Locally Weighted Regression Pseudo-Rehearsal for Adaptive Model Predictive Control

Grady R. Williams[*]
Georgia Institute of Technology
grady@vtr.us

Brian Goldfain[†]
Georgia Institute of Technology
bgoldfain3@gatech.edu

Keuntaek Lee
Georgia Institute of Technology
keuntaek.lee@gatech.edu

Jason Gibson
Georgia Institute of Technology
jgibson37@gatech.edu

James M. Rehg
Georgia Institute of Technology
rehg@gatech.edu

Evangelos A. Theodorou
Georgia Institute of Technology
evangelos.theodorou@gatech.edu

Abstract: We consider the problem of online adaptation of a neural network designed to represent system dynamics. The neural network model is intended to be used by an MPC control law for autonomous control. This problem is challenging because both input and target distributions are non-stationary, and naive approaches to online adaptation result in catastrophic forgetting. We present a novel online learning method, which combines the pseudo-rehearsal method with locally weighted projection regression. We demonstrate the effectiveness of the resulting Locally Weighted Projection Regression Pseudo-Rehearsal (LW-PR^2) method on an autonomous vehicle in simulation and real world data collected with a 1/5 scale autonomous vehicle.

1 Introduction

Autonomous systems operating in the real world must be capable of precise control in order to satisfy safety and performance criteria. This, in turn, requires highly accurate dynamics models which can be difficult to obtain. One of the most promising approaches is to learn the dynamics using function approximation methods. However, a major challenge with this approach is that the system dynamics could be constantly changing. Therefore, models which can adapt to changing dynamics online are desirable. In this work we examine the use of neural networks to model the system dynamics. Neural networks have recently been demonstrated to be effective at learning system models that can be used in high performance vehicle controllers [1], and are seeing increasing usage in the robotics control literature [2, 3, 4, 5]. However, adapting neural networks online is a notoriously difficult problem. The key issue that must be overcome is catastrophic forgetting [6, 7], which is the tendency for neural networks to forget old data when fed new data from a different distribution.

In order to overcome the catastrophic forgetting problem in neural networks, we make use of a class of modeling techniques which are naturally immune to catastrophic forgetting: locally weighted regression (LWR) [8]. LWR models work by building a global model up from a set of many small local models. Each model is equipped with a receptive field, which determines how much the model should respond to a given input. The output of the global model is then computed as a weighted average of all the local model outputs [9, 10, 11]. Since a given training pair only affects a localized region of the state space, LWR models can safely make incremental updates. One of the most mature local modeling methods is locally weighted projection regression (LWPR) [10]. LWPR has proven successful in modeling robot dynamics, even for high-dimensional systems in online learning scenarios.

[*]Author's affiliation has changed since this research was completed. New affiliation is Vtrus, Inc.

[†]Author's affiliation has changed since this research was completed. New affiliation is the Toyota Research Institute.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

Given the success of LWR models at the task of learning robot dynamics, a valid question is: why not use LWR models instead of neural networks for controlling the system? The issue with utilizing LWR in model predictive control is the computational cost. In the past, LWR models have been limited to inverse control, which only requires a single prediction per timestep, or offline trajectory/policy optimization [8, 12]. In cases where they have been used in MPC [13], the model had to be severely restrained in order to control the number of local models generated. The issue is that for LWR models to achieve high accuracy, typically thousands or tens of thousands of local models are required. Instead of utilizing LWR directly in an MPC controller, our method utilizes a neural network model for control, but updates it online using data collected from the system combined with pseudo-samples generated by an incrementally updated LWPR model.

2 Related Work on Neural Network Adaptation

Some of the earliest methods proposed for mitigating catastrophic forgetting in neural networks are rehearsal and pseudo-rehearsal [7, 14, 15, 16, 17]. In rehearsal methods, the original training data is retained and used alongside the new data in order to update the model. Pseudo-rehearsal methods do not retain old data, but instead they randomly create input vectors (pseudo-inputs) that are then fed through the current network in order to produce a corresponding output point (pseudo-output). The resulting artificially generated sample (pseudo-sample) can then be used for training the network alongside newly received data. The idea is that, by using the pseudo-samples alongside real data, the network can be encouraged to learn the new data without forgetting the current mapping. Recently, there has been a success using rehearsal [18] and pseudo-rehearsal based methods for vision tasks [19, 20, 21, 22]. In these methods the primary challenge that must be overcome is either storing previous data samples (in rehearsal methods) or randomly generating realistic inputs (for pseudo-rehearsal methods). In the case of learning system dynamics, generating pseudo-inputs is relatively easy due to the comparatively low dimension of the state-space. Instead, there is another challenge that must be overcome: *both* the input and target distributions are non-stationary. This means that sometimes new data must be learned while retaining old data, but other times new data must overwrite old data.

Besides rehearsal and pseudo-rehearsal methods, there are a variety of methods for updating neural networks which mitigate catastrophic forgetting by controlling how far the parameters of the model can move away from the current model. For instance, this is the approach taken in [23, 24, 25]. However, as in the case of rehearsal and pseudo-rehearsal, it is not clear how well controlling changes in the weights works when the target distribution is non-stationary, since in that case the network weights corresponding to previously learned data will need to be changed as well. Another promising approach to online adaptation for neural networks that has recently been explored is meta-learning [5]. However the meta-learning approach does not have an explicit mechanism to combat catastrophic forgetting, and it is currently unclear how to perform the meta-training in order to ensure that catastrophic forgetting cannot occur.

3 Problem Formulation

Consider an autonomous robot operating at some task, while performing the task the robot encounters system states, denoted $\mathbf{z}$, and executes controls, denoted $\mathbf{u}$. Our goal is to update the model of the system dynamics, which we assume is as a discrete time dynamical system:

$$\mathbf{z}_{t+1} = \mathbf{z}_t + \mathbf{F}(\mathbf{z}_t, \mathbf{u}_t; \theta)\Delta t \tag{1}$$

Where $\mathbf{F}$ is a model of the system dynamics and θ denotes the parameters of the model, in our case these are the weights of a neural network. Now, we define the following variables:

$$\mathbf{x} = \begin{pmatrix} \mathbf{z}_t \\ \mathbf{u}_t \end{pmatrix}, \quad \mathbf{y} = \frac{\mathbf{z}_{t+1} - \mathbf{z}_t}{\Delta t} \tag{2}$$

as the inputs and targets for our learning algorithm. As the robot navigates its environment, it encounters states and controls according to some probability distribution: $\mathbf{x} \sim \mathcal{P}_L(\mathcal{X})$. The distribution $\mathcal{P}_L$ is called the *Local Operating Distribution*, and it is task dependent. In addition to the *local operating distribution*, we assume that there is a system identification dataset, which has been collected beforehand and designed to contain data consisting of the necessary actions that the robot

needs to learn in order to operate competently. This dataset can be interpreted as a set of samples drawn from an underlying distribution: $\mathbf{x} \sim \mathcal{P}_{ID}(\mathcal{X})$ which we refer to as the *System Identification Distribution*. Note that we consider this distribution as stationary, but the mapping which takes input points drawn from this distribution to the corresponding outputs is not. We must be sure that by updating the model we do not forget any of the system modes contained in $\mathcal{P}_{ID}$.

First consider a simple approach to performing online model adaptation based on standard stochastic gradient descent (SGD). Suppose that we have access to streaming data, and that we maintain a set of recently encountered input and output pairs. By randomly drawing pairs from this set, we can get independent and identically distributed (I.I.D.) samples from the local operating distribution. Applying the standard SGD update law would then result in a model minimizing the objective: $\mathbb{E}_{\mathbf{x}\sim\mathcal{P}_L}\left[\|\mathbf{y} - \mathbf{f}(\mathbf{x};\theta)\|^2\right]$. However, this is *not* what we want! The issue with this is that the local operating distribution will typically not contain all the actions that the robot needs to operate effectively. A typical example is the case of an autonomous vehicle driving on the highway: during this mode of operation the vehicle only needs to maintain a constant velocity and make slight turns the vast majority of the time, if the model is updated with inputs purely drawn from a highway driving dataset, there is no guarantee that the model will remember the basic maneuvers necessary for other types of driving. This problem, known as *catastrophic forgetting*, is a well known deficiency of neural networks, and is especially problematic when the updated model is also used for control.

If we had access to I.I.D. samples from the system identification distribution, we could instead use an update law that jointly learns the target mapping for inputs drawn from both the system identification distribution and the local operating distribution. Additionally we would like to ensure that the model update rule never intentionally degrades the model on the system identification distribution. One way to achieve this is to ensure that update steps always move in the direction of the local minima for input data drawn from the system identification distribution. This constraint can be enforced by ensuring that the cosine of the angle between the update direction and the gradient computed from the system identification data is always positive, and we achieve this with the following update law:

$$\theta_{i+1} = \theta_i - \gamma\left(\alpha G_L(\theta_i) + G_{ID}(\theta_i)\right) \tag{3}$$

$$\alpha = \max_{a\in[0,1]} \; s.t \;\; \langle aG_L(\theta_i) + G_{ID}(\theta_i), G_{ID}(\theta_i)\rangle \geq 0 \tag{4}$$

Where γ is a learning rate, G_L is the gradient of the model parameters computed with data drawn from $\mathcal{P}_L$, and G_{ID} is the gradient computed with data from $\mathcal{P}_{ID}$. This update law balances the objective of simultaneously optimizing for local operating distribution and system identification distribution, and it constrains the combined gradient to always point in the same direction as the gradient computed from system identification data.

4 Locally Weighted Projection Regression Pseudo-Rehearsal

The problem with implementing the update rule defined by Eq. (3) is that, in an online setting, we only have access to data generated from the local operating distribution. Additionally, since the target mapping is changing, we cannot simply re-draw samples from the original system identification dataset or generate pseudo-outputs by running random inputs through the current model like in standard rehearsal and pseudo-rehearsal methods. Our strategy is to use artificially generated pseudo-training points to enforce the constraint, with the additional requirement that the pseudo-training points must match the changing target distribution. This means that artificially generating training points requires two steps (1) A method for generating artificial input points that are I.I.D. samples from $\mathcal{P}_{ID}(\mathcal{X})$, and (2) A method for computing the corresponding target, $\mathbf{y}$, for an artificially generated input point. This should be a function approximator that is capable of online adaptation, since the target mapping, $\mathbf{y}$, is actively changing.

Our approach is to train an LWPR model, which will be updated online, in order to compute the target mapping for artificially generated input points. For the generation of the input points, a Gaussian mixture model (GMM) is used. The artificial input/output pairs are then used to compute a synthetic gradient, which is used to regularize the online stochastic gradient descent via Eq. (3). We call this locally weighted projection regression pseudo-rehearsal (LW-PR^2), the algorithm consists of four sub-modules, which we now describe in detail. The overall flow of the algorithm is shown in Fig. 1.

Gaussian Mixture Model: The GMM generates synthetic input points consistent with the system identification dataset. We denote a mini-batch of synthetic input points as X_s. The GMM uses

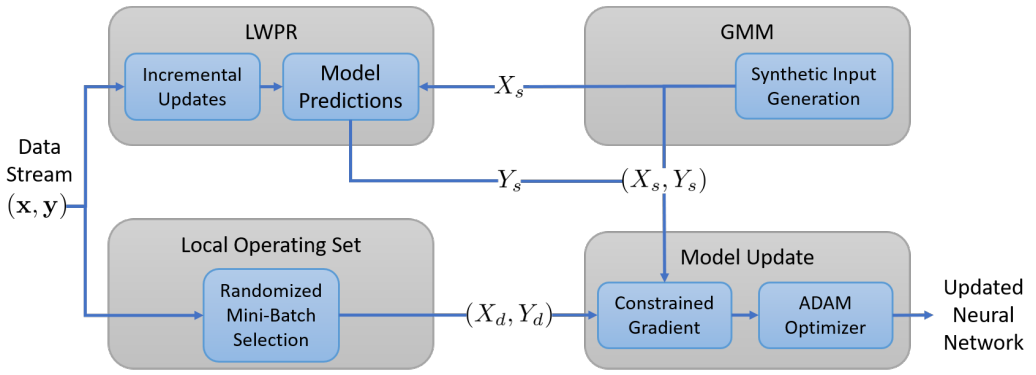

Figure 1: LW-PR^2 Algorithm. The GMM produces synthetic input points which are combined with predictions from LWPR to create synthetic training pairs. These are combined with randomized mini-batches created from recently collected data in order to compute the constrained gradient.

diagonal covariance matrices, and is trained using standard expectation maximization. We use the Bayesian Information Criterion (BIC) in order to select the number of Gaussian models used. After the initial training the GMM is not modified again.

LWPR Module: The LWPR module takes in the synthetic input points generated by the Gaussian mixture model, and then runs those input points forward through the LWPR model in order to produce synthetic output points. If we let $\mathbf{y}_i$ and $\mathbf{c}_i$ be the mean and center of the i_{th} local model and let D_i be the distance metric which defines the receptive field for the i_{th} model, then LWPR computes the global prediction as:

$$\mathbf{y} = \sum_{i=1}^{L} w_i \cdot f_i(\mathbf{x} - \mathbf{c}_i), \quad w_i = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_i)^{\mathrm{T}} D_i (\mathbf{x} - \mathbf{c}_i)\right)}{\sum_{j=1}^{L} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_j)^{\mathrm{T}} D_j (\mathbf{x} - \mathbf{c}_j)\right)} \tag{5}$$

Where f_i is the function defining the local model. Since the response of a given model to an input decays exponentially fast, model updates have only a negligible impact on models with centers far from the current input point. This is the feature that makes LWPR largely immune to catastrophic forgetting, and enables it to be safely updated online. The mini-batch output of the LWPR module is denoted Y_s. The LWPR model is initially trained over several epochs on the system identification dataset using the standard LWPR update rule. Online, the local linear models making up the LWPR model are continually updated.

Local Operating Set: The local operating set consists of the last several seconds of training points received from the stream of data generated by the system. Out of this set of data, randomized mini-batches are created (denoted as (X_d, Y_d)) and then fed into the model updater.

Model Update: The last module in the LW-PR^2 algorithm is the computation of the constrained gradient and the actual model update. First, the gradient for the local operating distribution, G_L, is computed using the mini-batch received from the local operating distribution, and then the gradient for the system identification dataset, G_{ID}, is computed using the artificial mini-batch received from the GMM and LWPR modules. The constrained gradient is computed via equation (3). After the constrained gradient is computed we use the ADAM optimizer [26] to perform the actual gradient descent update step.

5 Experimental Setup

We tested the LW-PR^2 algorithm on the task of learning vehicle dynamics on the AutoRally system [27]. The AutoRally system consists of a 1/5 scale ground vehicle built for high speed autonomous off-road driving, as well as a Gazebo simulation toolkit. For collecting the system identification dataset, we followed the procedure described in [28] and had an expert RC driver perform a set of choreographed maneuvers on a dirt test track. Collecting the system identification data is expensive, since it requires an expert driver to collect the data, followed by analysis and testing to ensure that enough data has been collected in the appropriate dynamic regimes. The notions of what constitutes

"enough data" and "appropriate dynamics regimes" are highly subjective and should be considered as a type of expert knowledge. The goal of remembering the system identification dataset can therefore be interpreted as an effort to preserve this expert knowledge.

We used a neural network structure with 2 hidden layers of 32 neurons each (the same structure as described in [28]). The input dimension is 6 and the output dimension is 4. Before the model adaptation algorithm can be applied, an initial model needs to be trained on the original system identification dataset. One option is to initialize the model using standard stochastic gradient descent like in [28], but we have found it is more effective to jointly train the initial model with the LWPR model and GMM model. This means that the actual system identification dataset takes the place of the local operating set, but synthetic data is still generated by the GMM and LWPR modules, which is used to compute the constrained gradient. We have observed that training the initial model in this manner has a negligible effect on the performance of the initial trained model, but helps with the adaptation since it synchronizes the neural network predictions with the LWPR predictions.

For a model predictive control algorithm we used Model Predictive Path Integral Control (MPPI) [28]. MPPI is a sampling based, gradient free, method which computes the optimal control as a cost weighted average over randomly sampled trajectories. We used a highly optimized GPU based implementation of MPPI in all of our experiments.

It is important to note that the LWPR model we use for generating synthetic data would not be feasible for usage with MPPI in a real-time control loop. Table 1 details the computational requirements of the neural network and LWPR respectively. For these calculations, we assume that a dot product operation takes $2N - 1$ floating point operations (FLOPs) for vectors of dimension N, and that a matrix-vector multiplication takes $2MN - M$ FLOPs where the matrix has dimension $M \times N$. We also assume that any non-linear function $(\exp, \tanh, (\cdot)^2)$ takes a single FLOP. For LWPR it can be difficult to predict the throughput required since the number of active local models can vary greatly, so we compute a lower bound based only on the minimum number of local model activations that must be computed. Computing a local model activation requires first subtracting the mean for the local model from the current input point (6 FLOPs), then individually squaring each result (6 FLOPs), then computing the dot product between the result and the receptive field weight ($2 \cdot 6 - 1$ FLOPs), and then computing the negative exponential of the result (2 FLOPs). This results in a total of 25 floating point operations for each local model, plus the additional computations required to actually compute the weighted average. The neural network simply consists of matrix-vector multiplies, the additions of the bias, and $\tanh$ non-linearities.

Table 1: Model Computation Comparison

	Size	FLOPs/Prediction
LWPR	5,645 (Receptive Fields)	$> 141,125$
Neural Network	1,412 (Weights and Biases)	$2,688$

The key takeaway from Table 1 is that making predictions with the neural network is two orders of magnitude cheaper than making predictions with an equally accurate LWPR model. Since MPPI needs to make millions of predictions per second, this is crucial. In contrast, the synthetic data generation rate only needs to match the rate of incoming real data points, which in our implementation is 40 Hz and easily achievable on modern hardware.

6 Results

We tested our LW-PR^2 approach using four different sets of experiments. Our first experiment, titled *Catastrophic Interference Test*, investigates the algorithm's ability to prevent catastrophic forgetting, using a dataset designed specifically to induce catastrophic forgetting on naive adaptation methods. The second experiment, titled *Modified Dynamics Test*, investigates the method's ability to adapt to drastic changes in the system dynamics using a driving dataset collected on a novel road surface. The third experiment, named *Simulated Autonomous Driving Test*, investigates how effective the updated model is when used as part of the MPC algorithm. Lastly, the fourth experiment, named *AutoRally Experimental Results*, consists of running the model adaptation during a full day of testing with the 1/5 scale autonomous driving system, and measures how well the algorithm works in a practical setting. Throughout these experiments there are 3 different types of experimental modes that are

run: (1) An **offline** test is a test where the model is not allowed to adapt during the experiment, (2) An **online** test is a test where the model is allowed to adapt during the experiment, but the adapted model is not used to control the vehicle, and (3) An **active** test is a test where the model is allowed to adapt during the experiment, and the adapted model is used to control the vehicle.

In an online training scenario, there is not an explicit training, validation, and test set. Instead, as each training pair is received, we compute the current error on that training pair, and we then record the result. After the error has been computed and recorded, the training pair is fed into the model updater. We compare our method against the base model (no adaptation), and the base model adapted with standard stochastic gradient descent. We also record the performance of the LWPR model used to generate synthetic training inputs.

Catastrophic Interference Test: Here we test our method on two off-road autonomous driving datasets, which we selected from the dataset accompanying [1]. The first dataset we call the *online training dataset* and the second is called the *offline validation dataset*. The online training dataset consists of 100 laps of slow speed driving around a roughly elliptical track in the clockwise direction. In order to test how well the model adaptation is able to remember other system modes, we utilize the offline validation dataset. The offline validation dataset consists of the same type of low speed monotonous driving in the *opposite* (counter-clockwise) direction of the online training dataset.

Table 2: Catastrophic Forgetting Test Results.

	Online Training Dataset				Offline Validation Dataset			
	Base	SGD	LW-PR^2	LWPR	Base	SGD	LW-PR^2	LWPR
Roll Rate	0.01	0.01	0.01	0.01	0.01	0.01	0.01	**0.00**
Long. Acc.	0.35	0.33	**0.32**	0.33	0.20	0.22	0.20	**0.1**
Lat. Acc.	0.69	0.63	**0.61**	0.65	0.99	1.56	1.10	**0.83**
Head. Acc.	2.11	**0.46**	0.49	0.53	1.47	2.11	0.83	**0.42**
Total MSE	0.79	**0.36**	**0.36**	0.38	0.67	0.97	0.53	**0.36**

The testing procedure works as follows: we first test the online performance of each of the methods using the online training dataset. Then, after the online test is finished, the final adapted model is frozen and an offline test (no adaptation allowed) is performed on the offline validation dataset. The results of these tests are shown in Table 2. All of the adaptive methods perform similarly on the online training dataset, and they all significantly decrease the total mean-squared error of the model predictions compared with the base neural network model. However, when using the final adapted model from the online training dataset on the offline validation dataset, the differences between the methods become apparent. The standard SGD method suffers from characteristic catastrophic forgetting, particularly in the heading acceleration which makes intuitive sense given the difference in the direction of travel between the two datasets. As expected, LWPR is unaffected by the change in local operating distribution, and performs better than the base network. Our LW-PR^2 method performs only slightly worse than LWPR and outperforms the base network, demonstrating its ability to overcome catastrophic forgetting.

Modified Dynamics Test: In this experiment we test the ability of the algorithm to adapt to highly modified vehicle dynamics. The dynamics are modified by running the vehicle on tarmac instead of dirt. All of the system identification data was collected on a dirt surface, so this is a completely novel dynamics regime for the system. In order to collect this dataset we ran MPPI (with the base model) for 100 laps in the clockwise direction (which generates the online training dataset), and then we ran the same algorithm for 100 laps in the counter-clockwise direction (which generates the offline validations dataset). The challenge in this test is that the *target* distribution has changed, which makes standard rehearsal or pseudo-rehearsal methods problematic since old data will be out of sync with the new data (i.e. the same input point may map to multiple outputs). By using incrementally updated LWPR to generate synthetic data, we can overcome this challenge since the artificially generated targets will eventually synchronize with current target distribution.

Table 3 shows the performance on the online training dataset and the offline validation dataset respectively. All of the online adaptation methods improve on the online training set, with LW-PR^2 and LWPR performing best. Interestingly, the standard SGD method performs best on the offline validation dataset. In this case SGD is able to correctly generalize knowledge gained from the clockwise direction to the counter-clockwise direction, whereas the local nature of LWPR and LW-PR^2

restricts the ability of the model to transfer knowledge about the clockwise direction to the counter-clockwise direction. This is a trade-off which is required to prevent catastrophic forgetting, and even though standard SGD worked well in this case, there is no guarantee that it will generalize knowledge correctly and it can instead forget previously learned knowledge (as shown in the previous experiment).

Table 3: Modified Dynamics Test Results

	Online Training (Tarmac)				Offline Validation (Tarmac)			
	Base	SGD	LW-PR^2	LWPR	Base	SGD	LW-PR^2	LWPR
Roll Rate	0.01	0.01	0.01	**0.01**	0.01	0.01	0.01	0.01
Long. Acc.	0.60	0.99	0.54	**0.38**	1.11	**0.47**	0.52	0.74
Lat. Acc.	0.58	0.24	0.35	**0.16**	1.75	**0.43**	0.49	0.84
Head. Acc.	1.04	0.76	0.53	**0.48**	2.84	**1.66**	1.82	2.32
Total MSE	0.56	0.50	0.36	**0.26**	1.43	**0.64**	0.71	0.98

Simulated Autonomous Driving Tests: In this experiment, we test the algorithm in an active setting where MPPI uses the updated model to control the vehicle. We use an open source Gazebo simulation of the AutoRally vehicle from [27] for these experiments. The system identification dataset that we use to train the base model is the same as in the previous sections (i.e. it is based on real world data). Note that the simulation dynamics are significantly different from the real-world AutoRally dynamics, so the starting base model is highly inaccurate.

We ran three different model adaptation settings: standard SGD, LW-PR^2, and no adaptation. For each setting we performed trials running 10 laps around the track, and we collected 5 trials for each different setting. The desired speed is set at $8m/s$. The results of all of the trials are shown in Table 4. Using the base model or the LW-PR^2 adapted model, the controller is able to successfully navigate around the track. However, when using standard SGD to update the model the controller consistently fails after completing 1 lap, typically the controller tries to take a turn at too high of a speed, resulting the vehicle rolling onto its side.

Table 4: Gazebo Simulation Results

	Base Network	SGD	LW-PR^2
Avg. Laps Completed	10	1	10
Avg. Trial MSE	1.84	2.49	**0.65**
Avg. Lap Time	34.78	N/A	**32.04**

Figure 2 shows the progression of lap times and total MSE per lap as each trial progresses. On average, it takes less than one lap for MPPI to start benefiting from the model adaptation: as the model adapts it realizes it can go faster without slipping in the simulation than it can in the real world and the result is a performance increase. The performance on the second lap is significantly better with the LW-PR^2 adapted model than with the base model. After the second lap, the model continues to make small improvements in the per lap MSE.

AutoRally Experimental Results: In this section, we examine how the model adaptation scheme works in a practical setting - the model adaptation is turned on at the beginning of a day of testing and allowed to run uninterrupted for the entire day. The resulting dataset consists of over 1 hour of autonomous data collected over a period of 4.5 hours. The 1 hour of autonomous data consists of approximately 18 kilometers of driving data with speeds up to 50 kph. Note that this dataset contains a significant amount of natural variation: the early morning runs are with a fresh damp track, whereas test runs in the afternoon are with a drier track that has less grip. Many of the tests start when the vehicle has a fully charged battery, and then end with a dead battery, and then are continued with another fully charged battery, this means that the adaptation has to constantly re-learn similar parts of the dynamics over again.

For this dataset we record the same performance metrics as in the earlier online dataset experiments for each of the four adaptation strategies. Additionally, we have available the active performance of LW-PR^2 data since the model produced by LW-PR^2 was being used to drive the vehicle autonomously. Note that the errors produced in these experiments are on average higher than the

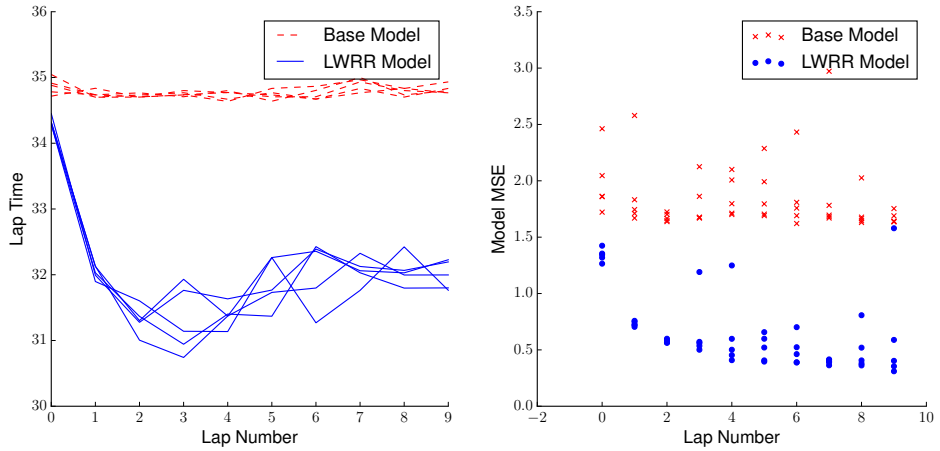

Figure 2: Lap time (in seconds) and total MSE accumulated per lap for LW-PR^2 and the base model.

	Base	SGD	LW-PR^2	LWPR
Roll Rate	0.01	0.01	0.01	0.01
Long. Acc.	2.73	2.28	2.30	**2.06**
Lat. Acc.	1.71	1.29	**1.24**	1.28
Head. Acc.	8.28	**4.48**	4.87	4.54
Total MSE	3.18	2.10	2.11	**1.97**
Active MSE	N/A	N/A	2.54	N/A

Table 5: Autonomous Field Test Results

Figure 3: AutoRally during field test with MPPI and LW-PR^2

errors reported in the previous sections, this is due to the vehicle moving faster and producing higher accelerations which in turn lead to higher errors. The results over the full day of testing are given in Table 5. Once again, all of the incremental methods significantly improve the performance from the base model. The SGD and LW-PR^2 methods perform nearly identically on the online test, but unlike the model produced by SGD, which resulted in crashes during the simulation tests, the model updated with LW-PR^2 can be utilized by MPPI to competently control the system.

7 Conclusion

In this paper we have presented LW-PR^2, which is a novel method for pseudo-rehearsal that is applicable to learning dynamics models in changing environments. The key contribution is the use of an incrementally updated LWPR model in order to create artificial training pairs. The use of incrementally updated LWPR enables pseudo-rehearsal to be applied to systems where both the input and target distributions are non-stationary. The usage of LWPR enables a constrained gradient update to be used, which ensures that the adaptation does not degrade the model performance on the system identification distribution. In order to test our method we created a series of datasets and simulation tests that stressed essential requirements for an online adaptation scheme: the ability to prevent catastrophic forgetting, adapt to drastic changes in the dynamics, and the ability to produce models usable by an MPC controller. These experiments demonstrated the capability and practicality of our approach on a challenging real world system.

Acknowledgments

Supported by Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. Also supported in part by Amazon Web Services (AWS) and Komatsu Ltd.

References

[1] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018. URL https://ieeexplore.ieee.org/document/8558663.

[2] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018. URL https://arxiv.org/abs/1708.02596.

[3] I. Lenz, R. A. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics Science and Systems (RSS)*, 2015. URL http://www.roboticsproceedings.org/rss11/p12.html.

[4] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017. URL https://arxiv.org/pdf/1610.00696.pdf.

[5] I. Clavera, A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt: Meta-learning for model-based control. *CoRR*, abs/1803.11347, 2018. URL http://arxiv.org/abs/1803.11347.

[6] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[7] R. Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

[8] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

[9] S. Schaal and C. G. Atkeson. Constructive incremental learning from only local information. *Neural computation*, 10(8):2047–2084, 1998.

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

[11] F. Meier, P. Hennig, and S. Schaal. Incremental local gaussian regression. In *Advances in Neural Information Processing Systems*, pages 972–980, 2014.

[12] D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal control for redundantly actuated arms. In *International Conference on Simulation of Adaptive Behavior*, pages 93–102. Springer, 2008. URL https://link.springer.com/chapter/10.1007/978-3-540-69134-1_10.

[13] G. Williams, E. Rombokas, and T. Daniel. Gpu based path integral control with learned dynamics. In *Neural Informations Processing Systems: Autonomously Learning Robots Workshop*, 2014. URL https://arxiv.org/abs/1503.00330.

[14] A. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pages 65–68. IEEE, 1993.

[15] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Journal of Neural Computing, Artificial Intelligence and Cognitive Research*, 7:123–146, 1995.

[16] A. Robins. Sequential learning in neural networks: A review and a discussion of pseudorehearsal based methods. *Intelligent Data Analysis*, 8(3):301–322, 2004.

[17] R. M. French. Using pseudo-recurrent connectionist networks to solve the problem of sequential learning. In *Proceedings of the 19th Annual Cognitive Science Society Conference*, volume 16, 1997.

[18] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5533–5542. IEEE, 2017. URL https://arxiv.org/abs/1611.07725.

[19] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017. URL https://arxiv.org/abs/1705.08690.

[20] C. Atkinson, B. McCane, L. Szymanski, and A. V. Robins. Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting. *CoRR*, abs/1812.02464, 2018. URL http://arxiv.org/abs/1812.02464.

[21] D. Mellado, C. Saavedra, S. Chabert, and R. Salas. Pseudorehearsal approach for incremental learning of deep convolutional neural networks. In *Latin American Workshop on Computational Neuroscience*, pages 118–126. Springer, 2017. URL https://link.springer.com/chapter/10.1007/978-3-319-71011-2_10.

[22] R. Kemker and C. Kanan. Fearnet: Brain-inspired model for incremental learning. *CoRR*, abs/1711.10563, 2017. URL http://arxiv.org/abs/1711.10563.

[23] S. Wan and L. E. Banta. Parameter incremental learning algorithm for neural networks. *IEEE Transactions on Neural Networks*, 17(6):1424–1438, Nov 2006. ISSN 1045-9227. doi:10.1109/TNN.2006.880581.

[24] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017. URL https://arxiv.org/abs/1612.00796.

[25] Z. Li and D. Hoiem. Learning without forgetting. *Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018. URL https://arxiv.org/abs/1606.09282.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[27] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg. Autorally: An open platform for aggressive autonomous driving. *IEEE Control Systems Magazine*, 39(1): 26–55, Feb 2019. URL https://ieeexplore.ieee.org/document/8616931.

[28] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic mpc for model-based reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1714–1721. IEEE, 2017.