lstm networks. In *Oceans 2021: San diego–porto* (pp. 1–6).

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Chen, C., Li, Y., & Sun, Y. (2022). Online active regression. In *International conference on machine learning* (pp. 3320–3335).

Chollet, F., et al. (2015). *Keras.* `https://keras.io`.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of artificial intelligence research*, *4*, 129–145.

Dasgupta, S. (2004). Analysis of a greedy active learning strategy. *Advances in neural information processing systems*, *17*.

Dasgupta, S., Kalai, A. T., & Monteleoni, C. (2005). Analysis of perceptron-based active learning. In *International conference on computational learning theory* (pp. 249–263).

Feng, L., Wang, Y., & Zuo, W. (2016). Quick online spam classification method based on active and incremental learning. *Journal of Intelligent & Fuzzy Systems*, *30*(1), 17–27.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, *3*(4), 128–135.

Ghahramani, Z., & Beal, M. (1999). Variational inference for bayesian mixtures of factor analysers. *Advances in neural information processing systems*, *12*.

He, J., Mao, R., Shao, Z., & Zhu, F. (2020). Incremental learning in online scenario. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 13926–13935).

Henson, M. A. (1998). Nonlinear model predictive control: current status and future directions. *Computers & Chemical Engineering*, *23*(2), 187–202.

Hildebrandt, M., & Hilljegerdes, J. (2010). Design of a versatile auv for high precision visual mapping and algorithm evaluation. In *2010 ieee/oes autonomous underwater vehicles* (pp. 1–6).

Hüllermeier, E., & Waegeman, W. (2021). Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, *110*, 457–506.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, *30*.

Loeffel, P.-X. (2017). *Adaptive machine learning algorithms for data streams subject to concept drifts* (Unpublished doctoral dissertation). Université Pierre et Marie Curie-Paris VI.

Lu, J., Zhao, P., & Hoi, S. C. (2016). Online passive-aggressive active learning. *Machine Learning*, *103*, 141–183.

Part, J. L., & Lemon, O. (2017). Incremental online learning of objects for robots operating in real environments. In *2017 joint ieee international conference on development and learning and epigenetic robotics (icdl-epirob)* (p. 304-310). doi: 10.1109/DEVLRN.2017.8329822

Rožanec, J. M., Trajkova, E., Dam, P., Fortuna, B., & Mladenić, D. (2022). Streaming machine learning and online active learning for automated visual inspection. *IFAC-PapersOnLine*, *55*(2), 277-282. doi: https://doi.org/10.1016/j.ifacol.2022.04.206

Schölkopf, B., Smola, A. J., Williamson, R. C., & Bartlett, P. L. (2000). New support vector algorithms. *Neural computation*, *12*(5), 1207–1245.

Sculley, D. (2007). Online active learning methods for fast label-efficient spam filtering. In *Ceas* (Vol. 7, p. 143).

Settles, B. (2009). Active learning literature survey. *MINDS@UW*.

Thuruthel, T. G., Falotico, E., Renda, F., & Laschi, C. (2017). Learning dynamic models for open loop predictive control of soft robotic manipulators. *Bioinspiration & biomimetics*, *12*(6), 066003.

Valdenegro-Toro, M. (2021). Exploring the limits of epistemic uncertainty quantification in low-shot settings. *arXiv preprint arXiv:2111.09808*.

Valdenegro-Toro, M., & Mori, D. S. (2022). A deeper look into aleatoric and epistemic uncertainty disentanglement. In *2022 ieee/cvf conference on computer vision and pattern recognition workshops (cvprw)* (pp. 1508–1516).

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual.* Scotts Valley, CA: CreateSpace.

Verwimp, E., De Lange, M., & Tuytelaars, T. (2021). Rehearsal revealed: The limits and merits of revisiting samples in continual learning. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 9385–9394).

Vijayakumar, S., D'souza, A., & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural computation*, *17*(12), 2602–2634.

Wehbe, B. (2020). *Long-term adaptive modeling for autonomous underwater vehicles* (Unpublished doctoral dissertation). Universität Bremen.

Wehbe, B., Fabisch, A., & Krell, M. M. (2017). Online model identification for underwater vehicles through incremental support vector regression. In *2017 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 4173–4180).

Wehbe, B., Hildebrandt, M., & Kirchner, F. (2019). A framework for on-line learning of underwater vehicles dynamic models. In *2019 international conference on robotics and automation (icra)* (pp. 7969–7975).

Wehbe, B., & Krell, M. M. (2017). Learning coupled dynamic models of underwater vehicles using support vector regression. In *Oceans 2017-aberdeen* (pp. 1–7).

Williams, C., & Rasmussen, C. (1995). Gaussian processes for regression. *Advances in neural information processing systems*, *8*.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)* (pp. 928–936).

A Hyperparameter Tuning

The hyperparameter tuning was performed using Bayesian optimization on a discrete set of parameters. For ease of use, the *Bayesian Optimization* tuner from Keras was used. The tuner was run for 60 iterations for each technique. Under the condition that the optimal parameters cover at least 5% of the search space, random search gives us a 0.95 probability of finding optimal parameters. Bayesian Optimization uses probability distribution to discover samples smartly, so the odds are good Bergstra & Bengio (2012). The explored set of parameters is given in Table A.1.

Parameter Name	**Considered Values**
Number of layers	{1, 2, 3, 4}
Units per layer	{4, 8, 16, 32, 64}
Learning rate	$\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$
Batch size	{1, 2, 4, 8, 16}
Early Stopping Patience	{3, 5, 9}

Table A.1: Hyperparameters considered in the tuning process

The size of the network was constrained to a maximum 4 × 64 as the network learns on a limited buffer size, therefore even if we want the number of parameters not to exceed the number of data points too much, we keep it like this. Each of the other parameters (patience, batch size, learning rate) was assessed for being relevant by fixing the rest of the parameters and trying to tune just one of them. This is how this reduced set of parameters was chosen.

The tuning process was run for each of the networks, with a fixed-sized buffer to 100 data points, fixed P in RIRO to 0.7, and fixed t of Threshold to 0.02. These were fixed without further thought and their tuning will be discussed further in Appendix D. This is done to decrease load in the hyperparameter running, as there were already many conditions to explore and these parameters are essential to the performance of the models, so they deserve more attention.

The parameters were trained by the iterative process discussed in the algorithms on the training set and were continually assessed using the with-held validation set based on the final cumulative MSE loss over the training process. The Offline baseline was assessed based on its MSE, as there is no cumulative MSE metric for it, as it does not follow the online learning process. The final hyperparameters chosen for each of the models can be found in Table A.2.

Parameter Name	**Offline**	**FIFO**	**FIRO**	**RIRO**	**Threshold**	**Greedy**	**Threshold-Greedy**
Number of layers	3	3	4	1	1	1	1
Units per layer	64	64	32	16	16	16	32
Learning rate	10^{-4}	10^{-4}	10^{-4}	10^{-3}	10^{-3}	10^{-3}	10^{-2}
Batch size	1	2	1	4	8	1	16
Early Stopping Patience	9	5	5	5	5	9	5

Table A.2: Optimal hyperparameters

B Formal Descriptions of Approaches

Define machine learning model M to be a learner, that can train on an assigned data set, and predict by performing a forward pass. We define the prediction function of the model to be $f : M \times X \rightarrow (m, s)$ where X is input, m represents the mean of the prediction, and s represents the standard deviation. Furthermore, we denote the set of the stored data as $D = \{(X_1, y_1), \dots (X_n, y_n)\}, n > 0$, the set of incoming data that the online learner learns on as $O = \{(X_1, y_1), \dots (X_k, y_k)\}, k > 0$, and finally the testing set as $T = \{(X_1, y_1), \dots (X_l, y_l)\}, l > 0$. Furthermore, we consider a testing metric $E : M \times T \rightarrow \mathbb{R}$, which we can apply in each iteration of the learning process.

The FIFO algorithm is described in Algorithm B.1. The FIRO algorithm is described in Algorithm B.2. The RIRO algorithm is described in Algorithm B.3. The Threshold algorithm is described in Algorithm B.4. The Greedy algorithm is described in Algorithm B.5. The Threshold-Greedy algorithm is described in Algorithm B.6.

Algorithm B.1 First-In-First-Out: FIFO

```
Require: Model $M$ with prediction function $f$, data set $D$ of size $n$ and training data $O$ of size $k$, empty
    sequence of results $R$.
  while $n > 0$ do
    $(X_c, y_c) \leftarrow (X_1, y_1)$
    $O \leftarrow \{(X_2, y_2), \dots (X_n, y_n)\}, (X_i, y_i) \in O$
    $n \leftarrow n - 1$
    $D \leftarrow \{(X_2, y_2), \dots (X_k, y_k), (X_c, y_c)\}, (X_i, y_i) \in D$
    Retrain $M$ on $D$
    $R \leftarrow [R, E(M, T)]$
  end while
```

Algorithm B.2 First-In-Random-Out: FIRO

```
Require: Model $M$ with prediction function $f$, data set $D$ of size $n$ and training data $O$ of size $k$, empty
    sequence of results $R$.
    while $n > 0$ do
      $(X_c, y_c) \leftarrow (X_1, y_1)$
      $O \leftarrow \{(X_2, y_2), \dots (X_n, y_n)\}, (X_i, y_i) \in O$
      $n \leftarrow n - 1$
      Pick random $i \in \{1, \dots k\}$
      $D \leftarrow \{(X_1, y_1), \dots (X_{i-1}, y_{i-1}), (X_{i+1}, y_{i+1}), \dots (X_k, y_k), (X_c, y_c)\}, (X_i, y_i) \in D$
      Retrain $M$ on $D$
      $R \leftarrow [R, E(M, T)]$
    end while
```

Algorithm B.3 Random-In-Random-Out: RIRO

```
Require: Model $M$ with prediction function $f$, data set $D$ of size $n$ and training data $O$ of size $k$, empty sequence of results $R$. Probability of acceptance parameter $p$.
  while $n > 0$ do
    $(X_c, y_c) \leftarrow (X_1, y_1)$
    $O \leftarrow \{(X_2, y_2), \dots (X_n, y_n)\}, (X_i, y_i) \in O$
    $n \leftarrow n - 1$
    Sample $a$ from $X \sim \text{Unif}(0, 1)$
    if $a < p$ then
      Pick random $i \in \{1, \dots k\}$
      $D \leftarrow \{(X_1, y_1), \dots (X_{i-1}, y_{i-1}), (X_{i+1}, y_{i+1}), \dots (X_k, y_k), (X_c, y_c)\}, (X_i, y_i) \in D$
      Retrain $M$ on $D$
    end if
    $R \leftarrow [R, E(M, T)]$
  end while
```

Algorithm B.4 Threshold

```
Require: Model $M$ with prediction function $f$, data set $D$ of size $n$ and training data $O$ of size $k$, empty sequence of results $R$. Threshold $t$ for acceptance of a point.
  while $n > 0$ do
    $(X_c, y_c) \leftarrow (X_1, y_1)$
    $O \leftarrow \{(X_2, y_2), \dots (X_n, y_n)\}, (X_i, y_i) \in O$
    $n \leftarrow n - 1$
    $(m, s) \leftarrow f(X_c)$
    if $s > t$ then
      Pick random $i \in \{1, \dots k\}$
      $D \leftarrow \{(X_1, y_1), \dots (X_{i-1}, y_{i-1}), (X_{i+1}, y_{i+1}), \dots (X_k, y_k), (X_c, y_c)\}, (X_i, y_i) \in D$
      Retrain $M$ on $D$
    end if
    $R \leftarrow [R, E(M, T)]$
  end while
```

Algorithm B.5 Greedy

```
Require: Model $M$ with prediction function $f$, data set $D$ of size $n$ and training data $O$ of size $k$, empty sequence of results $R$.
  while $n > 0$ do
    $(X_c, y_c) \leftarrow (X_1, y_1)$
    $O \leftarrow \{(X_2, y_2), \dots (X_n, y_n)\}, (X_i, y_i) \in O$
    $n \leftarrow n - 1$
    $(m, s) \leftarrow f(X_c)$
    $(X_i, y_i) \leftarrow \arg\min_{(X_j, y_j) \in D} s$ where $(m, s) = f(X_j)$
    $(m', s') \leftarrow f(X_i)$
    if $s > s'$ then
      $D \leftarrow \{(X_1, y_1), \dots (X_{i-1}, y_{i-1}), (X_{i+1}, y_{i+1}), \dots (X_k, y_k), (X_c, y_c)\}, (X_i, y_i) \in D$
      Retrain $M$ on $D$
    end if
    $R \leftarrow [R, E(M, T)]$
  end while
```

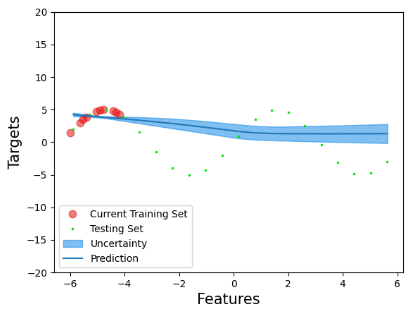
Algorithm B.6 Threshold-Greedy

```
Require: Model $M$ with prediction function $f$, data set $D$ of size $n$ and training data $O$ of size $k$, empty
    sequence of results $R$.
  while $n > 0$ do
    $(X_c, y_c) \leftarrow (X_1, y_1)$
    $O \leftarrow \{(X_2, y_2), \dots (X_n, y_n)\}, (X_i, y_i) \in O$
    $n \leftarrow n - 1$
    $(m, s) \leftarrow f(X_c)$
    $(X_i, y_i) \leftarrow \arg\min_{(X_j, y_j) \in D} s$ where $(m, s) = f(X_j)$
    $(m', s') \leftarrow f(X_i)$
    if $s > s' \wedge s > t$ then
      $D \leftarrow \{(X_1, y_1), \dots (X_{i-1}, y_{i-1}), (X_{i+1}, y_{i+1}), \dots (X_k, y_k), (X_c, y_c)\}, (X_i, y_i) \in D$
      Retrain $M$ on $D$
    end if
    $R \leftarrow [R, E(M, T)]$
  end while
```

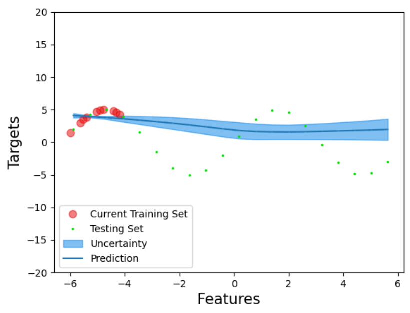
C Visualization of Applied Methods

The following figures visualize the methods used during the experiments using an example of a simple toy task. The dataset is composed of the values of the functions sin over a short range. During the incremental learning task, they are served to the model in order with increasing x.

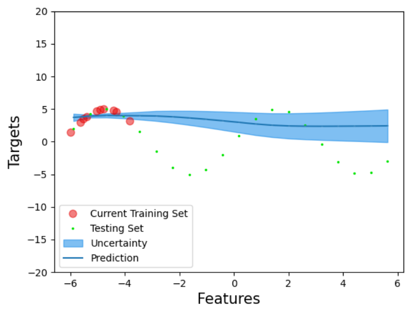
The following figures are gifs that can be viewed using pdf reader with a Javascript extension, for example, Okular or Adobe Reader. Furthermore, for the gifs to be activated, the current page has to be selected. This can be ensured by viewing the document in 1-page mode and viewing this single page at once.



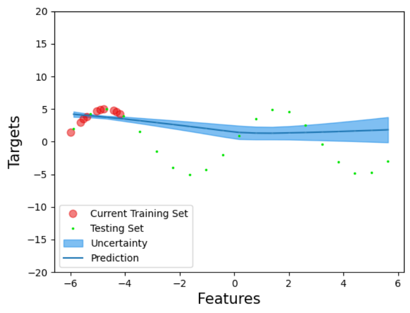
(a) **FIFO**



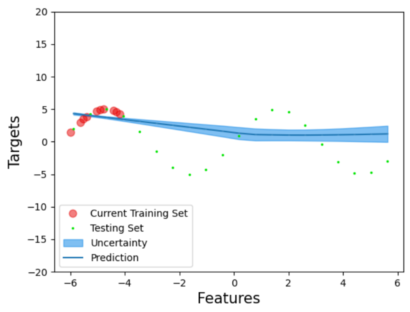
(b) **FIRO**



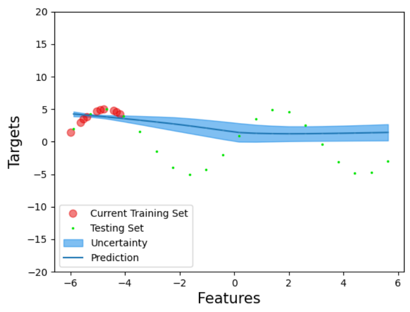
(c) **RIRO**



(d) **Greedy**



(e) **Threshold**

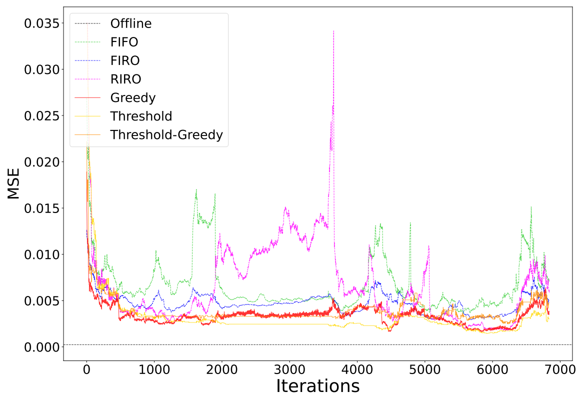


(f) **Threshold-Greedy**

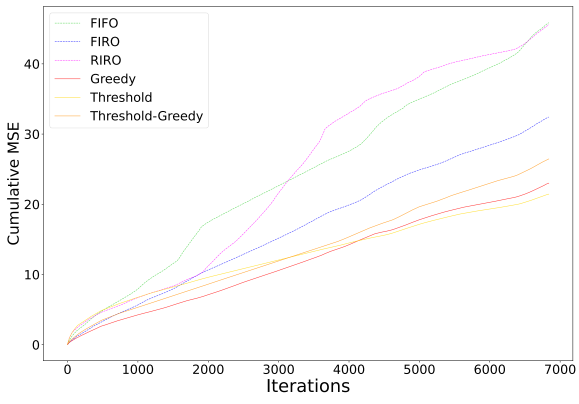
Figure C.1: Animated figures of the approaches taken in incremental online learning in this work.

D Extended Results

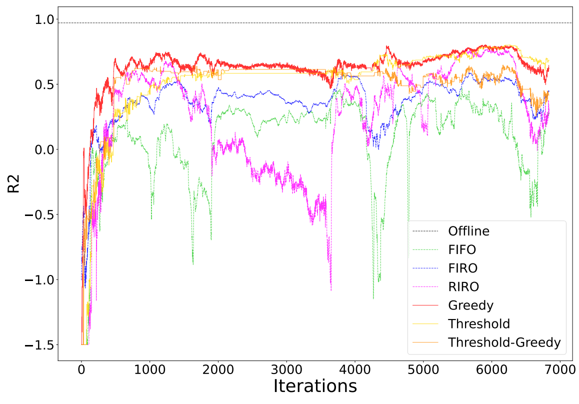
D.1 Basic Hyperparameter Tuning Results in More Detail



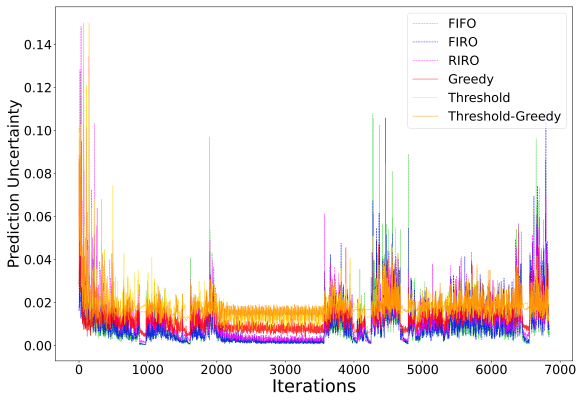
(a) MSE



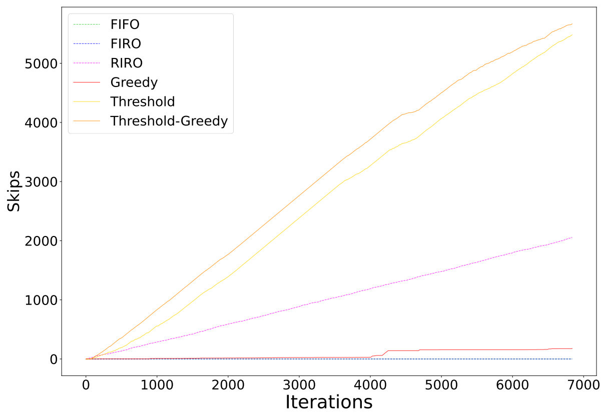
(b) Cumulative MSE



(c) R^2



(d) Prediction uncertainty



(e) Skips

Figure D.1: Basic hyperparameter tuning results

D.2 Tuning of RIRO in More Detail

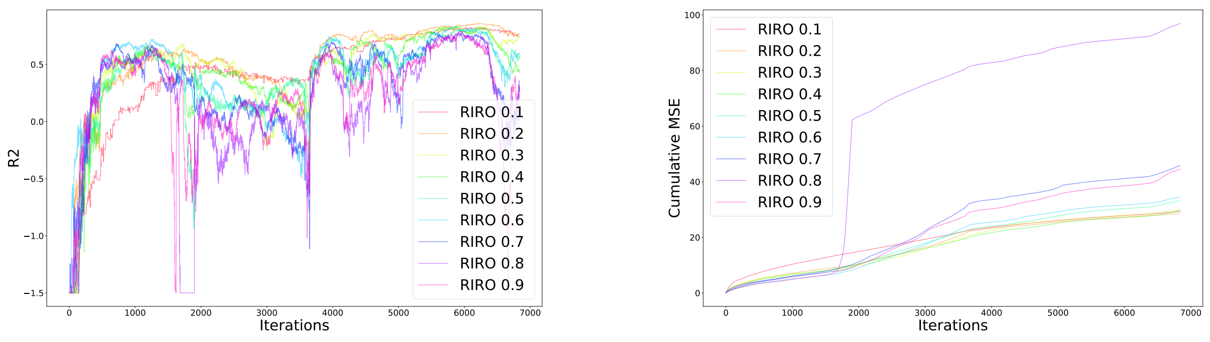


Figure D.2: Comparison of R^2 and Cumulative MSE for RIRO with varying p

D.3 Tuning of Threshold in More Detail

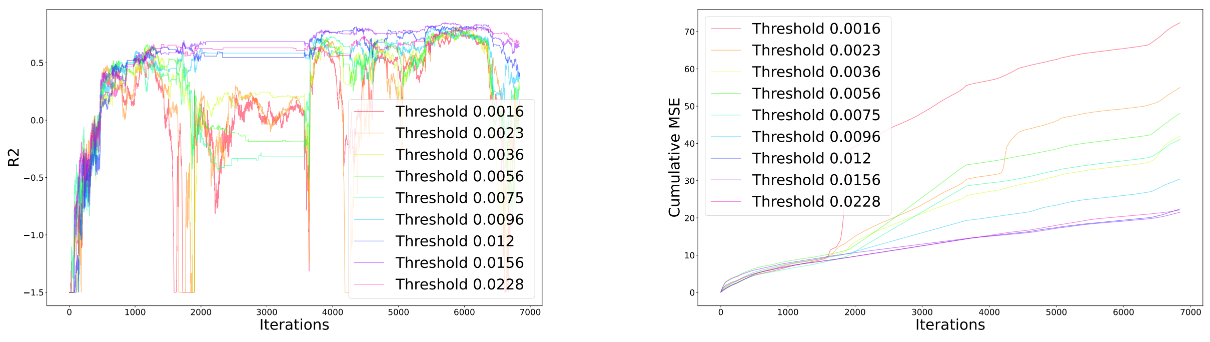


Figure D.3: Comparison of R^2 and Cumulative MSE for Threshold with varying t

D.4 Tuning of Threshold-Greedy in More Detail

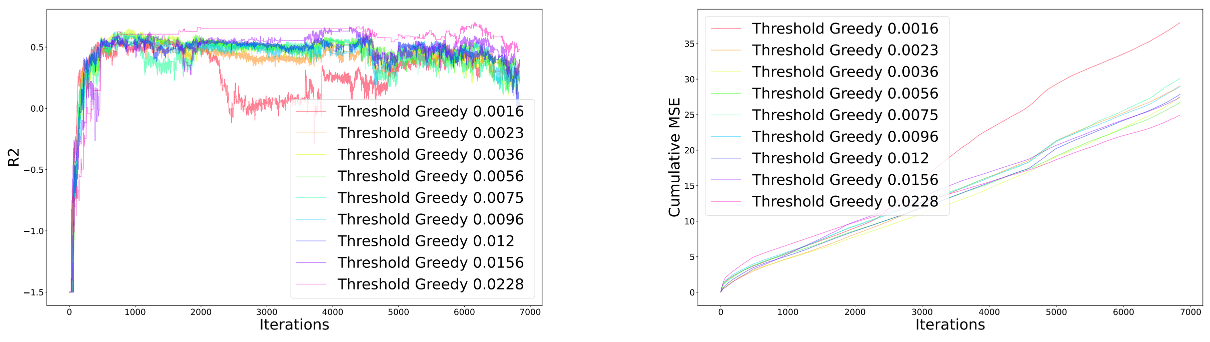
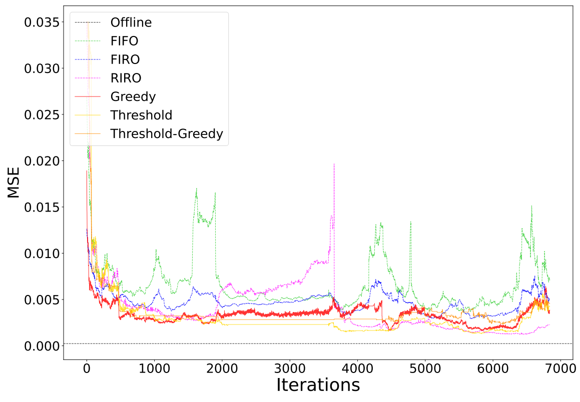
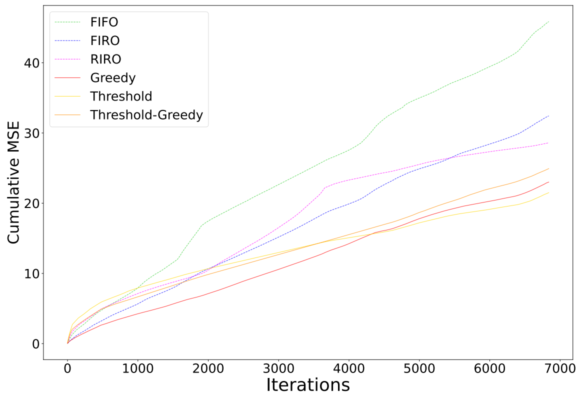


Figure D.4: Comparison of R^2 and Cumulative MSE for Threshold with varying t

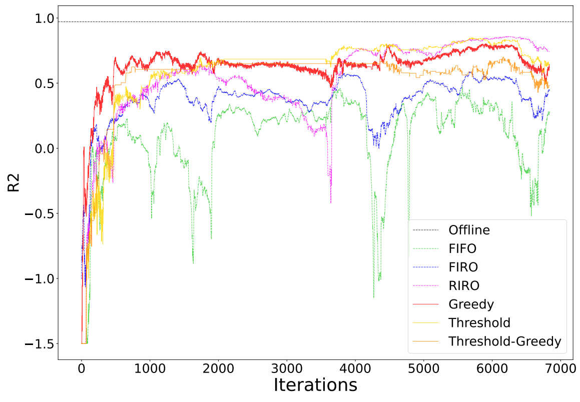
D.5 Final Comparison in More Detail



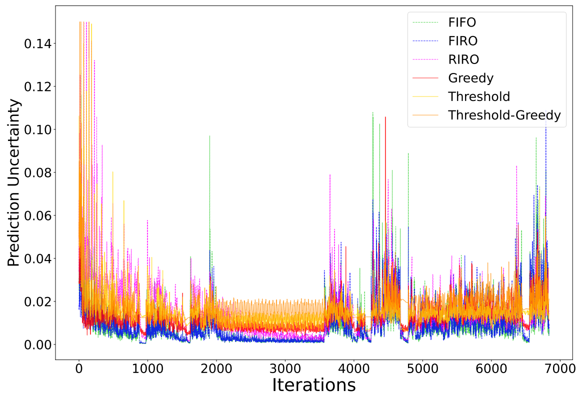
(a) MSE



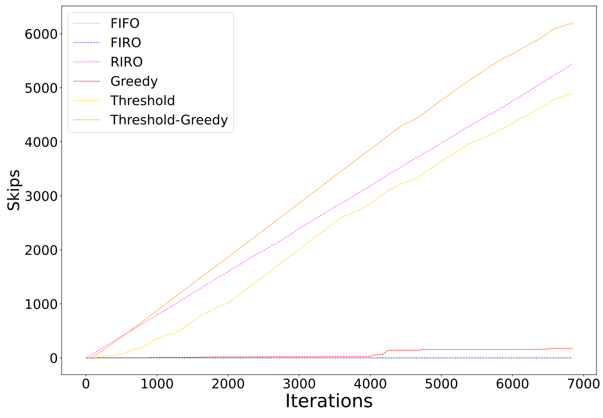
(b) Cumulative MSE



(c) R^2



(d) Prediction uncertainty



(e) Skips

Figure D.5: Final comparison plots

D.6 Analysis of Buffer in More Detail

D.6.1 FIFO

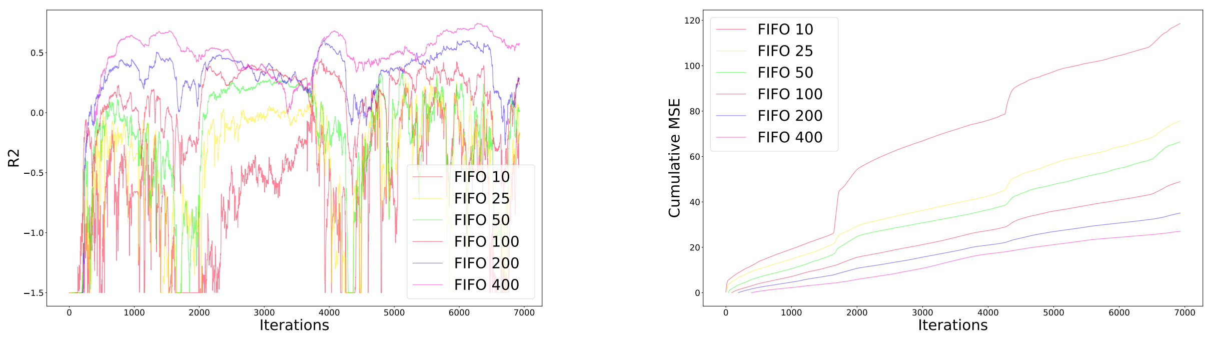


Figure D.6: Comparison of R^2 and Cumulative MSE for FIFO with varying buffer size

D.6.2 FIRO

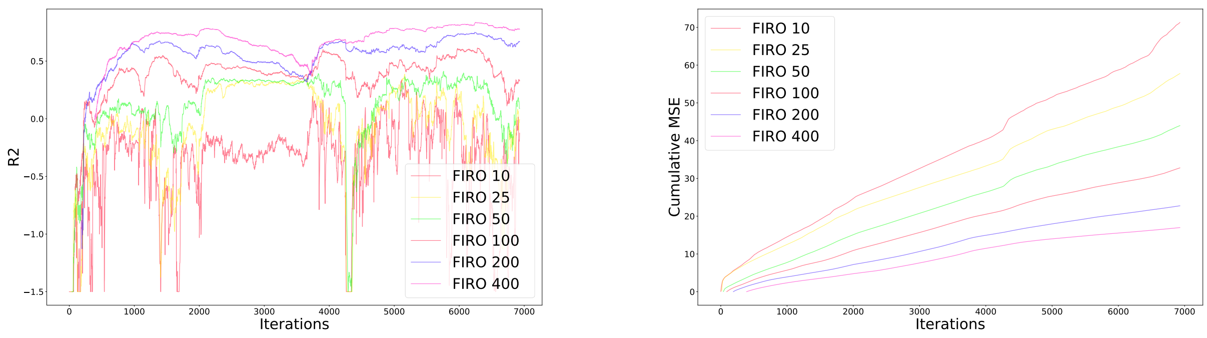


Figure D.7: Comparison of R^2 and Cumulative MSE for FIRO with varying buffer size

D.6.3 RIRO

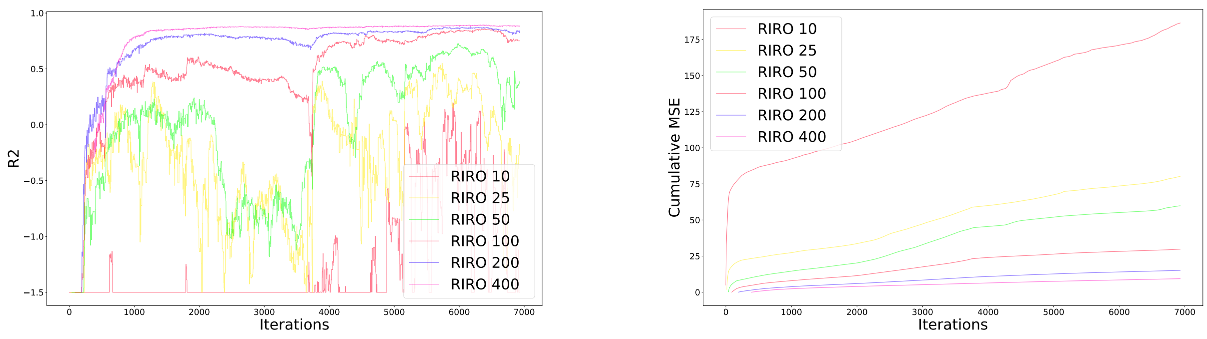
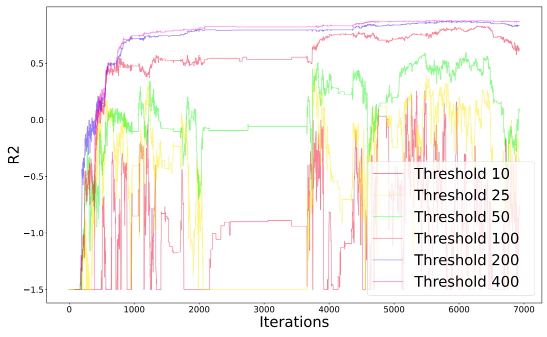


Figure D.8: Comparison of R^2 and Cumulative MSE for RIRO with varying buffer size

D.6.4 Threshold



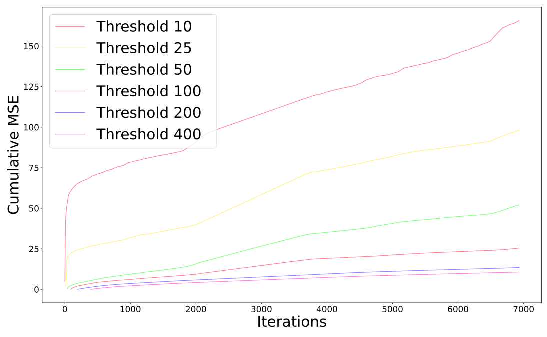
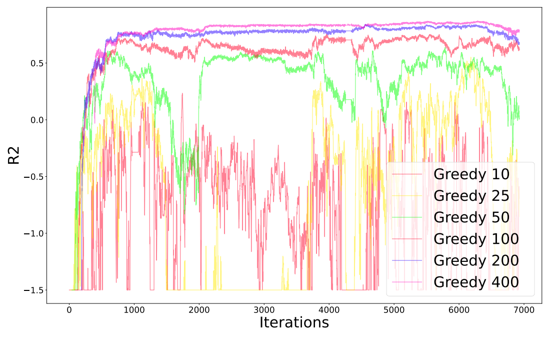


Figure D.9: Comparison of R^2 and Cumulative MSE for Threshold with varying buffer size

D.6.5 Greedy



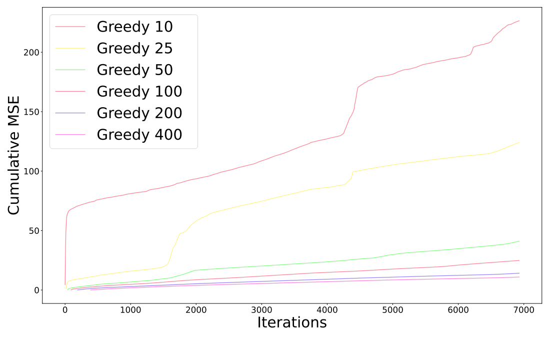
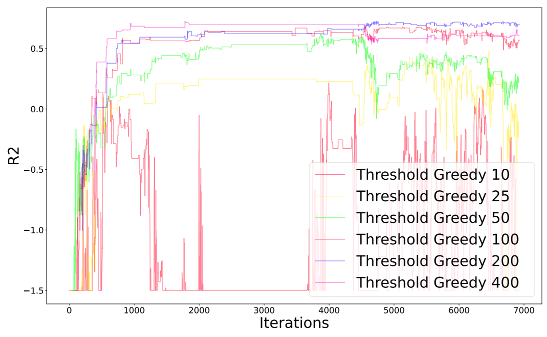


Figure D.10: Comparison of R^2 and Cumulative MSE for Greedy with varying buffer size

D.6.6 Threshold Greedy



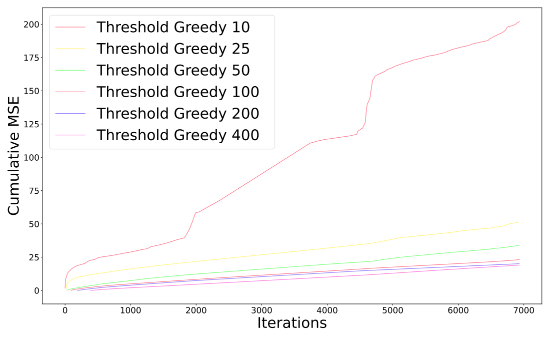


Figure D.11: Comparison of R^2 and Cumulative MSE for Threshold-Greedy with varying buffer size