Fast Re-learning of a Controller from Sparse Data

Charles E. Martin and Heiko Hoffmann
HRL Laboratories, LLC
Malibu, California 90265
{cemartin, hhoffmann}@hrl.com

***Abstract*—We address the problem of adapting a controller of a dynamical system to an unexpected change in dynamics. Such a system can be controlled using model predictive control if the model of the dynamics (forward model) is sufficiently accurate. The challenge is to adapt the forward model quickly. We motivate the requirement to achieve this adaptation given only sparse training data. To solve this challenge, we introduce the concept of preserving any a priori learned functional relationship in the dynamics, while adapting solely to the relatively simple functional relationship describing the change in dynamics. We show that this concept can be realized by augmenting a forward model with a simple corrector network and demonstrate feasibility on a challenging control problem in simulation.**

I. Introduction

Autonomous and semi-autonomous driving and flight require a high degree of reliability. Americans drive about 2 trillion miles per year; thus, even a few errors per 100,000 miles [1], [2] would scale the total number of yearly mistakes to more than 10,000,000 (more than the current total number of car accidents [3]). Reliability of autonomous cars and the corresponding trust that drivers have for their cars will be key for future development. Likewise, increased autonomy in flight will require similar assurances of reliability and trust if the benefits of reduced human intervention and reduced cost are ever to be achieved.

To improve resilience, the autonomous system has to automatically adapt its control to unexpected change or damage. After damage, the old internal/ forward model of the system is inappropriate for control and has to be replaced with a model that captures the new dynamics.

The challenge for autonomy is to do this adaptation fast. An autonomous car or aircraft might have only a few seconds to react before a crash occurs.

We see the largest obstacle to this challenge not in the computational complexity of adaptation but in the lack of useful data that can be sampled in a short period of time. Even if the computational resources were insufficient at present time, eventually the problem would vanish due to Moore's law. In contrast, the time to sample sufficient data will not increase. Increasing the sampling rate will be insufficient because the inertia of the system effectively creates an upper bound on the sampling rate. Beyond such bound, the added data is like linear interpolation that does not provide extra information to re-compute a new system model.

Thus, we need learning methods that adapt to few data. Methods like support vector regression [4], Gaussian processes [5], and principal component analysis [6] typically require a lot of data to relearn a functional relationship. On-line learning methods gradually adapt an internal model with every single data point [7]–[9] but cannot adapt to a sudden change. The expected change in the system dynamics may be large, particularly, after damage.

To cope with few data, the solution has to preserve any a priori learned structure in the dynamics, while adapting just to the change in the dynamics. On-line learning methods may do that but assume bounds on the magnitude of the change. We suggest instead that the key is to assume bounds on the functional description of the change. For example, the damage may keep intact most details in the functional description of the dynamics and alter only the offsets in a certain parameter region. The change in offsets could be large, but could be described with a simple function.

In this article, as a specific solution to our challenge, we propose to augment a given model of the forward dynamics with a simple feed-forward neural network. Specifically, we use a multi-layer perceptron with one hidden layer and only a few neurons, essentially, a linear combination of a few basis functions [10]. The perceptron trains on new data and adapts to a sudden change in dynamics without changing the potentially complex structure of the a priori learned or designed forward model.

The idea of adding the simple corrector network is related to having a growing neural network. These networks have been studied before [11]–[14], but not in the context of adapting a controller after system damage.

As a proof of concept, we demonstrated our method on a challenging control problem in simulation. The task was to balance two poles simultaneously on a cart, while controlling only the linear force on the cart [15]–[18]. To test robustness to damage, we abruptly changed the length of one of the poles. In this task, the adaptation had to be fast. Nevertheless, we could successfully adapt the forward model and use model predictive control [19] to compute the forces on the cart.

The remainder of this article is organized as follows. Section 2 describes the methods, in particular, our adaptation technique and control methodology. Section 3 presents the results, illustrating the benefits of our method and its dependance on the number of neurons in the corrector network. Section 4 covers conclusions and potential future work.

II. Methods

Our objective is to control a plant that is defined by a dynamic state and a set of fixed parameters describing its physical characteristics. The plant responds to a control signal that alters the state of the plant. We start with a controller that can maintain the state within a desired range of pre-defined target values.

After damage or unexpected system change, the physical characteristics and parameters of the plant change. Depending on the severity of the damage, the controller may no longer be capable of controlling the plant. If the changes in the plant's parameters are large enough, it may rapidly transition to a failure state.

The objective of a resilient controller is to adapt to the unknown changes in the parameters of the plant and thereby regain control of the plant's state or to at least slow the transition from normal operation to failure. In this section we first describe our *neural-model-predictive-control* (*neural-MPC*) architecture and then cover its implementation and test on the double-pole balancing problem.

A. Resilient Controller Architecture and Adaptation

Our neural-MPC model of resilient control permits rapid recovery or graceful failure of damaged plants. We begin under the assumption that we have a controller with adaptable parameters that does a good job of controlling the plant under normal operating conditions. We will refer to this controller as the *base controller*. However, once the plant is damaged and its parameters change we need a controller architecture that is capable of rapidly adapting to the modifications.

In this paper, we propose the controller architecture shown in Fig. 1. This architecture consists of an adaptable *base controller*, a static forward model, and an adaptable *forward model corrector*. The base controller accepts a measurement of the plant state as input and generates a control signal in response. The forward model accepts a measurement of the plant state and control signal as input and produces a prediction of the next plant state. The forward model corrector accepts the same input as the forward model and generates an additive correction to the forward model output in response. In this study we used neural network embodiments of the base controller and forward model corrector because they permit simple, robust, easily trained implementations.

The neural-MPC controller receives a measurement of the current state of the plant as input, and its components (base controller, forward model, and forward model corrector) work together to generate a control signal in response. Specifically, during each time-step of the control process, the base controller, forward model, and forward model corrector operate in a closed loop for a predefined period of time to generate sequences that consist of pairs of control signals and resulting predicted plant states. These sequences are used to adapt the base controller and determine the control signal generated by the neural-MPC controller.

When controlling an altered/damaged plant the neural-MPC controller operates by first accepting as input a measurement of the current state of the plant. This initial measurement is fed into the base controller, which generates a control signal in response. The forward model and forward model corrector both accept as input the measured state of the plant and the control signal and together generate a prediction of the next state of the plant. At this point the neural-MPC controller operates in a closed loop to generate a sequence of length L

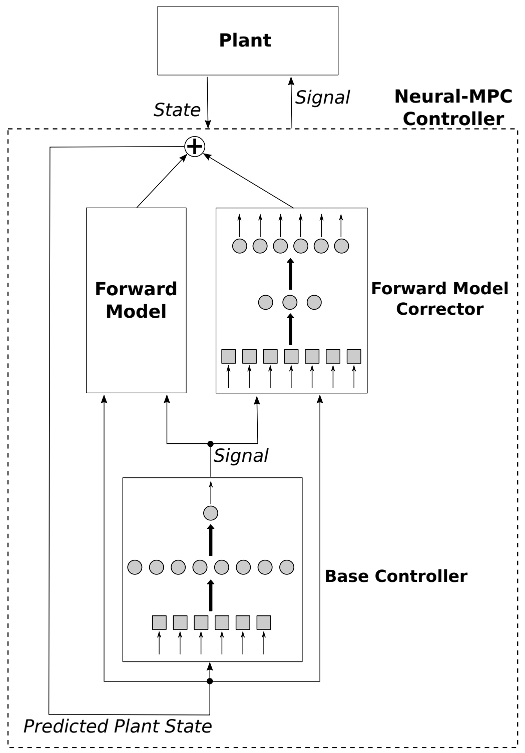

Fig. 1. Architecture of the neural-MPC controller, which consists of three primary components: a static forward model, an adaptable base controller, and an adaptable forward model corrector. After receiving the current state of the plant as input the neural-MPC controller operates in a closed loop to generate a sequence consisting of pairs of control signals and resulting predicted plant states. This process is repeated a pre-defined number of times to generate a set of signal-state sequences, each time starting with the most recently measured plant state and a randomly modified base controller policy.

of pairs of control signals and corresponding predicted states. The weights of the base controller are randomly modified (explained below) to create a new control policy, and a new control sequence of length L is generated, again starting from the most recently measured state of the plant. This process is repeated until N_s control sequences have been generated, where each sequence was created using a unique, randomly generated base controller.

The N_s sequences are then used to determine the next control signal to apply to the plant, and which of the randomly generated base controllers will be used next. Specifically, we assume that there exists an error function defined over the space of plant states, which specifies the quality of any given state. The next control signal is determined by finding the state that minimizes the error function among the final states in each of the N_s sequences, and then selecting the first control signal from the minimizing sequence. Furthermore, whichever base controller generated the minimizing sequence is the controller that is used on the next time-step.

Due to the alteration in the parameters of the plant, the original base controller is no longer entirely accurate. However, typically some of the information it contains will generalize to the new plant, limiting the search space of control policies implemented by the base controller and enabling quick convergence on a new control policy. To do so, we implemented a simple, but effective search strategy for modifying the weights (policy) of the base controller. On a given time-step, and for each of the N_s sequences, each weight w_i of the base controller was randomly modified according to

$$w_i \leftarrow w_i + 0.1|w_i|N(0,1), \tag{1}$$

where $N(0,1)$ is the Normal distribution with zero mean and unit variance. Thus, on each time-step of the control process, a local region of the weight-space of the base controller is searched for a better control policy.

However, it is not enough to only adapt the base controller. The forward model must also be adapted to correct its deficiencies, otherwise the hypothetical signal-state sequences being generated to adapt the base controller and select the next control signal will not be sufficiently accurate.

In our framework, we leave the parameters of the original forward model fixed and instead adapt the weights of a simple feed-forward network with a single hidden layer. This forward model corrector network provides additive corrections to the output of the forward model. Let $\vec{s}(t)$ be the measured state of the plant at time t, $F_c(t)$ the resulting control signal from the base controller, $\frac{d\vec{s}(t)}{dt} = \vec{M}_f(\vec{s}(t), F_c(t))$ the forward model, and $\vec{N}_f(\vec{s}(t), F_c(t))$ the output of the forward model corrector network. Given $\vec{s}(t)$ and $F_c(t)$ the predicted next plant state is given by

$$\vec{s}(t+\Delta t) = \vec{s}(t) + \int_t^{t+\Delta t} \vec{M}_f(\vec{s}(t), F_c(t))dt + \vec{N}_f(\vec{s}(t), F_c(t)). \tag{2}$$

Let $\hat{\vec{s}}(t+\Delta t)$ be the true, observed next plant state. The forward model corrector network is trained to minimize the prediction error

$$E(t;\vec{w}) = \left|\left|\hat{\vec{s}}(t+\Delta t) - \vec{s}(t+\Delta t)\right|\right|, \tag{3}$$

where $\vec{w}$ are the weights of the corrector network. On each time-step the gradient of $E(t;\vec{w})$ is computed with respect to $\vec{w}$ and one step of gradient descent is applied to the weights of the corrector network. As the controller attempts to control the damaged plant, the forward model corrector and base controller simultaneously adapt to the plant's new operating parameters.

B. Experimental Methods

This section covers the implementation details of the neural-MPC model and its application as a resilient controller for the double-pole balancing problem. This problem was selected because it is a challenging and widely used benchmark task in the domain of control. Furthermore, we are unaware of any previous work that has attempted to apply resilient controllers to this problem.

1) Double Pole Balancing: The double-pole balancing problem (Fig. 2) is a classic benchmark control problem, particularly, for neural network controllers [15]–[17]. The task is to balance two poles with different lengths that are hinged to the top of a cart that moves along a track of finite length. The controller attempts to keep the poles up-right by applying a force F_c to either side of the cart in a direction parallel to the track. To be successful, the controller must keep the cart within a specified distance x_{limit} from the center of the track, and it must keep each pole within a specified angular limit θ_{limit} from the vertical. The equations governing the dynamics of a cart with N poles are

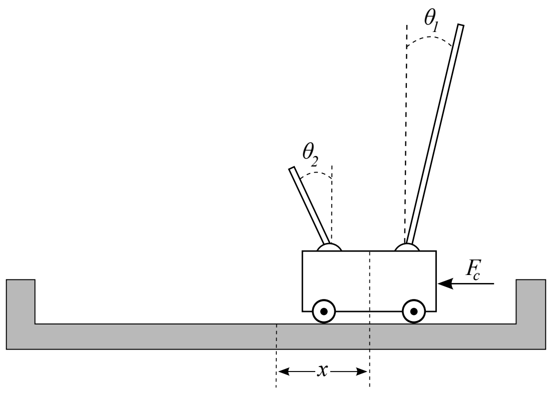

Fig. 2. The cart-pole system used in the double pole balancing problem. The state of the system is defined by the position x of the cart relative to the center of the track, and the angular positions θ_1 and θ_2 of the large and small poles relative to the vertical. The control force F_c is applied to the side of the cart, in a direction parallel to the track.

$$\ddot{x} = \frac{F_c - \mu_c \operatorname{sgn}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{m_c + \sum_{i=1}^{N} \tilde{m}_i}, \tag{4}$$

$$\ddot{\theta} = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_i\dot{\theta}_i}{m_i l_i}\right), \tag{5}$$

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4}m_i\cos\theta_i\left(\frac{\mu_i\dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right),$$

$$\tilde{m}_i = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right),$$

for $i = 1, 2, ..., N$, (see [17]). Here, F_c is the control force applied to the cart, and $g = 9.8\text{m/s}^2$ is the acceleration due to gravity. The variable x represents the position of the cart relative to the center of the track, $\dot{x}$ is its velocity, and $\ddot{x}$ is its acceleration. The mass of the cart is m_c, and the coefficient of friction between the cart and the track is μ_c. The angular position of the i^{th} pole relative to the vertical is θ_i, and its angular velocity and angular acceleration are $\dot{\theta}_i$ and $\ddot{\theta}_i$, respectively. The mass of the i^{th} pole is m_i, its length is $2l_i$, and the coefficient of friction between the pole and its hinge is μ_i. The variable $\tilde{F}_i$ represents the effective force from the i^{th} pole on the cart, and $\tilde{m}_i$ is its effective mass. The "sgn" symbol represents the signum function. This system of equations constitutes the forward model.

All experiments started with same base controller, which was trained to control the cart-pole system when its parameters were set to the most commonly used values [16]: mass of the cart $m_c = 1$kg, mass of the 1st pole $m_1 = 0.1$kg, mass of the 2nd pole $m_2 = 0.01$kg, coefficient of friction between the cart and the track $\mu_c = 5 \cdot 10^{-4}$Ns/m, coefficients of friction between the poles and their hinges $\mu_1 = \mu_2 = 2 \cdot 10^{-6}$Nms, length of the 1st pole $l_1 = 0.5$m, and length of the 2nd pole $l_2 = 0.05$m. The control force was restricted to the interval $F_c \in [-10\text{N}, 10\text{N}]$. The parameters defining the domain of successful control were set to $x_{limit} = 2.4$m, and $\theta_{limit} = 90°$. The initial state of the cart-pole system was drawn randomly and independently from the uniform probability distributions $x, \theta_1, \theta_2 \in U[-0.01, 0.01]$ and $\dot{x}, \dot{\theta}_1, \dot{\theta}_2 \in U[-0.1, 0.1]$. The equations governing the dynamics of the system were solved numerically using a fourth-order Runge-Kutta method with a step-size of 0.01s. Consequently, the original forward model was integrated using the same method. During a simulation, the state of the cart-pole system was given to the controller every 0.02s, at which point the control force was updated.

2) Implementation Details: The neural-MPC model was implemented in Java. The computational experiments presented in Section III each ran on a computer with a quad-core 2.3 GHz Intel Core i7 processor and 10 GB of shared RAM. The base controller was implemented as a feedforward neural network with 6 input nodes, 10 hidden nodes with hyperbolic tangent transfer functions, 1 output node with a hyperbolic tangent transfer function, and no bias unit. The output of the base controller was scaled by 10N. It was trained offline using a combination of particle swarm optimization and self-assembly known as Swarm Intelligent Network Optimization through Self-Assembly (SINOSA) [18].

When implemented in the neural-MPC model, the base controller was adapted using 50 control sequences ($N_s = 50$) each of length $L = 50$ (see Sec. II-A). The cost function used to determine the "winning" control sequence was

$$\text{Cost}\left(x, \dot{x}, \theta_1, \dot{\theta_1}, \theta_2, \dot{\theta_2}\right) = x^2 + \dot{x}^2 + \theta_1^2 + \dot{\theta_1}^2 + \theta_2^2 + \dot{\theta_2}^2. \quad (6)$$

This cost function defines the ideal state of the cart-pole system in which the cart is at the center of the track, both poles are perfectly upright, and there is no movement.

The forward model corrector network was implemented as a feedforward neural network with 7 input nodes, 6 linear output nodes, and no bias unit. We tried various numbers of nodes in the hidden layer, which all had hyperbolic tangent transfer functions. The corrector network was trained using basic gradient descent with a learning rate of 0.05. At the beginning of each trial all the weights were randomly initialized to values in the interval [-1,1], and the output weights were scaled by a factor of $\frac{0.004}{N_h}$, where N_h is the number of neurons in the hidden layer. This scaling was done to prevent the network from being too disruptive to the forward model predictions before it had a chance to learn. This scaling limited the initial impact of the corrector network on the forward model predictions, but the fairly large learning rate of 0.05 ensured that it could adapt quickly to correct inaccuracies in the forward model.

III. Results

In this section, we present the results of the experiments that we ran to test our neural-MPC resilient controller. The base controller was able to keep the *undamaged* cart-pole system close to the equilibrium state. To simulate a sudden damage, the length and mass of the large pole were reduced, and its state was perturbed away from the equilibrium state of $\left(x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2\right) = \vec{0}$. We tested the ability of our neural-MPC controllers to control the cart-pole system at six different levels of damage: 25%, 30%, 35%, 40%, 45%, and 50% reductions in the length and mass of the large pole. The greater the reduction in the length of the large pole, the more difficult it was to recover from the damage. We found that at damage levels less than 25% the original base controller was capable of preventing the system from failing, albeit the deviations from the equilibrium state were larger.

Figures 3 through 5 show an example of a neural-MPC controller that uses a forward model corrector network with 3 hidden neurons regaining control of the cart-pole system after 30% damage. The figures show cart and pole positions and velocities and the prediction error E after damage onset.

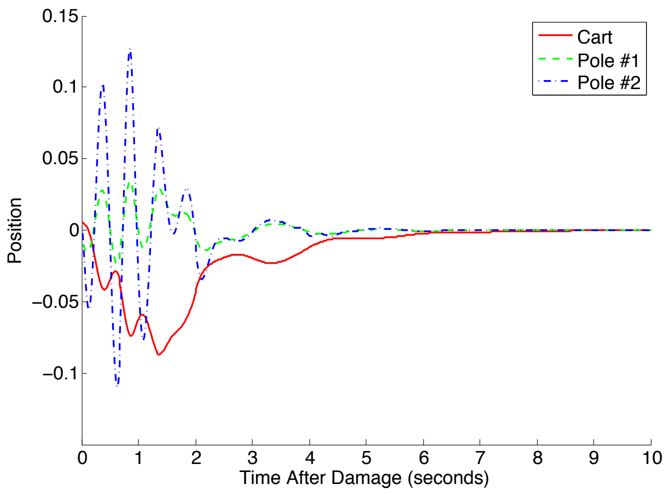

Fig. 3. Positions of the cart (in meters) and poles (in radians) after damage to the cart-pole system. After an initial transient period the neural-MPC controller learned enough about the new behavior of the damaged system and was able to center the cart and poles.

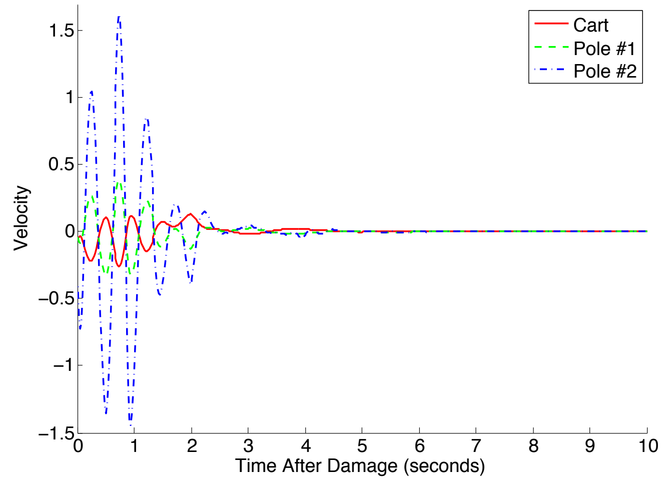

Fig. 4. Velocities of the cart (in $\frac{m}{s}$) and poles (in $\frac{rad}{s}$) after damage to the cart-pole system. After an initial transient period the neural-MPC controller learned enough about the new behavior of the damaged system and was able to stabilize its dynamics.

In Figures 3 and 4, the positions and velocities of the cart and poles varied substantially during the first two seconds after damage had occurred, but they quickly converged towards zero as the neural-MPC controller adapted to the changes in the system. This adaptation is reflected in Fig. 5 where the error exhibited large excursions from zero during the first second after damage had occurred and converged to zero as the corrector network learned to correct the inaccuracies in the forward model.

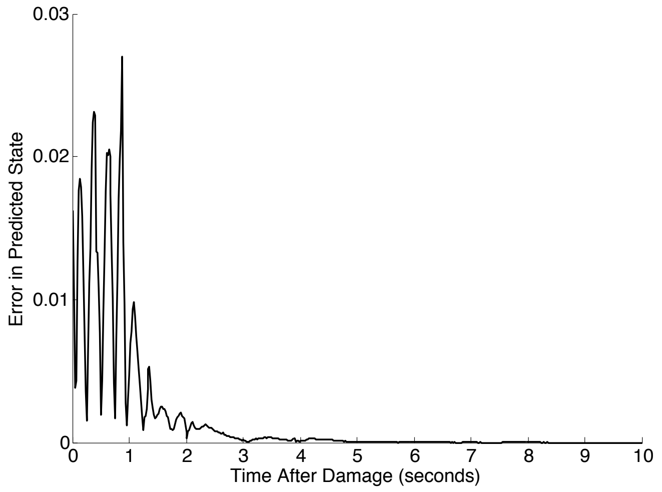

Fig. 5. Error in the predictions of the cart-pole system state made by the neural-MPC controller after damage to the cart-pole system. The forward model corrector network quickly learned to correct the inaccuracies in the forward model, thereby allowing the controller to regain control of the system. The error is defined to be the norm of the difference between the next state predicted by the controller and the actual state observed at the next time.

For each level of damage, we tested forward model corrector networks with 1, 2, 3, 4, 5, 10, 15, 20, 30, and 40 hidden neurons. For every level of damage and each corrector network size, we ran 200 trials where every trial started with a different randomly select initial state of the cart-pole system and a unique seed for the random number generator. Each trial was run until the cart-pole system failed or 3000 time-steps were taken, which is equivalent to $3000 * 0.02\mathrm{s} = 1$ minute of simulated time.

Figure 6 shows the probability of the cart-pole system remaining in the success domain after 20s for various levels of damage when using either the neural-MPC controller with 3 or 10 hidden neurons in the corrector network or just the base controller without forward-model adaptation. Figures 7 and 8 show the same statistics for the cases of 40s and 60s survival times, respectively.

In all three cases, the neural-MPC controllers substantially outperformed the original base controller. Even at the lowest levels of damage, the cart-pole system failed fairly quickly when under the control of the original base controller. In contrast, our neural-MPC controllers were often capable of completely regaining control of the system at lower levels of damage and exhibited much longer survival times at higher levels of damage. Furthermore, both the 3 and 10 hidden-neuron cases did well, indicating that it is possible to use relatively small networks when we exploit the a priori given functional relationship in the original forward model.

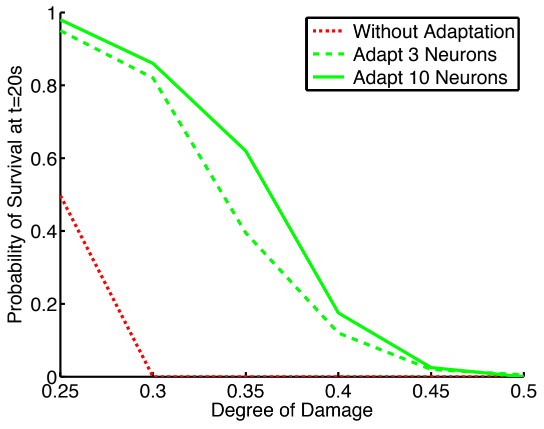

Fig. 6. Probability of the cart-pole system being successfully controlled for at least 20 seconds after various levels of damage occurred. The comparison is made for forward model corrector networks with 3 and 10 hidden neurons respectively and for the case of no adaptation. Our neural-MPC controllers substantially outperformed the non-adaptable controller at all levels of damage.

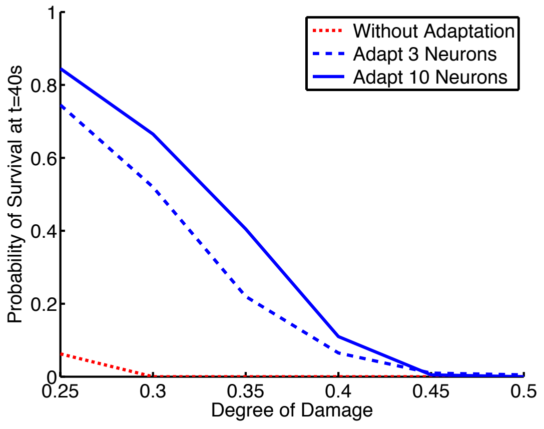

Fig. 7. Probability of the cart-pole system being successfully controlled for at least 40 seconds after various levels of damage occurred. The comparison is made for forward model corrector networks with 3 and 10 hidden neurons respectively and for the case of no adaptation. Our neural-MPC controllers substantially outperformed the non-adaptable controller at all levels of damage. The non-adaptable controller had zero probability of survival above the 25% level of damage.

We compared the performances of our neural-MPC controllers when different numbers of hidden neurons were used in the forward model corrector network. Figure 9 shows the average probability of survival for a range of different corrector network sizes for survival times of 20s, 40s, and 60s. For a given corrector network size and survival time, the average was taken over all six levels of damage. Controllers with larger corrector networks tended perform better on average. However, the improvements began to asymptote around 10 hidden neurons and good results were obtained with as few as 2 or 3 hidden neurons.

IV. Conclusions and Future Work

In this paper, we motivated the problem of fast re-learning the system dynamics after an unexpected change or damage to the system. We pointed out the importance to re-learn given

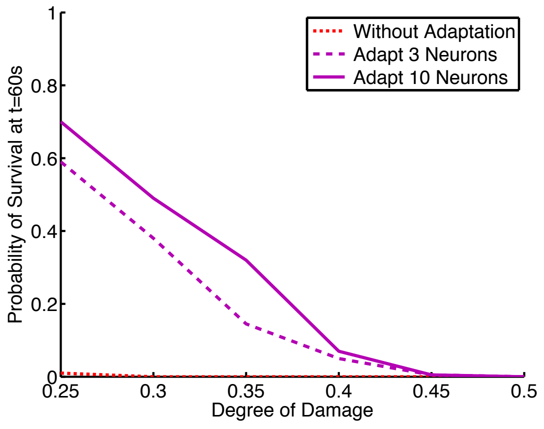

Fig. 8. Probability of the cart-pole system being successfully controlled for at least 60 seconds after various levels of damage occurred. The comparison is made for forward model corrector networks with 3 and 10 hidden neurons respectively and for the case of no adaptation. Our neural-MPC controllers substantially outperformed the non-adaptable controller at all levels of damage.

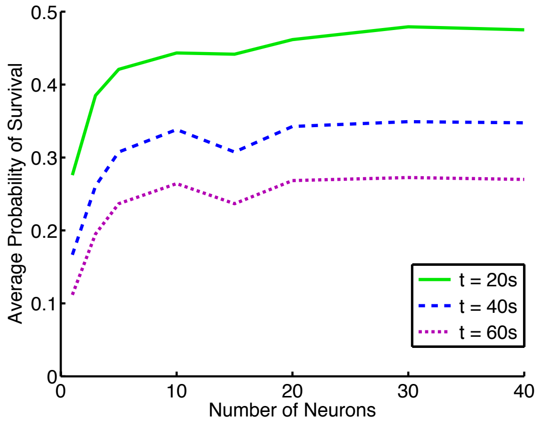

Fig. 9. Average probability of survival for forward model corrector networks of various sizes. The average was taken over all six levels of damage. Results are shown for survival times of 20s, 40s, and 60s. While corrector networks with 10 or more hidden neurons tended to perform the best, even those with only 2 or 3 hidden neurons did well on average.

only sparse data. Different from other work that learns from sparse data [13], we looked at the problem of recovery from damage, which is time-critical, provides data only in a short time window, and generally does not allow the use of prior knowledge of the change in dynamics caused by the damage.

We proposed to adapt a forward model to a sudden change by adding a simple corrector neural network. This strategy enables us to maintain the a priori learned or designed functional relationship, while capturing the change in dynamics with a simple function. On the down side, this simplicity may limit the complexity of the change that our method can handle.

We tested our approach on the challenging problem of controlling a cart that balances two poles simultaneously. To apply our augmented forward model, we used a model-predictive-control architecture. We demonstrated that we could successfully recover control after abruptly changing the length of a pole in settings that would result in failure without adaptation. The results also showed that a simple neural network with a few hidden neurons was sufficient for recovery, supporting the notion that a simple function was sufficient to capture the change in dynamics.

Since recovery after damage is time critical, the gains of the corrector network and its learning rate should be within suitable bounds. Empirically, we found that these parameters should at least have the right order of magnitude. The automatic setting of these parameters will be left for future research.

References

[1] Google's Self-Driving Cars: 300,000 Miles Logged, Not a Single Accident Under Computer Control. The Atlantic, 10 August 2012.

[2] Google Cars Drive Themselves, in Traffic. New York Times, 10 October 2010.

[3] 2012 Quick Facts. US National Highway Traffic Safety Administration, Publication No. 812006, 2014.

[4] Vapnik VN. The nature of statistical learning theory. Springer-Verlag, New York, 1995.

[5] Rasmussen CE, Williams CKI. Gaussian Processes for Machine Learning. MIT Press, 2006.

[6] Diamantaras KI, Kung SY. Principal Component Neural Networks. John Wiley & Sons, New York, 1996.

[7] Oja E. A simplified neuron model as a principal component analyzer. Journal of Mathematical Biology 15(3): 267-273, 1982.

[8] Ouyang S, Bao Z, Liao GS. Robust recursive least squares learning algorithm for principal component analysis. IEEE Transactions on Neural Networks 11(1): 215-221, 2000.

[9] Hoffmann, H., Schaal, S., and Vijayakumar, S. Local dimensionality reduction for non-parametric regression. Neural Processing Letters, Vol. 29, pp. 109-131, 2009.

[10] Haykin, S. Neural Networks: A Comprehensive Foundation. Prentice Hall, Paramus, NJ, 1998.

[11] Elizondo, E., Birkenhead, R., Góngora, M., et al. Analysis and test of efficient methods for building recursive deterministic perceptron neural networks. *Neural Networks*, Vol. 20, pp. 1095-1108, 2007.

[12] Fahlman, S. and Lebiere, C. The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Adv. in neural info. processing systems II* (pp. 524-532). San Francisco, CA: Morgan Kaufmann, 1990.

[13] Frean, M. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2, 198-209, 1990.

[14] Islam, M., Sattar, A., Amin, F., Yao, X., and Murase, K. A New Adaptive Merging and Growing Algorithm for Designing Artificial Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 39, pp. 705-722, 2009.

[15] Gruau, F., Whitley, D., and Pyeatt, L. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st annual conference on genetic programming (GECCO)*, pp. 81-89. Cambridge, MA: MIT Press, 1996.

[16] Jiang, F., Berry, H. and Schoenauer, M. Supervised and evolutionary learning of echo state networks. *Lecture Notes in Computer Science*, 5199, 215-224, 2008.

[17] Wieland, A. Evolving neural network controllers for unstable systems. In *Proceedings of the international joint conference on neural networks (IJCNN)*, vol. 2, pp. 667-673. New York: IEEE. 1991.

[18] Martin, C. E. Adapting Swarm Intelligence For The Self-Assembly And Optimization Of Networks (Dissertation). College Park, MD: University of Maryland, 2011.

[19] Allgwer; Zheng. Nonlinear model predictive control. Progress in Systems Theory 26. Birkhauser, 2000.

[20] B. Bischoff, D. Nguyen-Tuong, H. van Hoof, A. McHutchon, C. E. Rasmussen, A. Knoll, J. Peters, M. P. Deisenroth. Policy Search For Learning Robot Control Using Sparse Data, IEEE International Conference on Robotics and Automation, 2014.