Certifiably-correct Control Policies for Safe Learning and Adaptation in Assistive Robotics

Keyvan Majd[1†], **Geoffrey Clark**[1], **Tanmay Khandait**[1], **Siyu Zhou**[1],

Sriram Sankaranarayanan[2], **Georgios Fainekos**[3], **Heni Ben Amor**[1]

[1]Arizona State University, [2]University of Colorado Boulder, [3] Toyota NA-R&D

[†]majd@asu.edu

Abstract

Guaranteeing safety in human-centric applications is critical in robot learning as the learned policies may demonstrate unsafe behaviors in formerly unseen scenarios. We present a framework to locally repair an erroneous policy network to satisfy a set of formal safety constraints using Mixed Integer Quadratic Programming (MIQP). Our MIQP formulation explicitly imposes the safety constraints to the learned policy while minimizing the original loss function. The policy network is then verified to be locally safe. We demonstrate the application of our framework to derive safe policies for a robotic lower-leg prosthesis.

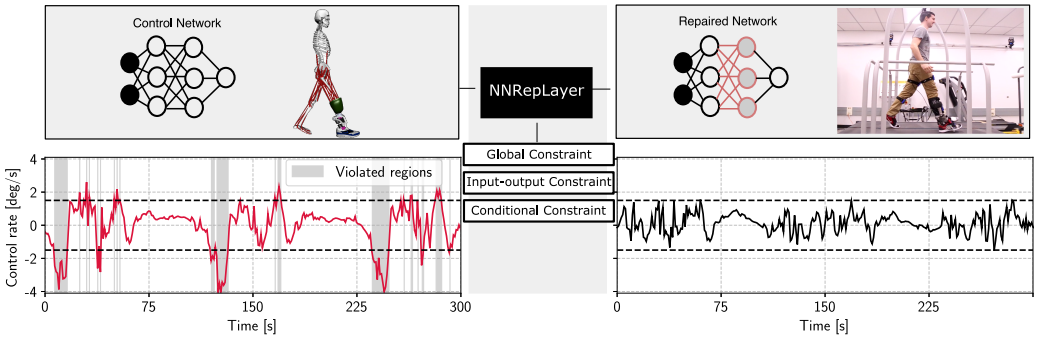

Figure 1: Left: A trained neural-network policy to control a prosthesis violates formal safety constraints. Right: Our framework repairs the violation while maintaining the underlying behavior.

1 Introduction

Deep network policies can help capturing the complicated human-robot interaction dynamics and adapting control parameters automatically to the user's individual characteristics in the assistive robotic devices. Despite this advanced capability, deep neural network (DNN) policies are not widely used in the field of assistive robotics since the trained policies may generate unsafe behaviors when encountering unseen inputs. In case of lower-leg prosthesis, for example, control values and joint angles should not exceed certain limits. One approach to ensure that a NN satisfies a given set of safety properties is to use retraining and fine-tuning based on counter-examples [1–5]. However, this approach has a number of pitfalls. First of all, in both retraining and fine-tuning the labels of the desired data, i.e., the data that satisfy the constraints, are not known. Most critically, retraining and fine-tuning are typically based on gradient descent optimization methods, so they cannot guarantee that the result

NeurIPS 2022 Workshop on Robot Learning: Trustworthy Robotics, Virtual, Virtual

satisfies the provided constraints. The methods presented in [6] and [7] rely on extending the trained policy architecture or producing decoupled networks, respectively, to modify the output of network only in the faulty linear regions of input space. The method proposed in [6] is only suitable to the classification tasks as it only satisfies the constraints for the faulty samples and does not consider the minimization of original loss function. It is also unclear how the approach scales for high dimensional inputs, since it requires partitioning the input space into affine subregions. The decoupling technique proposed in [7] causes the repaired network to be discontinuous, so it cannot be employed in robot learning and control. The application of [7] is also limited to the policies with less than three inputs. Finally, [8] only repairs the weights of final layer which drastically reduces the space of possible successful repairs (and mostly a repair is not even feasible).

Figure 2: Lower-leg prosthesis

We introduce NNRepLayer, a framework to certifiably repair a network policy to satisfy a set of given safety constraints with minimal deviation from the performance of original trained network. We particularly used our approach to derive controllers for a robotic lower-leg prosthesis that satisfy basic safety conditions, see Fig. 1. Given a set of safety constraints on the output of the trained policy for a set of input samples, NNRepLayer formulates a Mixed-integer Quadratic Programming (MIQP) to modify the weights of policy in a layer-wise fashion subject to the desired safety constraints. Applying NNRepLayer formally guarantees the satisfaction of constraints for the given input samples. Moreover, we propose an algorithm to employ NNRepLayer and a sound verifier in the loop that improves the adversarial accuracy of the trained policy (satisfaction of constraints for the inputs in a norm-bounded distance of the repaired samples). Note that the repair problem is fundamentally different from verification. Verification [9, 10] explores the input space of network to obtain tight bounds over the output while repair optimizes the NN parameters to ensure the satisfaction of constraints by the network's output.

Notation. We denote the set of variables $\{a_1, a_2, \cdots, a_N\}$ with $\{a_n\}_{n=1}^N$. Let π_θ be a network policy with L hidden layers. The nodes at each layer $l \in \{l\}_{l=0}^L$ are represented by x^l, where $|x^l|$ denotes the dimension of layer l (x^0 represents the input of network). The network's output $\pi_\theta(x^0)$ is denoted by y. We consider fully connected policy networks with weight and bias terms $\{(\theta_w^l, \theta_b^l)\}_{l=1}^{L+1}$. The training data set of N inputs x_n^0 and target outputs t_n is denoted by $\{(x_n^0, t_n)\}_{n=1}^N$ sampled from the input-output space $\mathcal{X} \times \mathcal{T} \subseteq \mathbb{R}^{|x^0|} \times \mathbb{R}^{|t|}$. The vector of nodes at layer l for sample n is denoted by x_n^l. In this work, we focus on the policy networks with the Rectified Linear Unit (ReLU) activation function $R(z) = \max\{0, z\}$. Thus, given the n^{th} sample, in the l^{th} hidden layer, we have $x^l = R\left(\theta_w^l x^{l-1} + \theta_b^l\right)$. The last layer is also represented as $y = \theta_w^{L+1} x^L + \theta_b^{L+1}$.

2 NNRepLayer

Without loss of generality, we motivate and discuss our approach using the task of learning safe robot controllers for a lower-leg prosthesis, see Fig. 2. The goal is to learn a policy π_θ which generates control values for the ankle angle of the powered prosthesis given a set of sensor values. The given policy may be optimized for task efficiency, e.g., stable and low-effort walking gaits, but may not yet satisfy any safety constraints Ψ. Our goal is to find an adjusted set of network parameters that satisfy any such constraint. We formulate our framework as the minimization of the loss function $E(\theta_w, \theta_b)$ subject to $(x^0, t) \in \mathcal{X} \times \mathcal{T}$ and $\Psi(y, x^0)$ for the inputs of interest $x^0 \in \mathcal{X}_r$. However, the resulting optimization is non-convex and difficult to solve due to the nonlinear forward pass of ReLU networks. Hence, we obtain a sub-optimal solution by just modifying the weights and biases of a single layer to adjust the predictions so as to minimize $E(\cdot)$ and to satisfy $\Psi(\cdot)$. The problem is defined as follows,

Problem Statement (Repair Problem). *Let π_θ denote a trained policy with L hidden layers over the training input-output space $\mathcal{X} \times \mathcal{T} \subseteq \mathbb{R}^{|x^0|} \times \mathbb{R}^{|t|}$ and $\Psi(y, x^0)$ denote a predicate representing constraints on the output y of π_θ for the set of inputs of interest $x^0 \in \mathcal{X}_r \subseteq \mathcal{X}$. NNRepLayer modifies the weights of a layer $l \in \{1, \cdots, L+1\}$ in π_θ such that the new network π_{θ_r} satisfies $\Psi(y, x^0)$ while minimizing the loss of network $E(\theta_w^l, \theta_b^l)$ with respect to its original training set.*

Since $\mathcal{X}_r$ and $\mathcal{X}$ are not necessarily convex, we formulate NNRepLayer over a data set $\{(x_n^0, t_n)\}_{n=1}^N \sim \mathcal{X} \times \mathcal{T} \cup \mathcal{X}_r \times \tilde{\mathcal{T}}$, where $\tilde{\mathcal{T}}$ is the set of original target values of inputs in $\mathcal{X}_r$. The predicate $\Psi(x^0, y)$ defined over $x^0 \in \mathcal{X}_r$ is not necessarily compatible with the target

$$
\begin{aligned}
(1)\quad & \min_{\theta_w^l,\theta_b^l,\delta,y_n,\{x_n^i\}_{i=l}^L,\{\phi_n^i\}_{i=l}^L} E(\theta_w^l,\theta_b^l)+\delta,\\
& \text{s.t.}\\
(2)\quad & y_n=\theta_w^{L+1}x_n^L+\theta_b^{L+1},\\
(3)\quad & x_n^i=R(\theta_w^i x_n^{i-1}+\theta_b^i), \quad \text{for } \{i\}_{i=l}^L\\
(4)\quad & \Psi(y_n,x_n^0), \quad \text{for } x_n^0\in\mathcal{X}_r\\
(5)\quad & \delta\geq\|\theta_w^l-\theta_w^{l,init}\|_\infty,\ \|\theta_b^l-\theta_b^{l,init}\|_\infty.
\end{aligned}
$$

Algorithm 1: NNRepLayer & Verifier

Input: $\pi_\theta^o, \mathcal{X}_r, \Psi$
Output: π_θ^r

$\pi_\theta^r \leftarrow \pi_\theta^o$
while $\mathcal{X}_r \notin \emptyset$ **do**
  $\pi_\theta^r \leftarrow \text{NNREPLAYER}(\pi_\theta^r, \mathcal{X}_r, \Psi)$
  $\mathcal{X}_r \leftarrow \text{VERIFIER}(\pi_\theta^r)$
end

values in $\tilde{\mathcal{T}}$. It means that the predicate may bound the NN output for $\mathcal{X}_r$ input space such that not allowing an input $x^0 \in \mathcal{X}_r$ to reach its target value in $\tilde{\mathcal{T}}$. It is a natural constraint in many applications. For a given layer l, we also define $E(\theta_w^l, \theta_b^l)$ in the form of sum of square loss $E(\theta_w^l, \theta_b^l) = \sum_{n=1}^N \|y_n(x_n^0, \theta_w^l, \theta_b^l) - t_n\|_2^2$, where $\|\cdot\|_2$ denotes the Euclidean norm. Since we only repair the parameters of target layer l, $\{(\theta_w^i, \theta_b^i)\}_{i=l+1}^{L+1}$ are fixed. We define NNRepLayer as follows.

NNRepLayer Optimization Formulation. Let π_θ be a policy with L hidden layers, $\Psi(y, x^0)$ be a predicate, and $\{(x_n^0, t_n)\}_{n=1}^N$ be an input-output data set sampled from $(\mathcal{X} \times \mathcal{T}) \cup (\mathcal{X}_r \times \tilde{\mathcal{T}})$. NNRepLayer minimizes the loss (1) by modifying θ_w^l and θ_b^l subject to the constraints (2)-(5).

Here, constraints (2) and (3) represent the linear forward pass of network's last layer and hidden layers starting from the layer l, respectively. Except the weight and bias terms of the l^{th} layer, i.e. θ_w^l and θ_b^l, the weight and bias terms of the subsequent layers $\{(\theta_w^i, \theta_b^i)\}_{i=l+1}^{L+1}$ are fixed. The sample values of x_n^{l-1} are obtained by the weighted sum of the nodes in its previous layers starting from x_n^0 for all N samples $\{n\}_{n=1}^N$. Each ReLU node x^l is formulated using Big-M formulation [11, 12] by $x^l \geq \theta_w^l x_n^{l-1} + \theta_b^l$, $x^l \leq \left(\theta_w^l x_n^{l-1} + \theta_b^l\right) - lb(1-\phi)$, and $x^l \leq ub\ \phi$, where $x^l \in [0, \infty)$, and $\phi \in \{0, 1\}$ determines the activation status of node x^l. The bounds $lb, ub \in \mathbb{R}$ are known as Big-M coefficients, $\theta_w^l x_n^{l-1} + \theta_b^l \in [lb, ub]$, that need to be as tight as possible to improve the performance of MIQP solver. We used Interval Arithmetic (IA) Method [13, 9] to obtain tight bounds for ReLU nodes (read Appx. A for further details on IA). Constraint (4) is a given predicate on y defined over $x^0 \in \mathcal{X}_r$. NNReplayer addresses the predicates of the form $\bigvee_{c=1}^C \psi_c(x^0, y)$ where C represents the number of disjunctive propositions and ψ_i is an affine function of x^0 and y. Finally, constraint (5) bounds the entry-wise max-norm error between the weight and bias terms θ_w^l and θ_b^l, and the original $\theta_w^{l,init}$ and $\theta_b^{l,init}$ by δ. Considering the quadratic loss function $E(\cdot)$ and the affine disjunctive forms of $\Psi(\cdot)$ and $R(\cdot)$, we solve NNRepLayer as a Mixed Integer Quadratic Program (MIQP). Any feasible solution to the NNRepLayer guarantees that for all input samples x_n^0 from $\{(x_n^0, t_n)\}_{n=1}^N$, $\Psi(\pi_{\theta_r}(x_n^0), x_n^0)$ is satisfied (for theorems and proofs read Appx. B).

Our technique only ensures the satisfaction of constraints for the repaired samples, so the satisfaction of constraints for the unseen adversarial samples is not theoretically guaranteed. To address this problem, we propose Alg. 1 that guarantees the satisfaction of constraints Ψ in $\mathcal{X}_r$. In this algorithm, NNReplayer is employed with a sound verifier [14, 15] in the loop such that our method first returns the repaired network π_θ^r. Then, the verifier evaluates the network. If the algorithm terminates, the network is guaranteed to be safe for all other unseen samples in the target input space. Otherwise, the network is not satisfied to be safe and the verifier provides the newly found adversarial samples $\mathcal{X}_r$ for which the guarantees do no hold. In turn, NNRepLayer uses the given samples by the verifier to repair the network. This loop terminates when the verifier confirms the satisfaction of constraints.

3 Evaluation

Since DNN may change the control more rapidly than what is feasible for the robotic prosthesis or for the human subject to accommodate, we propose an **Input-output constraint** over the possible change of control actions from one time-step to the next. This constraint should act to both smooth the control action in the presence of sensor noise, as well as to reduce hard peaks and oscillations in the control action. To capture this constraint as an input-output relationship, we trained a three-hidden-layer policy network with 32 ReLU nodes at each hidden layer. The network receives the

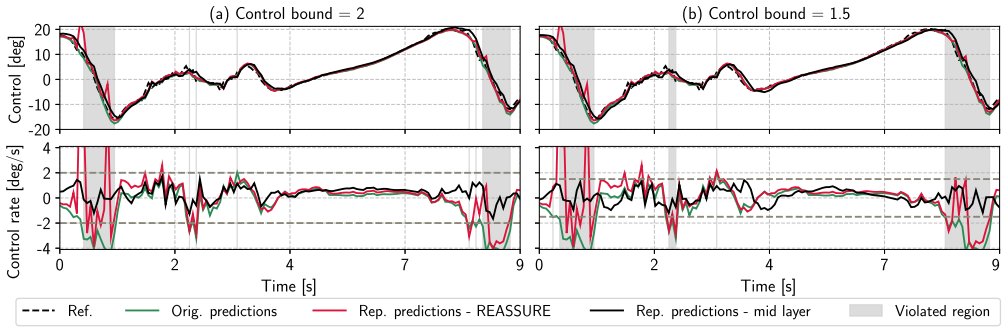

Figure 3: Bounding Ankle angles and Ankle angle rates for bounds (a) $\Delta\alpha_a = 2$ and (b) $\Delta\alpha_a = 1.5$.

previous dt control actions $\{\alpha_a(i)\}_{i=t-dt}^{t-1}$, and the angle and velocity from the upper and lower limb sensors, $\alpha_{ul}, \dot{\alpha}_{ul}, \alpha_{ll}, \dot{\alpha}_{ll}$ (network inputs x^0), respectively. The network then predicts the ankle angle α_a (network output y). Using NNRepLayer, we then bound the control rate by applying the constraint $\Delta\alpha_a$ by $|\alpha_a(t) - \alpha_a(t-1)| \leq \Delta\alpha_a^{max}$ in network repair (repairing a middle layer). In our tests, we bounded the control rate by $\Delta\alpha_a^{max} = 1.5$ [deg/s] and $\Delta\alpha_a^{max} = 2$ [deg/s]. Our simulation results in Fig. 3 demonstrate that NNRepLayer satisfies both bounds on the control rate which subsequently results in a smoother control output. It can also be observed that NNRepLayer successfully preserves the tracking performance of controller. Table 2 compares our method with fine-tuning, retraining [1–4], and REASSURE [6]. As shown in Table 2, retraining and NNRepLayer both perform well in maintaining the minimum absolute error and the generalization of constraint satisfaction to the testing samples. Comparing to [6], while REASSURE guarantees the satisfaction of constraints in the local faulty linear regions, we showed that this method significantly reduces the performance of network in the repaired regions, see Fig.3. Figure 3 also shows that [6] cannot address the given constraints for the faulty samples (introduces almost 500 times more faulty samples compared to our technique). Moreover, we tested Alg. 1 on a global constraint that ensures α_a stays within a certain range. We used the sound verifier proposed in [9] for the verification. To evaluate the algorithm, we used the adversarial accuracy metric (ACC_ϵ). For a given adversarial sample set $X_{adv} \subseteq X_r$, this metric evaluates the portion of samples that are robust to perturbations in the l_∞ ball of radius ϵ around each sample. Our method results $ACC_{0.5} = 100\%$ after one round of repair, $ACC_1 = 88\%$ after the second iteration, and $ACC_1 = 100\%$ after the third iteration. Finally, the largest network that we successfully repaired had 256 neurons in each hidden layer that took up to 10 hours. Similar network structure and sizes are frequently used in robotics and control tasks for example researchers in Google Brain trained a robot locomotion task using a network with 2 hidden layers and 256 nodes [16]. For details on the experimental setup and other results, read Appx. C.

Table 1: Repair Stats*. RT: runtime, MAE: Mean Absolute Error between the repaired and the original outputs, RE: the percentage of adversarial samples that are repaired, and IB: the percentage of test samples that were originally safe but became faulty after the repair.

	RT [s]	MAE	RE [%]	IB [%]
NNRepLayer	112 ± 122	0.5 ± 0.03	98 ± 1	0.19 ± 0.18
REASSURE [6]	30 ± 8	0.6 ± 0.03	19 ± 4	85 ± 5
Fine-tune	8 ± 2	0.6 ± 0.03	88 ± 2	2.47 ± 0.49
Retrain	101 ± 1	0.5 ± 0.04	98 ± 1	0.28 ± 0.32

*Average of 50 runs.

4 Conclusion

We introduced a framework for training neural network controllers that certifiably satisfy a formal set of safety constraints. Our approach, NNRepLayer, performs a global optimization step in order to perform layer-wise repair of neural network weights tested on a lower-leg prosthesis satisfying a variety of constraints. We argue that this type of approach is critical for human-centric and safety-critical applications of robot learning, e.g., the next-generation of assistive robotics.

Acknowledgment

This work was partially supported by the National Science Foundation under grants CNS-1932068, IIS-1749783, and CNS-1932189.

References

[1] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitry Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. In *International Conference on Learning Representations*, 2019.

[2] Xuhong Ren, Bing Yu, Hua Qi, Felix Juefei-Xu, Zhuo Li, Wanli Xue, Lei Ma, and Jianjun Zhao. Few-shot guided mix for dnn repairing. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 717–721. IEEE, 2020.

[3] Guoliang Dong, Jun Sun, Xingen Wang, Xinyu Wang, and Ting Dai. Towards repairing neural networks correctly. In *IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pages 714–725, 2021.

[4] Vincenzo Taormina, Donato Cascio, Leonardo Abbene, and Giuseppe Raso. Performance of fine-tuning convolutional neural networks for hep-2 image classification. *Applied Sciences*, 10 (19):6940, 2020.

[5] Xiaodong Yang, Tom Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T Johnson, and Danil Prokhorov. Neural network repair with reachability analysis. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 221–236. Springer, 2022.

[6] Feisi Fu and Wenchao Li. Sound and complete neural network repair with minimality and locality guarantees. In *10th International Conference on Learning Representations (ICLR)*, 2021.

[7] Matthew Sotoudeh and Aditya V Thakur. Provable repair of deep neural networks. In *42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 588–603, 2021.

[8] Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In *23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73, pages 260–278, 2020.

[9] Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[10] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, Mykel J Kochenderfer, et al. Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404, 2021.

[11] Pietro Belotti, Leo Liberti, Andrea Lodi, Giacomo Nannicini, Andrea Tramontani, et al. Disjunctive inequalities: applications and extensions. *Wiley Encyclopedia of Operations Research and Management Science*, 2:1441–1450, 2011.

[12] Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained relu neural networks. *Advances in Neural Information Processing Systems*, 34:3068–3080, 2021.

[13] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. Introduction to interval analysis/ramon e. *Moore, R. Baker Kearfott, Michael J. Cloud. Philadelphia*, 2009.

[14] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International conference on computer aided verification*, pages 3–29. Springer, 2017.

[15] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

[16] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems*, 2019.

[17] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[18] Ross J Cortino, Edgar Bolívar-Nieto, T Kevin Best, and Robert D Gregg. Stair ascent phase-variable control of a powered knee-ankle prosthesis. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5673–5678. IEEE, 2022.

[19] Chang Gao, Rachel Gehlhar, Aaron D Ames, Shih-Chii Liu, and Tobi Delbruck. Recurrent neural network control of a hybrid dynamical transfemoral prosthesis with edgedrnn accelerator. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5460–5466. IEEE, 2020.

[20] David C Morgenroth, Alfred C Gellhorn, and Pradeep Suri. Osteoarthritis in the disabled population: a mechanical perspective. *PM&R*, 4(5):S20–S27, 2012.

[21] Antonis Ekizos, Alessandro Santuz, Arno Schroll, and Adamantios Arampatzis. The maximum lyapunov exponent during walking and running: Reliability assessment of different marker-sets. *Frontiers in Physiology*, 9:1101, 2018. ISSN 1664-042X.

[22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL `https://www.gurobi.com`.

[23] Gabriel I Fernandez, Colin Togashi, Dennis W Hong, and Lin F Yang. Deep reinforcement learning with linear quadratic regulator regions. *arXiv preprint arXiv:2002.09820*, 2020.

[24] Christian Landgraf, Bernd Meese, Michael Pabst, Georg Martius, and Marco F Huber. A reinforcement learning approach to view planning for automated inspection tasks. *Sensors*, 21 (6):2030, 2021.

[25] Allison Pinosky, Ian Abraham, Alexander Broad, Brenna Argall, and Todd D Murphey. Hybrid control for combining model-based and model-free reinforcement learning. *The International Journal of Robotics Research*, page 02783649221083331, 2022.

[26] Matthieu Zimmer, Yann Boniface, and Alain Dutech. Developmental reinforcement learning through sensorimotor space enlargement. In *2018 Joint IEEE 8th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 33–38. IEEE, 2018.

[27] Morten B Kristoffersen, Andreas W Franzke, Raoul M Bongers, Michael Wand, Alessio Murgia, and Corry K van der Sluis. User training for machine learning controlled upper limb prostheses: a serious game approach. *Journal of NeuroEngineering and Rehabilitation*, 18(1):1–15, 2021.

[28] Babak Hassibi, David Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. *Advances in neural information processing systems*, 6, 1993.

[29] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

A Interval Arithmetic Method

To illustrate how we generated a tight valid bound for each ReLU activation node, we used the Interval Arithmetic method [13, 9]. Interval arithmetic is widely used in verification to find an upper and a lower bounds over the relaxed ReLU activations given a bounded set of inputs. We used the same approach to find the tight bounds over the ReLU nodes assuming the weights can only perturb inside a bounded l_∞ error with respect to the original weights. Assume we denote each input variable of repair layer L as $x^{L-1}(i)$, the weight term that connect $x^{L-1}(i)$ to $x^L(j)$ as $\theta_w^L(ij)$, and the bias term of nodes $x^L(j)$ as $\theta_b^L(j)$. Given the bounds for variables $\theta_w^L(ij) \in \left[\underline{\theta}_w^l(ij), \bar{\theta}_w^L(ij)\right]$ and $\theta_b^L(ij) \in \left[\underline{\theta}_b^L(ij), \bar{\theta}_b^L(ij)\right]$, the interval arithmetic gives the valid upper and lower bounds for $x^L(j)$ as

$$\bar{x}^L(j) = \sum_i \left(\bar{\theta}_w^L(ij)\max(0, x^{L-1}(i)) + \underline{\theta}_w^L(ij)\min(0, x^{L-1}(i))\right) + \bar{\theta}_b^L(ij), \text{ and}$$

$$\underline{x}^L(j) = \sum_i \left(\underline{\theta}_w^L(ij)\max(0, x^{L-1}(i)) + \bar{\theta}_w^L(ij)\min(0, x^{L-1}(i))\right) + \underline{\theta}_b^L(ij),$$

respectively. The bounds over the ReLU nodes in the subsequent layers $l = L+1, \cdots N$ are obtained as

$$\bar{x}^l(j) = \sum_i \left(\bar{x}^{l-1}\max(0, \theta_w^l(ij)) + \underline{x}^{l-1}\min(0, \theta_w^l(ij))\right) + \theta_b^l(ij),$$

$$\underline{x}^l(j) = \sum_i \left(\underline{x}^{l-1}\max(0, \theta_w^l(ij)) + \bar{x}^{l-1}\min(0, \theta_w^l(ij))\right) + \theta_b^l(ij).$$

B Theorems and Proofs

Theorem 1 (Soundness of NNRepLayer)**.** *Given the predicate $\Psi(y, x^0)$, and the input-output data set $\{(x_n^0, t_n)\}_{n=1}^N$ sampled from $(\mathcal{X} \times \mathcal{T}) \cup (\mathcal{X}_r \times \tilde{\mathcal{T}})$ over the sets $\mathcal{X}$, $\mathcal{X}_r$, $\mathcal{T}$, and $\tilde{\mathcal{T}}$, assume that θ_w^l and θ_b^l are feasible solutions to (1)-(5). Then, $\Psi(\pi_{\theta_r}(x_n^0), x_n^0)$ is satisfied for all input samples x_n^0.*

Proof. Since the feasible solutions θ_w^l and θ_b^l satisfy the hard constraint (4) for the repair data set $\{(x_n^0, t_n)\}_{n=1}^N$, $\Psi(\pi_{\theta_r}(x_n^0), x_n^0)$ is satisfied. □

Given Thm. 1, the following Corollary is straightforward.

Corollary 1. *Given the predicate $\Psi(y, x^0)$, and the input-output data set $\{(x_n^0, t_n)\}_{n=1}^N$ sampled from $(\mathcal{X} \times \mathcal{T}) \cup (\mathcal{X}_r \times \tilde{\mathcal{T}})$ over the sets $\mathcal{X}$, $\mathcal{X}_r$, $\mathcal{T}$, and $\tilde{\mathcal{T}}$, assume that $\theta_w^{l^*}$ and $\theta_b^{l^*}$ are the optimal solutions to the NNRepLayer (1)-(5). Then, for all input samples x_n^0 from $\{(x_n^0, t_n)\}_{n=1}^N$, $\Psi(\pi_{\theta_r}(x_n^0), x_n^0)$ is satisfied.*

Theorem 2 (Soundness of Alg. 1)**.** *Assume* VERIFIER() *is a sound verifier. If the Alg. 1 terminates, the predicate Ψ is satisfied by the repaired network π_θ^r.*

Proof. Given that VERIFIER() is assumed to be sound, if the algorithm terminates, $\mathcal{X}_r$ is empty which means VERIFIER() did not find other samples that violate Ψ. Therefore, the predicate Ψ is guaranteed to be satisfied by π_θ^r. □

C More Details on Experimental Results

C.1 Experimental Setup

We trained a policy network π_θ for controlling a prosthesis, which then undergoes the repair process to ensure compliance with the safety constraints. To this end, we first train the model using an imitation learning [17] strategy. For data collection, we conducted a study approved by the Institutional Review Board (IRB), in which we recorded the walking gait of a healthy subject without any prosthesis. Walking data included three inertial measurement units (IMUs) mounted via straps to the upper leg (Femur), lower leg (Shin), and foot. The IMUs acquired both the angle and angular velocity

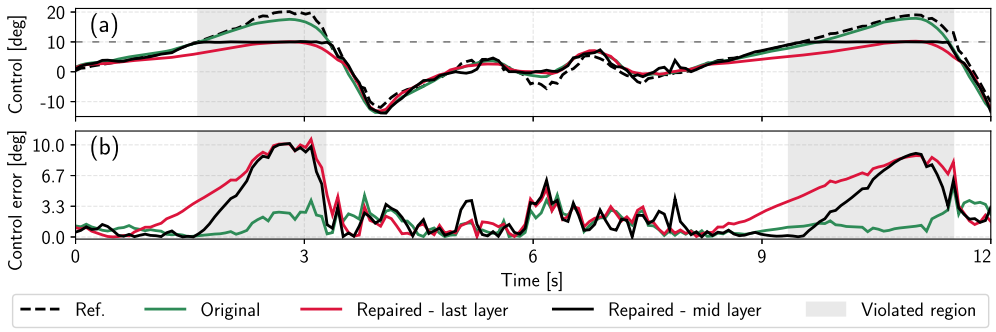

Figure 5: Global constraint: (a) ankle angle, α_a, (b) the error between the predicted and the reference controls.

of each limb portion in the world coordinate frame at 100Hz. Ankle angle α_a was calculated as a post process from the foot and lower limb IMUs. We then trained the NN to generate the ankle angle from upper and lower limb IMU sensor values. We used a sliding window of input variables, denoted as dt ($dt = 10$ in all our experiments), to account for the temporal influence on the control parameter and to accommodate for noise in the sensor readings. Therefore, the input to the network is $dt \times |x^0|$, or more specifically the current and previous dt sensor readings. After the networks were fully trained we assessed the policy for constraint violations and collected samples for NNRepLayer. We tested NNRepLayer on the last and the second to the last layer of network policy to satisfy the constraints with a subset of the original training data including both adversarial and non-adversarial samples. In all experiments, we used 150 samples in NNRepLayer and a held out set of size 2000 for testing. Finally, the repaired policies to satisfy global and input-output constraints are tested on a prosthetic device for 10 minutes of walking, see Fig. 4. More specifically, the same healthy subject was fitted with an ankle bypass; a carbon fiber structure molded to the lower limb and constructed such that a prosthetic ankle can be attached to allow the able-bodied subject to walk on the prosthesis. The extra weight and off-axis positioning of the device incline the individual towards slower, asymmetrical gaits that generates strides out of the original training distribution [18, 19]. The participant is then asked to walk again for 10 minutes to assess whether constraints are satisfied. Adversarial samples in the repair data set are hand-labeled for fine-tuning and retraining so that the target outputs satisfy the given predicates. In fine-tuning, as proposed in [1, 4], we used the collected adversarial data set to train all the parameters of the original policy by gradient descent using a small learning rate (10^{-4}). To avoid over-fitting to the adversarial data set, we trained the weights of the top layer first, and thereafter fine-tuned the remaining layers for a few epochs. The same hand-labeling strategy is applied in retraining, except that a new policy is trained from scratch for all original training samples. In both methods, we trained the policy until all the adversarial samples in the repair data set satisfy the given predicates on the network's output. Our code is available on GitHub: `https://github.com/k1majd/NNRepLayer.git`.

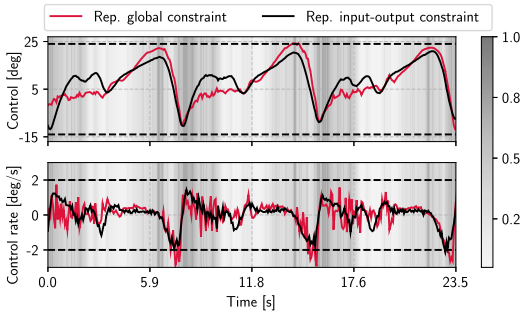

Figure 4: Real prosthesis walking test results for imposing the global constraint of $[-14, 24]$ to the control (shown in red) and bounding the control rate by 2 [deg/s] (shown in black). The color bar represents the normalized L_2-distance of each test input to its nearest neighbor in the repair set.

C.2 Experiments and Results

Global Constraint. The global constraint ensures that the prosthesis control, i.e., α_a, stays within a certain range and never outputs an unexpected large value that disturbs the user's walking balance. Additionally, the prosthetic device we utilized in these scenarios contains a parallel compliant mechanism. As such, either the human subject or the robotic controller could potentially drive the

Table 2: The table reports: RT: runtime, MAE: Mean Absolute Error between the repaired and the original outputs, RE: the percentage of adversarial samples that are repaired (Repair Efficacy), and IB: the percentage of test samples that were originally safe but became faulty after the repair (Introduced Bugs). The metrics are the average of 50 runs.

	NNRepLayer				REASSURE [6]			
	RT [s]	MAE	RE [%]	IB [%]	RT [s]	MAE	RE [%]	IB [%]
Global	233 ± 159	1.4 ± 0.11	99 ± 1	0.09 ± 0.20	14 ± 1	2.3 ± 0.78	97 ± 1	0
Input-output	112 ± 122	0.5 ± 0.03	98 ± 1	0.19 ± 0.18	30 ± 8	0.6 ± 0.03	19 ± 4	85 ± 5
Conditional	480 ± 110	0.35 ± 0.07	93 ± 2	0.11 ± 0.26	Infeasible	Infeasible	Infeasible	Infeasible
	Fine-tune				Retrain			
	RT [s]	MAE	RE [%]	IB [%]	RT [s]	MAE	RE [%]	IB [%]
Global	25 ± 13	1.2 ± 0.03	97 ± 4	0.95 ± 0.45	127 ± 30	1.4 ± 0.08	98 ± 3	0.65 ± 0.40
Input-output	8 ± 2	0.6 ± 0.03	88 ± 2	2.47 ± 0.49	101 ± 1	0.5 ± 0.04	98 ± 1	0.28 ± 0.32
Conditional	18 ± 3	0.7 ± 0.10	72 ± 5	0.27 ± 0.25	180 ± 2	0.31 ± 0.03	76 ± 2	0.12 ± 0.35

mechanism into the hard limits, potentially damaging the device. In our walking tests, see Fig. 4, we therefore specified global constraints such that the ankle angle stays within the bounds of $[-14, 24]$ [deg] regardless of whether it is driven by the human or the robot. In simulation experiments. see Fig. 5 (a), we enforced artificially strict bounds on the ankle angle α_a to never exceed $\alpha_a = 10$ [deg] which is a harder bound to satisfy.

Conditional Constraint. Depending on the ergonomic needs and medical history of a patient, the attending orthopedic doctor or prosthetist may identify certain body configurations that are harmful, e.g., they may increase the risk of osteoarthritis or musculoskeletal conditions [20, 21]. Following this rationale, we define a region $\mathcal{S}$ of joint angles space that should be avoided. An example of such a region is demonstrated in Fig. 6 as a grey box $\mathcal{S} = \{(\alpha_{ul}, \alpha_a) \mid \alpha_{ul} \in [-2, -0.5], \alpha_a \in [1, 3]\}$ in the joint space of ankle and femur angles. To satisfy this constraint the control rate should be tuned such that the joint ankle and femur angles stay out of set $\mathcal{S}$. This constraint can be defined as an if-then-else proposition $\alpha_{ul} \in [-2, -0.5] \implies (\alpha_a \in (-\infty, 1]) \vee (\alpha_a \in [3, \infty))$ which can be formulated as the disjunction of linear inequalities on the network's output. Figure 6 demonstrates the output of new policy after repairing with NNRepLayer. As it is shown, our method avoids the joint ankle and femur angles to enter the unsafe region S. Finally, we observed that repairing the last layer does not result in a feasible solution.

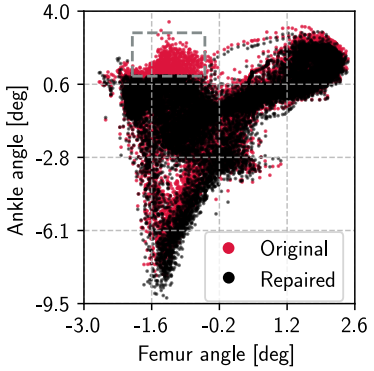

Figure 6: Enforcing the conditional constraints to keep the joint femur-ankle angles out of the grey box.

Comparison w\Fine-tuning, Retraining, and REASSURE. Table 2 better illustrates the success of our method in satisfying the constraints while maintaining the control performance. As shown in Table 2, retraining and NNRepLayer both perform well in maintaining the minimum absolute error and the generalization of constraint satisfaction to the unseen testing samples for global and input-output constraints. However, the satisfaction of if-then-else constraints is challenging for retraining and fine-tuning as the repair efficacy is dropped by almost 30% using these techniques. It also highlights the power of our technique in generalizing the satisfaction of conditional constraints to the unseen cases.

C.3 Testing Repair on Larger Networks

To demonstrate the scalability of our method, we conducted a repair experiment on a network with 256 neurons in each hidden layer. We used 1000 samples for repair and 2000 samples for testing. We formulate this problem in MIQP and run the program on a Gurobi [22] solver. We terminated the solver after 10 hours and report the best found feasible solution. Figure 7 and Table 3 show the control signal, and the statistical results of this experiment, respectively. As demonstrated, our

technique repaired a network with up to 256 nodes with 100% repair efficacy in 10 hours. Similar network structure and sizes are frequently used in robotics and controls tasks. Examples include [23] (3 hidden layer, 256 nodes), [24] (2 hidden layer, 64 nodes), [25] (2 hidden layer, 200 nodes), [26] (2 hidden layer, 50 nodes), and [27] (2 hidden layer, 50 nodes).

Table 3: Experimental results for repairing a network with 256 nodes in each hidden layer for the input-output constraint repair, maximum ankle angle rate of 2 [rad/s]. The table reports the size of network, the number of samples, the Mean Absolute Error (MAE) between the repaired and the original outputs, the percentage of adversarial samples that are repaired (Repair Efficacy), and the runtime.

	Network Size	Number of Samples	MAE	Repair Efficacy [%]	Runtime [h]
Input-output Constraint	256	1000	0.62	100	10

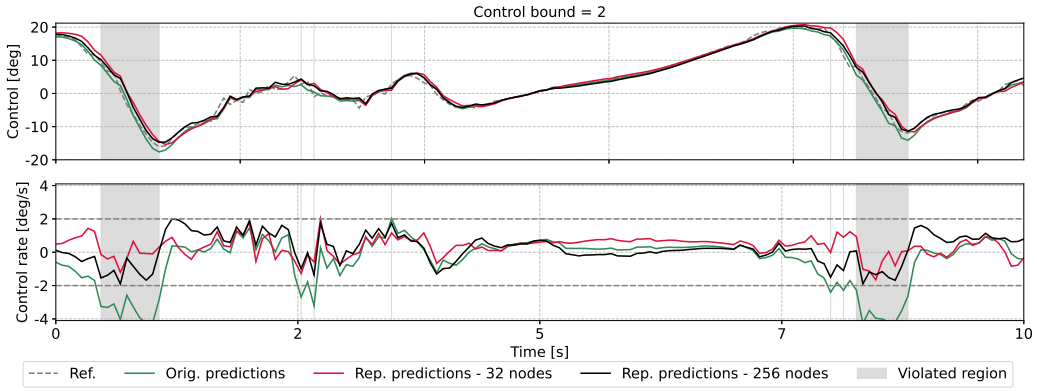

Figure 7: Input-output constraints for networks with 32 (red) and 256 (black) nodes in each hidden layer: Ankle angles and Ankle angle rates for bounds $\Delta\alpha_a = 2$.

C.4 Heuristics for Computational Speedups of NNRepLayer

In this section we provide examples for computational speedups and heuristics that lead to faster neural network repair. To this end, we repaired randomly selected nodes of a single hidden layer in a network with 64 hidden nodes for 35 times. We let the solver run for 30 minutes in each experiment (versus the full repair that is solved in 6 hours). Figure 8 demonstrates the mean absolute error (MAE), the total number of repaired weights, repair efficacy, and the original MIQP cost. Here, to detect the sparse nodes that can satisfy the constraints, we solved the original full repair by adding the l_1 norm-bounded error of repaired weights with respect to their original values to the MIQP cost function. The bold bars in Fig. 8 demonstrate the results of repairing the 10 randomly-selected sparse nodes. Repairing of the obtained sparse nodes reached a cost value very close to the cost value of the originally full repair problem (42.99 versus 40.29), **in only 30 minutes** versus 6 hours. As illustrated, repair of some random nodes also results in infeasibility (blank bars) that shows these nodes cannot repair the network. In our future work, we aim to explore the techniques such as neural network pruning [28, 29] to select and repair just the layers and nodes that can satisfy the constraints instead of repairing a full layer. Our results in this experiment show that repairing partial nodes can significantly decrease the computational time of our technique.

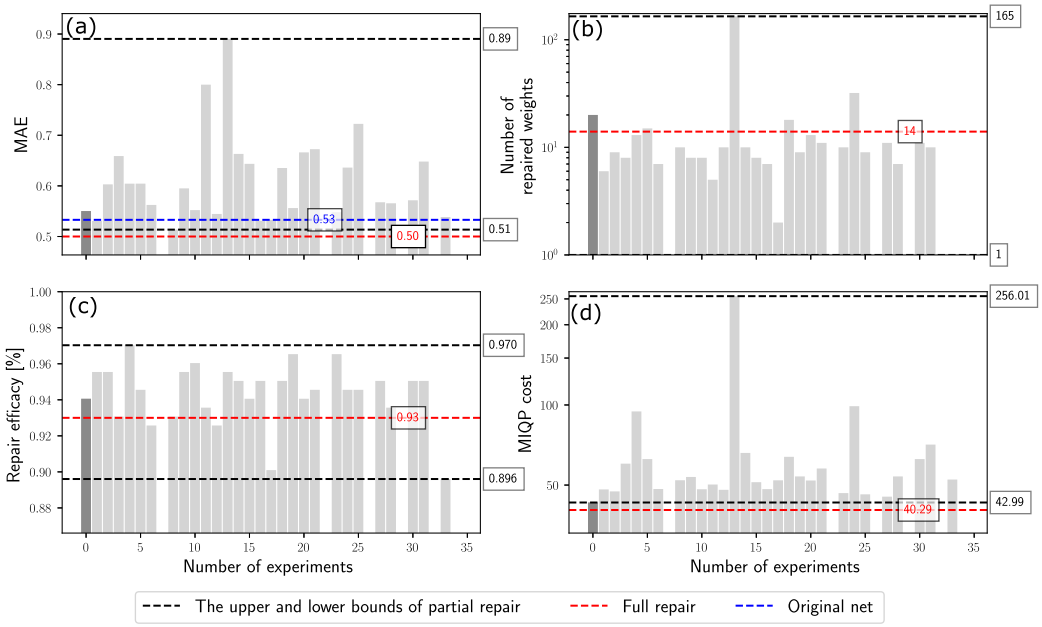

Figure 8: Partial repair versus full repair: we repaired 10 randomly selected nodes for 30 minutes in a network with 64nodes in each hidden layer (a) mean absolute error (MAE), (b) the total number of repaired weights, (c) repair efficacy, and (d) MIQP cost. We performed this random selections for 35 times.