Application of deep and reinforcement learning to boundary control problems

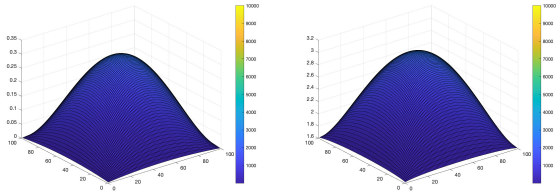
Zenin Easa Panthakkalakath,[1] **Juraj Kardoš,** [1] **Olaf Schenk** [1]

[1] Università della Svizzera italiana
zenin.easa.panthakkalakath@usi.ch, juraj.kardos@usi.ch, olaf.schenk@usi.ch

Abstract

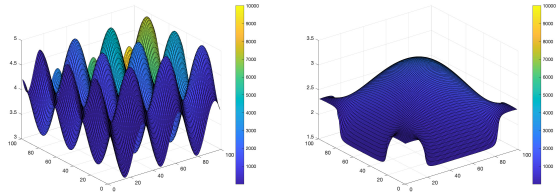
The boundary control problem is a non-convex optimization and control problem in many scientific domains, including fluid mechanics, structural engineering, and heat transfer optimization. The aim is to find the optimal values for the domain boundaries such that the enclosed domain adhering to the governing equations attains the desired state values. Traditionally, non-linear optimization methods, such as the Interior-Point method (IPM), are used to solve such problems.

This project explores the possibilities of using deep learning and reinforcement learning to solve boundary control problems. We adhere to the framework of iterative optimization strategies, employing a spatial neural network to construct well-informed initial guesses, and a spatio-temporal neural network learns the iterative optimization algorithm using policy gradients. Synthetic data, generated from the problems formulated in the literature, is used for training, testing and validation. The numerical experiments indicate that the proposed method can rival the speed and accuracy of existing solvers. In our preliminary results, the network attains costs lower than IPOPT, a state-of-the-art non-linear IPM, in 51% cases. The overall number of floating point operations in the proposed method is similar to that of IPOPT. Additionally, the informed initial guess method and the learned momentum-like behaviour in the optimizer method are incorporated to avoid convergence to local minima.



Desired Optimal solution

(a) Simple monotonous function



Desired Optimal solution

(b) Highly nonlinear function

Figure 1: Governing function of the desired profile and optimal solution obtainable

Introduction

Optimal control problems arise in a plethora of applications, many requiring accurate and fast solution methods. Using traditional solution methods is time-consuming; some simulations often take days. Performance improvement of such solvers leads to reduced time consumption, enabling more opportunities to fine-tune parameters.

Boundary control problems involve controlling the values at the boundaries while adhering to control and state constraints. The objective is to find the optimal boundary values such that the values in the domain are as close as possible to their desired values, with the closeness defined using residual sum of squares. For simple desired profiles, matching the domain values may be attainable, as in Figure 1a, while for complicated ones, the best achievable could be far from the desired, as in Figure 1b.

Many classical approaches and tools use Interior-point method (IPM), a non-linear optimization method, to solve such an optimization problem. IPMs typically require factorization and the solution of a linear system involving the Hessian matrix in each iteration. For large-scale problems, computation of the factorization can be computationally expensive and requires excessive storage, thus dominating the overall complexity of the algorithm. Additionally, as non-convex optimization problems can have multiple local minima, IPMs are not guaranteed to find a global minimum as these are sensitive to the starting point. These limitations serves as the motivation for exploring methodologies in deep learning and reinforcement learning to solve such problems.

Literature review

The paper by Maurer and Mittelmann (Maurer and Mittelmann 2000) presents control problems for semi-linear elliptic equations subject to control and state constraints. They approached the problem by transforming the control problem into a non-linear programming problem and used

the interior-point method to solve the same. An analytical setting for optimal boundary control of one-dimensional heat equations is described by Lang and Schmitt (Lang and Schmitt 2023). Non-elliptic boundary control problems using the Cahn-Hilliard equation as governing PDE exist in literature (Colli, Gilardi, and Sprekels 2015).

A plethora of studies on using deep learning and reinforcement learning to expedite numerical computing tasks exist. Scientific simulations involving solving partial differential equations (PDEs) can be solved using neural networks (Lagaris, Likas, and Fotiadis 1998; Jiang et al. 2023; Brandstetter, Worrall, and Welling 2022; Chamberlain et al. 2021; Cai and Diagne 2021). Fawzi et al. (Fawzi et al. 2022) used reinforcement learning to arrive at a matrix multiplication method that improves upon Strassen's algorithm, which was considered the most optimized for the past fifty years. Deep reinforcement techniques are used on optimization and control problems (Williams 1992; Zhu 2018). Attempts to analyze and improve optimizer methods following gradient descent are researched (Bello et al. 2017; Liu et al. 2023). A few papers describing the use of graph neural networks in control are of interest (Chen et al. 2021; Shen et al. 2019; Meirom et al. 2021; Rozemberczki et al. 2021; Gama and Sojoudi 2021). Observe a combination of Q-learning with policy-gradients with soft actor-critic method in the work by Haarnoja et al. (Haarnoja et al. 2018). These methods are combined to solve real-world problems (Bonny, Kashkash, and Ahmed 2022).

Control and optimization studies without deep learning or reinforcement learning also exist. Zafar and Manchester (Zafar and Manchester 2023) discusses a preconditioned conjugate gradient method for solving finite-horizon linear-quadratic optimal control problems. Several studies and tools using the interior-point method can be observed in the literature (Tasseff et al. 2019; Kardoš, Kourounis, and Schenk 2020; Kardoš et al. 2022; Pacaud et al. 2023; Wächter and Biegler 2006).

Methodology

High-level architecture

Copious iterative optimization algorithms have a common feature of starting with an initial guess and iteratively updating the current point until reaching the best-fitting solution. As depicted in Figure 2, the proposed method first makes an informed initial guess after considering problem parameters using deep learning, and subsequently, it iteratively updates the solution using an optimizer created using policy gradient reinforcement learning. From this point forward, the part of the proposed method that makes an informed intial guess is referred to as *initial guess method* and that which performs iterative update is called *optimizer method*.

Defining cost and reward

Mathematically, the boundary control problem is defined as:

$$\underset{y,u}{\textbf{Minimize}} \ \frac{1}{2}\int_{\Omega}(y(x)-y_d(x))^2\,dx+\frac{\alpha}{2}\int_{\Gamma}(u(x)-u_d(x))^2\,dx$$

$$\textbf{Subject to } \nabla^2 y = c,\ \ y_{min} < y < y_{max},\ \ u_{min} < u < u_{max}$$

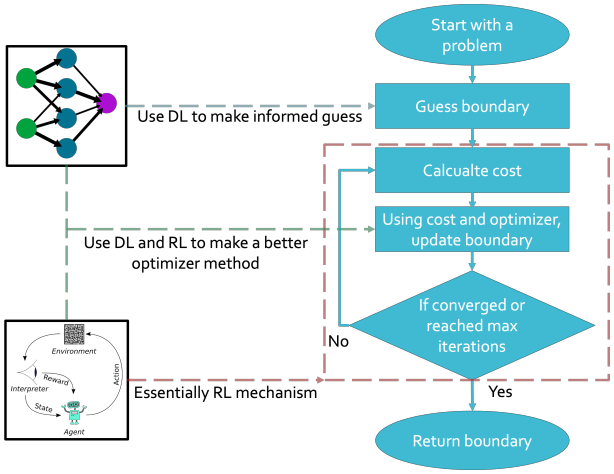


Figure 2: High-level solution architecture

where Ω is the domain, Γ is the boundaries, y_d is the desired values for the domain, u_d is the desired values for the boundaries, y is the values for the domain, u is the values for the boundaries, α is a non-negative constant that determines how much weight must be given to the cost for the boundaries, c is a constant sourcing term, y_{min} and y_{max} are upper and lower bound of y, and, u_{min} and u_{max} are upper and lower bound of u.

There are constraints associated with the objective function. To deal with the governing PDE, structure the proposed method to output the boundary values and calculate the domain values using a numerical PDE solver. Along with the objective function, the bound constraints are incorporated into the cost function using a barrier function. The cost function is mathematically defined as:

$$F = F_o + \beta F_v$$

$$F_o = \frac{1}{2}\int_{\Omega}(y(x)-y_d(x))^2\,dx+\frac{\alpha}{2}\int_{\Gamma}(u(x)-u_d(x))^2\,dx$$

$$F_v = \int_{\Omega} f_{\Omega}(x)dx + \int_{\Gamma} f_{\Gamma}(x)dx$$

$$f_{\Omega}(x) = \begin{cases} (y(x)-y_{min})^2 & \text{if } y(x) \in (-\infty, y_{min}) \\ (y(x)-y_{max})^2 & \text{if } y(x) \in (y_{max}, \infty) \\ 0 & \text{otherwise} \end{cases}$$

$$f_{\Gamma}(x) = \begin{cases} (u(x)-u_{min})^2 & \text{if } u(x) \in (-\infty, u_{min}) \\ (u(x)-u_{max})^2 & \text{if } u(x) \in (u_{max}, \infty) \\ 0 & \text{otherwise} \end{cases}$$

where β is the penalty factor for constraint violation, which should have a large positive value.

The reward could be any strictly monotonous decreasing function of the cost function, as maximizing it leads to minimizing the cost. For simplicity, the reward function shall be the negative of the cost function.

Data generation

We scope our study to boundary control problems with Dirichlet boundary conditions. Due to the unavailability of

a ready-to-use dataset containing real-world information, an equation generator is created based on four boundary control problems with Dirichlet boundary condi- tions presented in the literature (Maurer and Mittelmann 2000). To randomly create reasonable problems, it must be ensured that the parameters and coefficients can only vary within a well-defined range, as detailed below.

- **Domain size**: A random integer between 10 and 100.
- **Alpha factor**: Set to 0.01 for all problems.
- **Target profile equation**: The desired values in the domain are calculated using this equation. It shall have quadratic and sine-squared terms in both directions; the coefficients for each of the quadratic terms are integer values between -5 and 5 and the frequency and the phase angles of the sine squared terms are respectively set to be an integral multiple of π between -5 and 5, and, an integral fraction of π between 1 and 6. Examples:
  - $x_2^2 - x_2 + \sin^2(2\pi x_1 + \frac{\pi}{5})$
  - $\sin^2(3\pi x_1 + \frac{\pi}{6}) + \sin^2(\pi x_2 + \pi)$
  - $2x_2^2 + x_2$
- **Desired boundary values**: Set to zero for all problems.
- **Bounds**: The domain lower bound in all of the problems in the original set of problems was -10^{20}, which signifies negative infinity, which is used in the problems generated here as well. The rest of the bound values are chosen based on the target profile equation; by generating the desired domain profile by solving the target profile equation for the domain size. The maximum, minimum and median values in the desired domain are used.

  The following are the specifications of the Bounds
  - *Lower bound for domain* is set to -10^{20} for all problems.
  - *Upper bound for domain* is a uniformly sampled random number between the median and the maximum values.
  - *Lower bound for boundary* is the minimum value plus half of uniform random value generated between positive and negative difference between maximum and minimum values.
  - *Upper bound for boundary* is the maximum value plus half of uniform random value generated between positive and negative difference between maximum and minimum values.
- **Sourcing term**: A random value chosen from the set $\{0, -10, -20, -30, -40, -50\}$.
- **Additional filters**: Once the problem is generated, there are two additional sets of filters based on certain thresholds that were incorporated to improve the quality of the equations generated. The first one is a threshold for the maximum and minimum values observed in the domain; a difference of less than 0.3 is discarded. The second one is a threshold on the cost that IPOPT predicts for the generated problem; a problem with a cost of more than 0.2 per cell is discarded.

Experimental setup

Setting up the baselines

Two separate sets of baselines are defined to assess the effectiveness of the initial guess and optimizer methods.

Baselines for initial guess method Three baselines for initial guess are created with the boundaries set to the mean, the median or the values at the edges of the desired domain, and calculating the cost. This is done for all the generated problems, and the information is stored to make it easier for analysis later.

Baselines for optimizer method Two baselines are used to evaluate the optimizer method.

1. *Comparison against solvers previously tried and tested to work for boundary control problems*. The state-of-the-art large-scale nonlinear optimization problem solver, IPOPT (Wächter and Biegler 2006), is used to create this baseline.
2. *Comparison against optimizers used in deep learning*. On a network with only a single bias layer with the size of the boundaries, backpropagation is performed using the gradients computed from cost calculation. After several iterations, the bias layer is expected to achieve an optimal boundary value. Two different baselines are generated using this approach with Stochastic gradient descent (SGD) and adaptive moment estimation (Adam) as optimizers, and running for 100 optimizer steps.

Note that the former baseline method is computationally less expensive and can run for all the generated problems. However, since the latter is computationally expensive, the baseline is generated with only 400 problems.

Performing the experiments

For both initial guess and optimizer methods, neural networks are designed based on intuition, which may involve some feature engineering. The dataset is divided in the ratio 80:10:10 for training, validation and testing. The best model for a given network design is chosen based on the lowest validation cost, while the overall best network design is selected based on the lowest testing cost. The idea is to iteratively design, train, validate and test different networks until models that outperforms the baselines are found.

Initial guess method Inputs to the initial guess method are the desired domain profile, bound constraints and the sourcing term, and the output is the guessed boundary values. While designing the network, ensure that the input and output dimensions are variable and the output dimension is a function of input dimensions. If the target profile array is of size $N \times N$, then the output boundary value size would be $4 \times N$. Intuitively, one could use convolution, graph convolution and different pooling layers in the network. Indeed, there may be other layers that could work with this constraint. A fully connected layer, a.k.a. linear layer, can not work with such a constraint as it requires the input and output dimensions to be known beforehand to work.

Optimizer method Inputs to the optimizer method are the boundary values and the calculated gradients with respect to the cost, and the output is updated boundary values. All of these have size $4 \times N$. Inspired by several widely used optimizers, the network may have the ability to work with both spatial and temporal information. The temporal features would help in taking the previous iteration into consideration. Hence, layers like convolution, graph convolution, recurrent and other layers that can work with spatial and temporal information shall be employed.

Results

Initial guess method

Architecture After designing and evaluating several networks, the architecture of the best one obtained is shown in Figure 3. Starting with the desired domain profile (a 2-

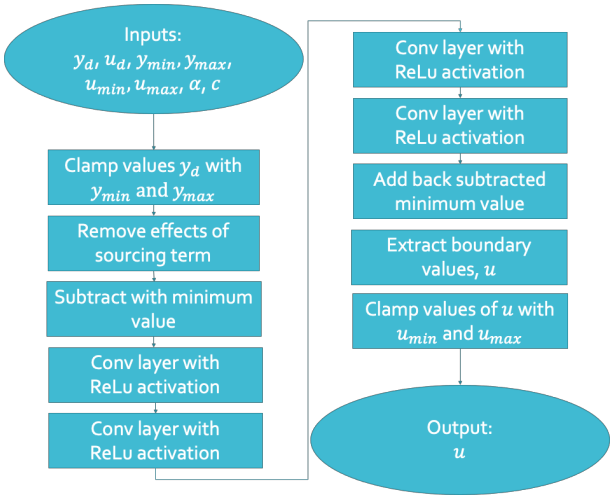
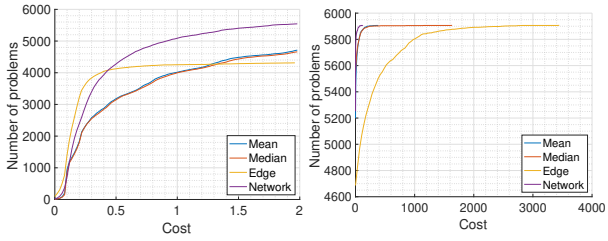


Figure 3: Network architecture for initial guess method

dimensional array), the values are clamped to the domain's upper and lower bounds. From this, the sourcing term effect is subtracted. Following this, to ensure that all the values are greater than or equal to zero, the minimum value in the array is determined, and all the elements are subtracted with this minimum value. The resulting array passes through four convolution layers with rectified linear unit activations, following which the previously subtracted minimum value is added back. The boundaries of the resulting array are extracted and clamped to the boundary's upper and lower bounds. The result forms the guessed boundary values.

Quantitative Table 1 summarizes the comparison between the baselines and the initial guess method filtered for the problems for which IPOPT found feasible solutions.

A cumulative histogram plot with information from the baselines and the network is shown in Figure 4. The plot depicts the count of problems with costs less than or equal to the value in the X-axis at any given point. Observe that there are about 4000 problems for which using edge values may be more advantageous than the other three. However, after the



(a) Trimmed till cost value of 2 (b) Larger cost values

Figure 4: Cumulative plot comparing cost for different methods for making initial guess

cost value mark of about 0.42, the method appears to have the upper hand. From the sub-plot for larger cost values, it is evident that the edge method performs significantly worse than the other three methods at later points as it goes slowly into the regions with more and more cost.

Although it is evident that the proposed initial guess method performs better than baselines by mean and median values of the desired profile, it is inconclusive as to whether it is better than that of the edge values. However, it can be argued that using the edge values as a starting point is unreliable as it could be the best or the worst in different situations; therefore, the use of the more consistent initial guess method is advisable.

Optimizer method

Architecture Figure 5 shows the architecture of the best method obtained after several iterations. The method en-

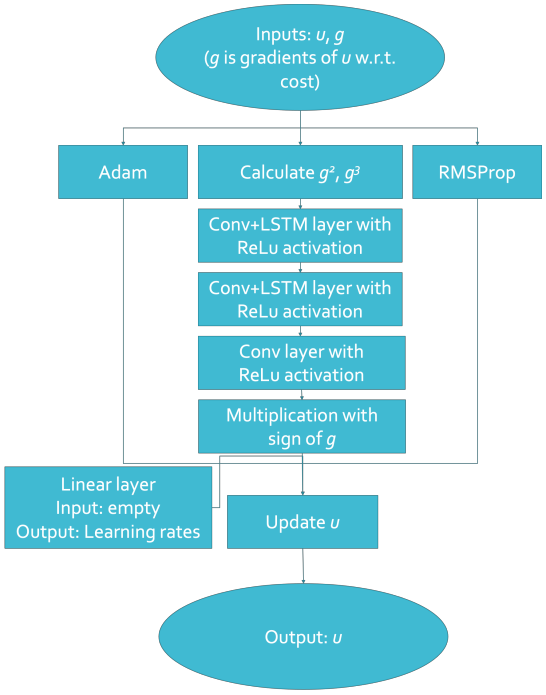


Figure 5: Network architecture for optimizer method

	Initial guess				Iterative optimization				
	Mean	**Median**	**Edge**	**Method**	**IPOPT**	**SGD**	**Adam**	**Method**	
Mean cost	4.5059	5.6269	87.1563	1.0591	0.1223	8378.6	0.1808	0.1721	0.1223
Median cost	0.4422	0.4572	0.2010	0.2709	0.1252	0.3439	0.1672	0.1469	0.1181
Lowest cost	0.0087	0.0091	0.000023215	0.0081	0.0097	0.0093	0.0164	0.0100	0.0034
Highest cost	388.7509	1642.7	3478.2	125.8895	0.1995	126160	0.5770	1.0894	0.6616
Iterations					43*	100	100	8	32

* The number of iterations for IPOPT is the rounded mean value

Table 1: Summary of method costs compared with the baselines

capsulates the Adam optimizer, RMSProp optimizer and a custom spatio-temporal network. The gradient is the input to these three, and the outputs are some intermediary values, which are scaled using three separate learnable learning rates and used to update the boundaries.

The spatio-temporal part begins by concatenating the gradients, and their squared and cubed values, which is passed through a couple of custom temporal convolution layers with rectified linear unit activation and, finally, a convolution layer with rectified linear unit activation. This intermediary output is multiplied with the sign of the original gradient values. Note that the custom temporal convolution layer encapsulates convolution and LSTM layers.

Quantitative Out of the 400 problems selected, IPOPT found feasible solutions for 252. The costs corresponding to these problems by IPOPT, SGD and Adam are used as baselines. Comparison between the baselines and the optimizer method run for 8 and 32 steps are summarized in Table 1. During training and validation, the network ran for 8 steps. The results show that the method achieves lower cost after 32 iterations, indicating it is indeed iteratively minimising the cost.

A cumulative histogram plot comparing the baselines with the optimizer method is shown in Figure 6. Observe the

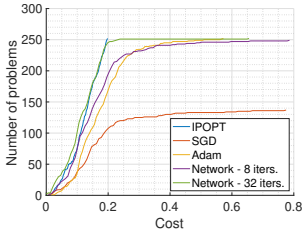


Figure 6: Cumulative plot comparing cost for different optimizers with proposed optimizer method

fiece competition between IPOPT and the optimizer method running 32 iterations; the lines corresponding to these interset quite a few times.

In a head-to-head comparison between the optimizer method and the baselines, it beats IPOPT, SGD and Adam-based baselines 127, 226 and 244 times, respectively, out of 252 cases, which means that the network beats the respective baselines 50.40%, 89.68% and 96.83% of the times. The method arrived at is indeed quite good.

Analysis

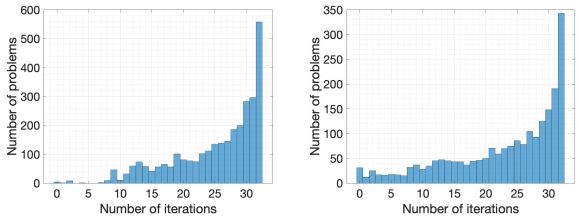
A detailed analysis is performed using all 10000 problems and comparing exclusively against IPOPT.

Simple statistics

Out of the 10000 problems generated, IPOPT found feasible solutions in 5907 cases. Among these cases, the cost predicted by the proposed method is lower than IPOPT in 3011 cases, which is 50.97% cases. However, we use a slightly relaxed barrier function for cost calculation. The maximum and mean absolute constraint violation for a particular cell in the domain or the boundary is 0.1033 and 8.3×10^{-4}. The total number of cases wherein the method predicted a feasible solution with zero constraint violation is 2127, which is 36.01% cases.

Iteration at which the network finds the best solution

Figure 7 shows histograms that help visualize the iteration at which the network finds the lowest cost for different problems. Observe the general trend of obtaining lowest cost



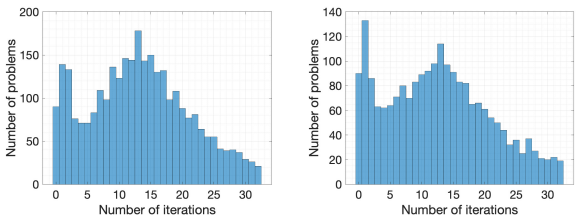
(a) With/without constraint violation (b) Without constraint violation

Figure 7: Histograms depicting the number of problems for which the method achieved the lowest cost at which iteration

at later iterations. The median number of iterations for the lowest cost is 28 and 26, respectively, for with and without constraint violation. Furthermore, the tremendous increase in the bar size at iteration number 32 may suggest that lower costs are obtainable with more iterations for some problems.

Iteration at which the network beats IPOPT

Histograms that help in visualizing the iteration at which the network achieves lower cost than IPOPT are shown in Figure 8. There are two peaks in both histograms. The former



(a) With/without constraint violation (b) Without constraint violation

Figure 8: Histograms depicting the number of problems for which the method first surpassed IPOPT's cost at which iteration

peak attributes the contribution of the initial guess method to the overall results, while the latter shows the prowess of the optimizer method. The median number of iterations to beat IPOPT's cost is 13 and 12, respectively, with and without constraint violation. Note that this plot is not informative in cases where the network has a higher cost than IPOPT.

Comparing cost and violation with IPOPT

A semi-transparent scatter plot with the cost by IPOPT in one axis and cost by proposed method in the other is shown in Figure 9. The line of equality (in orange) coincides with

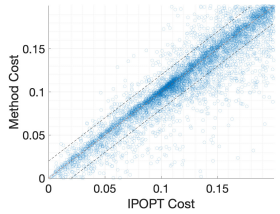
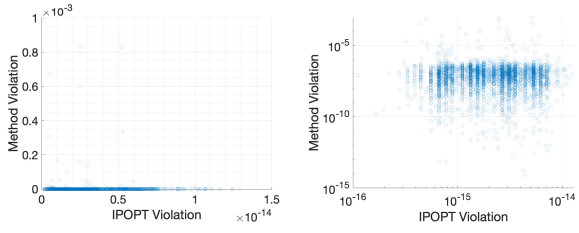


Figure 9: A scatter plot comparing costs

many scatter plot points, indicating the cost by IPOPT and method to be equal. 4607 points lie between two parallel lines to the line of equality with X and Y intercepts of 0.02, respectively, which is nearly 78% of all the points.

Figure 10 shows a scatter plot for the violations made by IPOPT and the method. The violation is zero in 1581 cases



(a) Standard axes (b) Log-log axes

Figure 10: Scatter plots comparing constraint violations

for the proposed method, which are not visible in the logarithmic version of the plot. However, there are cases wherein the violations made by the optimizer method are orders of magnitude larger than what is by IPOPT; much of that by IPOPT is in the range between 10^{-15} and 10^{-14}, whereas the same by the optimizer method is in the range between 10^{-8} and 10^{-6}.

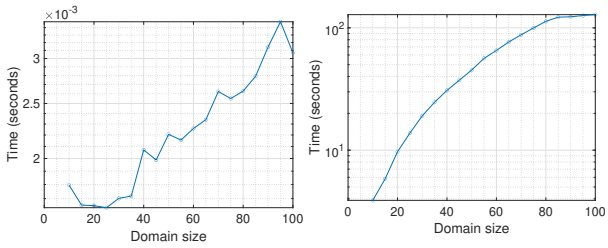
Performance comparison with IPOPT

Comparing the performance is a bit tricky for two reasons.

1. There are cases where the method achieves lower cost at very early iterations, and there are cases where it does not accomplish the same even after the maximum number of iterations.
2. The proposed method and IPOPT are implemented in Python and C++, respectively. C++ is between 10 to 100 times faster than Python.

Therefore, take the following analysis with a pinch of salt.

Timing The execution time for initial guess and optimizer methods for varying domain sizes are shown in Figure 11. Python's built-in'time' module was used for this. Observe



(a) For initial guess method (b) For optimizer method

Figure 11: Execution time for the initial guess method and 32 iterations of the optimizer method

that the initial guess method takes in the order of 10^{-3} seconds to execute, which is negligible compared to the execution time of the optimizer method, which is in the order of 10^2 seconds.

A comparison between the execution time by the proposed method and IPOPT for 32 iterations is shown in Figure 12. If implemented in a lower-level language, the

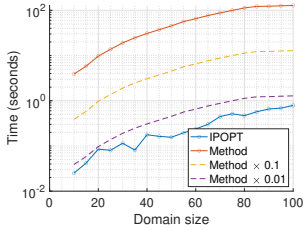
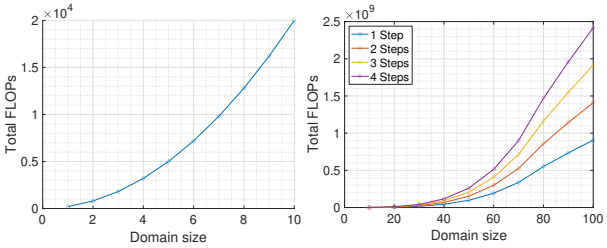


Figure 12: Comparing execution time for the proposed method with IPOPT

method's execution time would be between 0.1 and 0.01

times the current. These are shown in the plot as well. Even the optimistic scaled version would be slower than IPOPT for the same number of iterations.

Floating point operations The number of floating point operations involved in the computation of the initial guess and the optimizer methods are shown in Figure 13. This is calculated using PyTorch's profiler. Observe a quadratic



(a) For initial guess method (b) For optimizer method

Figure 13: FLOPs for the initial guess method and the optimizer method

increase in the FLOPs for increasing domain size for both the initial guess and the optimizer methods. The number of FLOPs for the initial guess method is negligible compared to that for the optimizer method; the former is in the order of 10^4 while the latter is in the order of 10^9.

The comparison of the number of floating point operations between the proposed method and IPOPT for 32 iterations is shown in Figure 14. Observe that the proposed

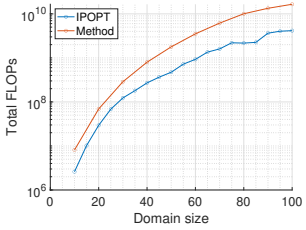


Figure 14: Comparing the number of FLOPs for the proposed method with that for IPOPT

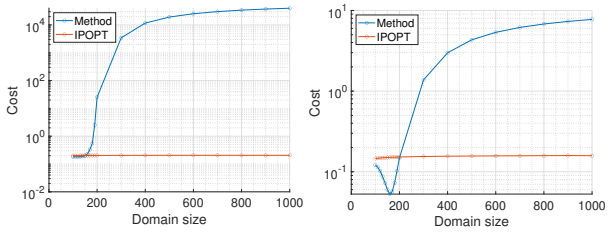
method has slightly more floating point operations than IPOPT for the same number of iterations and hence, will likely be slower even if implemented in C++.

Effect of larger domain sizes on accuracy

The data generated has domain sizes varying between 10 and 100. Figure 15 shows the cost calculated by the method and IPOPT for selected problems for domain sizes beyond this range. Although the method may work till about a domain size of 150, it fails miserably for larger domain sizes.

Conclusions and discussions

This study applies deep learning and reinforcement learning to boundary control problems by following an architecture framework, similar to most iterative optimization algorithms, containing two parts: one for making an initial guess,



(a) Problem index 0 (b) Problem index 17

Figure 15: Cost for larger domain sizes

called the initial guess method, and the other, for making iterative improvement, called the optimizer method. The initial guess method is predominantly a convolution neural network. The optimizer method is a policy-gradient reinforcement learning method with the policy represented by a spatio-temporal neural network. Experimentation and analysis suggest that both initial guess and optimizer methods are effective. The initial guess method successfully outperforms trivial methods of generating an initial guess value for a given problem. The optimizer method achieves similar accuracy and performance to IPOPT, arriving at a lower cost solution in 50.97% cases.

However, further research and exploration are required to provide additional conclusions and address the limitations of this research. These are summarized below.

- Evaluating the method on real-world problems is in order. In future work, we aim to tackle problems from fluid mechanics, structural engineering and heat transfer domains by creating a dataset with real-world conditions and retraining the model to study its performance.
- Explore additional model extensions addressing the suboptimal performance on larger domain sizes and employ mitigation strategies.
- Strategies to reduce the computations per iteration and the overall number of iterations without affecting the accuracy need investigating. Some potential avenues are:

  1. *Faster PDE solvers*: Currently, the governing PDE is solved using the finite difference method. Improvements here could expedite the solver's performance. Neural network methods could perhaps be employed for this purpose as well.
  2. *Faster gradient computation*: Currently, the gradients are computed using PyTorch's autograd. Explore ways to calculate or approximate it between the cost and the boundary values faster.

Overall, approaching boundary control problems with deep learning and reinforcement learning has merits. Further research may enable arriving at initial guess and optimizer methods that can achieve lower costs with fewer computations. The idea followed in this project may extend to optimization problems beyond boundary control problems; for instance, optimal power flow problems.

References

Bello, I.; Zoph, B.; Vasudevan, V.; and Le, Q. V. 2017. Neural Optimizer Search with Reinforcement Learning. *CoRR*, abs/1709.07417.

Bonny, T.; Kashkash, M.; and Ahmed, F. 2022. An efficient deep reinforcement machine learning-based control reverse osmosis system for water desalination. *Desalination*, 522: 115443.

Brandstetter, J.; Worrall, D.; and Welling, M. 2022. Message passing neural PDE solvers. *arXiv preprint arXiv:2202.03376*.

Cai, X.; and Diagne, M. 2021. Boundary Control of Nonlinear ODE/Wave PDE Systems With a Spatially Varying Propagation Speed. *IEEE Transactions on Automatic Control*, 66(9): 4401–4408.

Chamberlain, B.; Rowbottom, J.; Eynard, D.; Di Giovanni, F.; Dong, X.; and Bronstein, M. 2021. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34: 1594–1609.

Chen, S.; Dong, J.; Ha, P.; Li, Y.; and Labi, S. 2021. Graph neural network and reinforcement learning for multi-agent cooperative control of connected autonomous vehicles. *Computer-Aided Civil and Infrastructure Engineering*, 36(7): 838–857.

Colli, P.; Gilardi, G.; and Sprekels, J. 2015. A boundary control problem for the pure Cahn–Hilliard equation with dynamic boundary conditions. *Advances in Nonlinear Analysis*, 4(4): 311–325.

Fawzi, A.; Balog, M.; Huang, A.; Hubert, T.; Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Ruiz, F. J. R.; Schrittwieser, J.; Swirszcz, G.; Silver, D.; Hassabis, D.; and Kohli, P. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53.

Gama, F.; and Sojoudi, S. 2021. Graph Neural Networks for Distributed Linear-Quadratic Control.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 1861–1870. PMLR.

Jiang, Z.; Jiang, J.; Yao, Q.; and Yang, G. 2023. A neural network-based PDE solving algorithm with high precision. *Scientific Reports*, 13(1): 4479.

Kardoš, J.; Kourounis, D.; and Schenk, O. 2020. Two-Level Parallel Augmented Schur Complement Interior-Point Algorithms for the Solution of Security Constrained Optimal Power Flow Problems. *IEEE Transactions on Power Systems*, 35(2): 1340–1350.

Kardoš, J.; Kourounis, D.; Schenk, O.; and Zimmerman, R. 2022. BELTISTOS: A robust interior point method for large-scale optimal power flow problems. *Electric Power Systems Research*, 212: 108613.

Lagaris, I. E.; Likas, A.; and Fotiadis, D. I. 1998. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5): 987–1000.

Lang, J.; and Schmitt, B. A. 2023. Exact Discrete Solutions of Boundary Control Problems for the 1D Heat Equation. *Journal of Optimization Theory and Applications*, 1–13.

Liu, H.; Li, Z.; Hall, D.; Liang, P.; and Ma, T. 2023. Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training. arXiv:2305.14342.

Maurer, H.; and Mittelmann, H. D. 2000. Optimization techniques for solving elliptic control problems with control and state constraints: Part 1. Boundary control. *Computational Optimization and Applications*, 16(1): 29–55.

Meirom, E.; Maron, H.; Mannor, S.; and Chechik, G. 2021. Controlling graph dynamics with reinforcement learning and graph neural networks.

Pacaud, F.; Schanen, M.; Shin, S.; Maldonado, D. A.; and Anitescu, M. 2023. Parallel Interior-Point Solver for Block-Structured Nonlinear Programs on SIMD/GPU Architectures. *arXiv preprint arXiv:2301.04869*.

Rozemberczki, B.; Scherer, P.; He, Y.; Panagopoulos, G.; Riedel, A.; Astefanoaei, M.; Kiss, O.; Beres, F.; Lopez, G.; Collignon, N.; and Sarkar, R. 2021. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, 4564–4573.

Shen, Y.; Shi, Y.; Zhang, J.; and Letaief, K. B. 2019. A Graph Neural Network Approach for Scalable Wireless Power Control.

Tasseff, B.; Coffrin, C.; Wächter, A.; and Laird, C. 2019. Exploring Benefits of Linear Solver Parallelism on Modern Nonlinear Optimization Applications. arXiv:1909.08104.

Wächter, A.; and Biegler, L. T. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106: 25–57.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, 5–32.

Zafar, A.; and Manchester, I. R. 2023. Structured linear quadratic control computations over 2D grids. arXiv:2304.04980.

Zhu, X. 2018. An optimal control view of adversarial machine learning. *arXiv preprint arXiv:1811.04422*.

Appendix

Code and data

The implementation code, generated dataset, results and analysis are available in https://github.com/zenineasa/MasterThesis.

To set up the Anaconda environment and IPOPT, follow the instructions in the Jupyter Notebook file named *Code/setup.ipynb*. The environment information is available in the file named *Code/environment.yml*. If the environment installation process does not work as specified in Jupyter Notebook, please manually install the packages listed in Table 5.

The program that generates synthetic data is in the file named *Code/dataGenerator.py*, and the generated data is in the file named *Code/Data/data.csv*. Files related to baseline generation and a few initial experiments are in the folder *Code/experiment*. Files about applying deep learning and reinforcement learning methods to solve boundary control problems are in the folder *Code/FinalAttempt*.

Hardware and software setup

The specification for the device used in training and validating all the neural networks is provided in Table 2. Testing was performed using several regular nodes in an HPC cluster with specifications listed in Table 3. For performance evaluation, an old Linux machine with administrator privileges available to run a performance analyzing tool named 'perf' was used, the specification of which is described in Table 4. The information on the software and libraries used in this project are listed in Table 5.

Component	Specification
Hardware	MacBook Pro 2019, 16-inch
Processor	2.6 GHz Intel Core i7
No. of cores	6
Graphics	AMD Radeon Pro 5300M 4 GB, Intel UHD Graphics 630 1536 MB
Memory	16 GB DDR4 @ 2667 MHz
OS	macOS 13.3.1

Table 2: Specification for the machine used for training, testing and validating neural networks

Component	Specification
Processor	Intel(R) Xeon(R) CPU E5-2650 v3
No. of cores	20
Memory	64GB DDR4 @ 2133MHz
OS	CentOS Linux 8 (Core)

Table 3: Specification for the nodes used from the HPC Cluster

Component	Specification
Hardware	Lenovo ideapad 330-15ARR
Processor	AMD Ryzen 5 2500U
No. of cores	8
Graphics	AMD Radeon vega 8 graphics
Memory	8 GB (2× 4 GB DDR4 @ 2400 MHz)
OS	Ubuntu 22.04.02 LTS

Table 4: Specification of the machine used for performance evaluation

Software / Library	Version
Python	3.9.16
PyTorch	2.0.0
Pandas	1.3.5
Anaconda	4.13.0
IPOPT	3.14.12
MATLAB	R2022b

Table 5: Versions of different software and libraries used

The implementation of the proposed method is written in Python using PyTorch and other libraries, while the analysis is in MATLAB, which makes it easier to know which files in the codebase are implementation and analysis related.

Removing the effect of the sourcing term

The following are two relationships related to the sourcing term of Laplace's equation.

- **Relationship between solutions for different sourcing term values**: Setting the boundary values to zero and numerically solving Poisson's equation for different sourcing term values, one can observe that the solution matrix when the sourcing term is -20 is double that of when it is -10. Similarly, for -30, -40 and -50, the solution matrices are three, four and five times that for -10, respectively.
- **Relationship between solutions for random and zero boundary values**: For a fixed domain size, if A is the numerical solution for a given sourcing term with boundary values set to zero, B is the numerical solution for random boundary values with the sourcing term set to zero, and C is the numerical solution for the same random boundary values and the given sourcing term, then it can be observed that,
$$A + B = C \tag{1}$$

Let's put these relationships together. Forward solving for Poisson's equation with a sourcing term of -10 using zero boundary values of dimensions $4 \times N$ will result in a matrix of size $N \times N$. For reusing, store these matrices for different values of N. To get the corresponding value for other sourcing terms, simply multiplying the values in the stored matrix for the desired size with the ratio of the required sourcing term and -10 would suffice. This matrix can be subtracted from the solution to Poisson's equation with a constant sourcing term to get the solution to Poisson's equation without a sourcing term!

Initial guess method and edge values

It was hard to conclude whether using the initial guess method or the edge values for the initial guess was more beneficial. Let us analyze the 3606 cases where using the edge values for initial guess were observed to have lower cost and see if there is a pattern.

Partial correlation between the cost differences between the initial guess method and the edge values, and the problem parameters were calculated. A relatively high absolute partial correlation between the sourcing term and the cost differences was observed, which is a value of 0.4593. Although this correlation in itself is not significantly high, because the correlation with every other parameter is below 0.07, it may be useful to have a look into this.

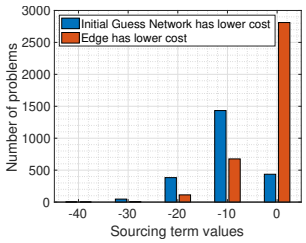


Figure 16: Number of problems for which initial guess method or edge values had lower cost for different sourcing term values

The number of cases where one method has a lower cost than the other can be compared easily. Figure 16 shows a bar graph helping visualize the same. Observe that using the edge values might be more desirable when the sourcing value is 0. In every other case, using the initial guess method would be more desirable.

This may indicate that one could create a better initial guess mechanism by adding a control statement that would let one guess the initial values as the edge values if the sourcing term value is zero and use the initial guess method in every other case. However, it is also likely that there exists some bias in the problem generator that has caused this.

Adam and RMSProp in optimizer method

There are two reasons why Adam and RMSProp were included along with the spatio-temporal network in the optimizer method.

1. Adam and RMSProp have been widely used in deep learning as optimization algorithms to tune network weights. They have demonstrated to work well in quite a lot of scenarios.
2. Adam and RMSProp appear to be guiding the spatio-temporal network to train. Without them, the best values observed would be the first set of values (the output of the initial guess method), and the network iterations would not be considered during backpropagation.

Indeed, the second issue could have been addressed by tweaking the training method. It is also true that either one of Adam or RMSProp could have been used for this and not both simultaneously. Furthermore, several other optimizers are being used in research like Adaptive Gradient Algorithm (Adagrad), AdaDelta, Stochastic Gradient Descend (SGD), etc. This is just the decision that was made.

Contribution of Adam, RMSProp and spatio-temporal parts

The learnt learning rates for Adam, RMSProp and the Spatio-temporal part of the optimizer are respectively 0.0223, 0.0221 and 0.0645. However, this does not conclusively mean that the contribution of Spatio-temporal is higher than that by Adam and RMSProp.

The information regarding contribution of each of the three parts are recorded for 32 optimizer iterations along with the domain size and order of iteration. The average magnitude of contribution per optimizer iteration for Adam, RMSProp and the Spatio-temporal part are, respectively, 0.0151, 0.0133 and 0.0251. Clearly, the contribution from the network is higher in magnitude.

Furthermore, the partial correlation between Adam, RMSProp and Spatio-temporal parts with the number of iterations are -0.6017, -0.3360 and -0.0086, respectively. The effects of this correlation are visualized in Figure 17. For Adam and RMSProp, the values are as expected; as the number of iterations increases, the contribution would decrease, indicating that they are converging to a solution. However,

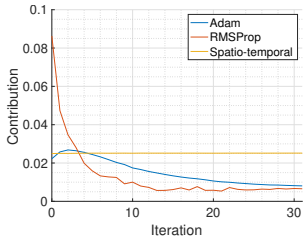
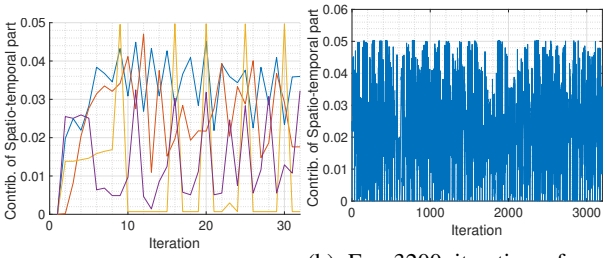


Figure 17: Comparing the contribution by Adam, RMSProp and Spatio-temporal part over different iteration

for some peculiar reason, the magnitude of contribution from the Spatio-temporal part does not seem to change, suggesting that this part may not be doing anything meaningful, as it may just be outputting some constant value. Let us investigate whether that is true or not.

Investigating Spatio-temporal part This investigation is to know if the network is doing something meaningful and not just outputting the biasses. If it is all about the bias terms, then for different inputs to the Spatio-temporal part, the output would more or less be a constant value.

In the setup, a minor modification is made to store the value of the output of the Spatio-temporal part in the matrix form for the first iteration. Then, for every subsequent iteration, the output of the Spatio-temporal part at that iteration is subtracted with the stored value, following which the mean of the absolute values are calculated and logged. Figure 18 helps visualize logged information. Observe that the contribution from the network is not constant. The variations observed are between 0 and 0.0503, which is significant.



(a) For four different problems

(b) For 3200 iterations from different problems

Figure 18: Contribution by Spatio-temporal part visualized

Although it is still quite interesting to see that the contribution by the Spatio-temporal network has a constant-ish mean absolute value, it does not mean in any way that the network itself is giving us a constant-ish value.