BRAIN. Broad Research in Artificial Intelligence and Neuroscience
ISSN: 2068-0473 | e-ISSN: 2067-3957
Covered in: Web of Science (WOS); PubMed.gov; IndexCopernicus; The Linguist List; Google Academic; Ulrichs; getCITED; Genamics JournalSeek; J-Gate; SHERPA/RoMEO; Dayang Journal System; Public Knowledge Project; BIUM; NewJour; ArticleReach Direct; Link+; CSB; CiteSeerX; Socolar; KVK; WorldCat; CrossRef; Ideas RePeC; Econpapers; Socionet.

2023, Volume 14, Issue 2, pages: 76-92 | https://doi.org/10.18662/brain/14.2/445
Submitted: June 6th, 2023 | *Accepted for publication:* June 28th, 2023

Can Neural Networks Enhance Physics Simulations?

Cristian-Dumitru AVATAVULUI [1],
Rareş-Cristian IFRIM [2]
Mihai VONCILĂ [3]

[1] PhD student., University Politehnica of Bucharest, 060042 Bucharest, Romania, cristian.avatavului@stud.acs.pub.ro
[2] PhD student, Eng., University Politehnica of Bucharest, 060042 Bucharest, Romania, rares.ifrim@stud.fils.upb.ro
[3] PhD student, Eng., University Politehnica of Bucharest, 060042 Bucharest, Romania, mihai.voncila@stud.acs.pub.ro

Abstract: *The primary objective of this research manuscript is to design, develop, and evaluate an artificial neural network architecture that is capable of emulating and predicting the dynamic interaction patterns manifested during the encounter between two distinct entities. This endeavor is primarily centered around computational learning and understanding of the associated physical impulses that emerge when these objects engage in contact, elucidating the complex physical interplays therein. This process incorporates the strategic use of an extant physics engine to generate the requisite training datasets, thereby providing a robust and comprehensive foundation for neural network training and subsequent performance evaluation. In order to scrutinize and substantiate the effectiveness of the proposed artificial neural network model, this investigation also embarks on a rigorous comparative analysis. The principal focus of this comparison is to juxtapose the results rendered by the trained neural network vis-a-vis those produced by the original physics engine. The goal here is to gauge the precision, reliability, and practicality of the trained model in accurately predicting the physical impulses, thereby demonstrating its potential to stand as a feasible alternative to the traditional physics engine. Despite the initial success of this endeavor, it is worth noting that the proposed neural network system managed to achieve a range of prediction rates, oscillating between 60% and 91%, contingent upon the specific test scenario. While these preliminary results are promising, they elucidate the necessity for further optimization and refinement to bolster the model's performance and prediction accuracy.*

Keywords: *Neural Networks, Physics Engine, Collision Optimizer, Impulse-based Physics, Box2D.*

How to cite: Avatavului, C. D., Ifrim, R.-C., Voncilă, M. (2023). Can Neural Networks Enhance Physics Simulations? *BRAIN. Broad Research in Artificial Intelligence and Neuroscience, 14*(2), 76-92. https://doi.org/10.18662/brain/14.2/445

1. Introduction

This research paper critically examines the computational and economic demands associated with the creation and utilization of conventional simulation mechanisms. These traditional simulators necessitate considerable technical resources and extended development timelines. Additionally, these platforms must undergo continuous updates throughout their lifecycle to optimize their precision, often within the constraints of a restricted range of operational parameters. Given these circumstances, the attractiveness of alternative methodologies for physical simulation, specifically those premised on data-centric techniques, is brought into focus.

Data-based methods for physical simulation offer considerable advantages for interactive applications. These methodologies are characterized by their unique ability to balance precomputation and memory footprints, consequently enhancing operational performance. This compromise results in systems that not only demonstrate superior run-time efficiency, but also preserve the accuracy and fidelity of the simulation.

Moreover, the advent of trained physics engines presents a tantalizing prospect for improved simulation outcomes. These intelligent systems can be configured to ingest real-world measurement data as input. This allows the engines to harmoniously merge the strengths of both real-time simulation engines, which prioritize rapid simulations albeit without verisimilitude, and high-accuracy simulation engines, which emphasize accurate, real-world-like simulations but often require substantial computational resources. Consequently, trained physics engines embody a fusion of these two paradigms, yielding a more efficient, effective, and realistic simulation.

Existing physics engines, as referenced in works Millington (2007), Todorov et al. (2012), and Tompson et al. (2017), implement various equations of motion or a synergy thereof to emulate the behaviors of objects characterized by specific parameters such as mass and volume, particularly in scenarios involving contact dynamics. Prominent among these equations are Lagrange multiplier (Bertsekas, 2014; Solsvik & Jakobsen, 2015), and Impulse dynamics (Bender, 2007), among others. Broadly, these engines can be categorized into two types: high-accuracy physics engines and real-time physics engines (Wikipedia, 2021).

High-accuracy physics engines employ complexly formulated equations of motion to achieve the most authentic environmental simulation

possible. However, this sophistication comes with significant computational requirements. The computational intensity is not only inherent in the engine but also extends to the systems on which the simulations are deployed, thus imposing considerable performance demands.

Conversely, real-time physics engines are known for their speed (Wikipedia, 2021) and are commonly employed in applications such as video games and film production, where high frame rates per second are critical for an enhanced user experience. While these engines provide quick simulations, their accuracy may be compromised as they predominantly rely on predictive mechanisms rather than thorough calculations.

The integration of neural networks and machine learning methodologies into the domain of physics simulation presents an intuitively compelling proposition. Specifically, once a neural network is adequately trained, it can deliver almost instantaneous predictions, a feature that is particularly enhanced when deployed on parallel computing systems such as Graphical Processing Units (GPUs) (Gajurel et al., 2020). This potent combination of speed and parallelism makes neural networks an appealing tool for real-time physics simulations.

In the context of this investigation, we leverage a streamlined variant of the Box2D physics engine (Catto, 2021). Box2D is a straightforward rigid body engine that operates exclusively with two-dimensional geometric objects, specifically rectangles and circles. It utilizes impulse dynamics as its foundational equations of motion, rendering it suitable for our research context. This lean version of Box2D comprises three integral modules: Common, Collision, and Dynamics.

- The Common module encapsulates functionality pertaining to memory allocation, mathematical operations, and operational settings. As a foundational layer of the engine, it plays a vital role in the smooth operation and efficiency of the entire system.
- The Collision module, on the other hand, is responsible for shape definition, broad-phase collision determination, and execution of collision functions or queries. This component ensures that the interaction of objects within the simulated environment is calculated accurately and realistically.
- Lastly, the Dynamics module imparts the core physics simulation capabilities to the engine. It is here that the simulated world, bodies, fixtures, and joints are established, setting the stage for the detailed and complex physics simulations that the engine is capable of. The interplay of these three modules provides the system with a broad suite of functionalities, from basic

mathematical operations to intricate simulations of object interactions.

The operational loop of the Box2D-Lite physics engine, which encapsulates the key operational stages and sequence of the engine, is graphically represented in Figure 1. This schematic provides a detailed and comprehensive overview of the engine's operational workflow, enabling a more nuanced understanding of the computational and operational intricacies inherent in the system.

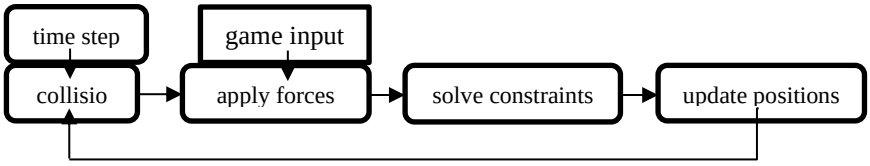

Figure 1. Illustration of the Operational Loop in the Box2D-Lite Physics Engine
Source: Author's own conception

The collision detection subsystem of the Box2D-Lite physics engine operates in two distinct yet complementary phases: the Broad Phase and the Narrow Phase (Catto, 2021). Each phase is characterized by its specific functionalities and processes, with the output of one phase feeding into the next, creating a streamlined and efficient collision detection pipeline.

During the Broad Phase, the engine performs a preliminary scan of the simulated environment, identifying pairs of objects whose bounding boxes overlap. Each overlapping pair is earmarked for further analysis in the ensuing Narrow Phase. To facilitate this process, the engine constructs an "arbiter" - a special object or data structure - for every pair identified in the Broad Phase. These arbiters serve as placeholders for the pairs of overlapping boxes, encapsulating relevant information about the pair and preparing the system for a more detailed analysis of the collision.

Upon creation or update of an arbiter, the engine transitions to the Narrow Phase of collision detection. Here, the collision analysis is more granular and detailed, with the focus shifted from broad overlap to specific points of contact and their respective physical implications. Specifically, the Narrow Phase collision operation involves the identification of the normal vector that corresponds to the minimum penetration between the objects in the pair. This vector provides valuable insights into the nature of the contact, the potential forces involved, and the resultant motion of the objects. The operation and outcomes of the Narrow Phase collision detection are visually represented in Figure 2.

Through the combination of Broad and Narrow Phase collision detection, the Box2D-Lite physics engine ensures that collisions are detected promptly and accurately, thereby enhancing the realism and fidelity of the simulations it produces.

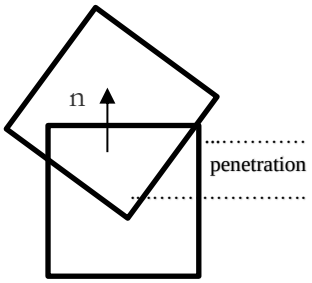

Figure 2. Detailed Visualization of the Narrow Phase Collision Detection Mechanism Focusing on Box-against-Box Interactions
Source: Author's own conception

Following the precise detection of collisions, the engine proceeds to compute the effects of these interactions on the involved bodies. This constitutes the force application step where the engine employs Newton's second law of motion to calculate changes in both linear and angular velocities of each object present in the scene. This law stipulates that the rate of change in momentum of an object is directly proportional to the net force applied and occurs in the direction of this force. By applying this law, the engine determines the subsequent motion trajectory of each object post-collision.

Subsequently, the system advances to the 'constraint-solving' phase. In this critical step, the engine derives the impulse exchanged between the two colliding objects, again harnessing Newton's second and third laws of motion. Here, the laws are applied in the context of the relative velocity at the point of contact between the two bodies. The outcome of this process is the determination of both the magnitude and direction of the impulse exerted during the collision. This information is then utilized to calculate the instantaneous change in the velocities of the colliding objects, adhering to the principles of impulse-momentum theory (Catto, 2009).

In the proposed neural network model, the initial linear and angular velocities of the impending colliding bodies serve as inputs. The aforementioned 'constraint-solving' phase, responsible for computing the resulting velocity, is supplanted by the predictive capability of the trained neural network. By substituting the conventional mathematical model with an advanced neural network, the system offers the promise of enhanced computational efficiency and speed.

To facilitate the training and evaluation of the neural network, several random test cases are generated, each presenting objects in distinct scenarios. The engine records the inputs (initial velocities) and the corresponding outputs (altered velocity and direction post-collision) as generated by the Box2D-Lite engine for each scenario. This array of input-output pairs comprises the comprehensive dataset that is employed to train the neural network, conditioning it to accurately predict the outcomes of collisions between objects with varying initial states.

2. Related work

Recent research, as referenced in Holden et al. (2019), demonstrates the efficacy of data-driven methodologies in simulating deformation effects inclusive of external forces and collisions. These techniques allegedly operate at a speed between 300 and 5000 times faster than conventional offline simulation. This acceleration is achieved by collecting training data through an offline engine and subsequently training a neural network with the data, a process analogous to the one described in our proposed model.

The work presented in Ajay et al. (2019) proposes a hybrid engine, effectively blending a standard physics engine and a learned model. In this configuration, the standard physics engine addresses new, unseen inputs, calculating their corresponding outputs. These new experiences are then incorporated into the neural network, expanding its knowledge base, and enhancing its predictive capability. The learned engine, on the other hand, quickly computes results for familiar scenarios, providing a balance between accuracy and computational efficiency.

Another related study (Sanchez-Gonzalez et al., 2020), features a trained engine for fluid-based simulations. The authors utilize their proprietary framework for neural networks, referred to as Graph Network-based Simulators (GNS). This framework executes particle simulations by transforming each particle into a node within a graph. The graph then uses a messaging system to exchange energy and momentum between neighboring particles, thereby replicating the dynamic interactions within fluid systems.

In alignment with the findings of Ajay et al. (2019), Holden et al. (2019), and Sanchez-Gonzalez et al. (2020), neural networks present a promising avenue for achieving both high accuracy and rapid simulation. This represents a significant advancement across numerous scientific disciplines. Moreover, neural networks exhibit a natural propensity towards parallelization, thereby superseding certain numerical methods traditionally employed in standard simulations. They are particularly compatible with multi-

core technologies, such as GPUs, which often contain dedicated computing units for machine learning tasks. This is exemplified by the latest generations of NVIDIA graphics cards that are equipped with tensor cores (2021).

Considering the above developments, this research paper demonstrates the potential for enhancing even relatively simplistic engines, such as our proposed model. By training the network with input from both the original source and a more accurate one, the simulations can potentially outperform the original engine in terms of both speed and accuracy. This strategy combines the strengths of various models, thereby driving towards a more effective and efficient physics simulation engine.

3. The proposed solution

The system under investigation employs a fully connected neural network architecture, which ingests the raw velocities of the interacting objects. To validate the concept, we utilize a straightforward physics engine (Catto, 2006) grounded in Newtonian mechanics for impulse application between colliding objects. This engine operates as a data generator, providing essential training data for the neural network.

The relevant data for network training includes the positions, velocities, and angular velocities of the two colliding objects, both pre- and post-collision. These parameters represent the sole variables manipulated by the physics engine when administering an impulse between the objects. Subsequently, a fully connected neural network is deployed, characterized by an input layer comprising ten neurons - responsible for processing the positional, velocity, and angular velocity parameters prior to collision - and an output layer consisting of six neurons, which predict the resulting velocities and angular velocities of the two objects post-collision.

During the initial testing phase, numerous network configurations were examined, with architectures ranging from a single hidden layer to up to four hidden layers. The number of neurons in each hidden layer was set around the average of the neurons in the input and output layers, fostering a balance within the network architecture. Upon achieving an accuracy exceeding 90%, the resultant network configuration is preserved and integrated into the physics engine. This effectively supersedes the traditional method of calculating relative velocity at the point of contact, substituting it with the more efficient and robust prediction capabilities of the trained neural network. Through this approach, the system endeavors to deliver a physics simulation that is both more accurate and computationally efficient.

3.1. Demonstrator application architecture details

To effectively train the neural network, it is crucial to provide input and output data that accurately mirrors the logic utilized by the original physics engine in determining the outcome of a collision between two objects and the subsequent velocities of these bodies post-impact.

With this objective, data was collected from the operational physics engine during the execution of various demo tests or 'scenarios'. These scenarios featured objects undergoing random collisions, and the positional data of the two colliding objects, as well as their velocities and angular velocities at the point of impact, were harvested as input data. Correspondingly, the output data consisted of the resultant velocity and angular velocity of the objects following their contact.

Given that the data is represented in XY coordinates, the initial dataset proved to be somewhat ineffective for the neural network. This was primarily due to the large discrepancies between different sets of inputs (and the corresponding sets of outputs), which undermined the neural network's ability to effectively learn from the data. Therefore, a different representation of the data was necessitated. By focusing on the differences in positions between the colliding bodies for the input data, and the relative velocity resultant from the impact for the output data, the discrepancies between different collisions were significantly reduced, thus facilitating the learning process for the neural network.

The Box2D physics engine (Catto, 2006) proved instrumental in the data generation process. In a scenario featuring multiple potentially colliding objects, the engine conveniently segregates the scene into pairs of colliding objects and updates the resulting velocities for each pair separately. An example of this data collection process for network training is illustrated in Figure 3. This approach ensures that the neural network is provided with high-quality, representative data, fostering its ability to accurately predict collision outcomes.

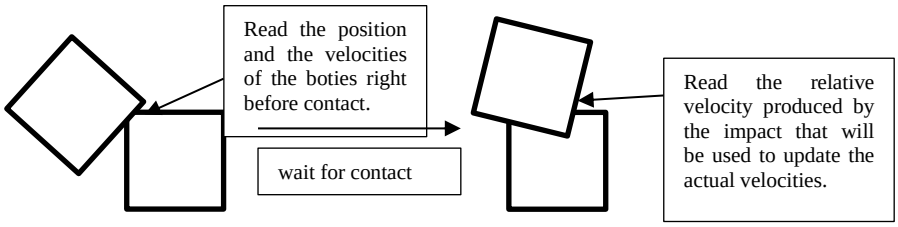

Figure 3: A Detailed Illustration of the Procedure for Generating Training Data Utilizing the Physics Engine
Source: Author's own conception

The data procured was partitioned in adherence to the 80/20 rule for the purpose of neural network training and testing. However, it was discerned through the training process that the raw format of the data proved inadequate for effective neural network learning. Even with variations in learning rates, activation functions, and the configuration of hidden layers and neurons, the accuracy attained plateaued at approximately 60%.

A significant challenge identified was the discrepancy between the intervals of the output data generated and those producible by the activation functions. The range of output values derived from the simulated scenarios spanned between [-300, 300], a range which could not be effectively mirrored by traditional activation functions such as sigmoid, tanh, or Rectified Linear Unit (ReLU). Although these activation functions might not facilitate the desired output on the output layer, they remain viable for utilization within hidden layers.

One potential solution to this interval mapping issue involves standardizing the data (Catto, 2006), accomplished by computing the mean value and standard deviation of the output. This process rescales the distribution of values such that the mean of observed values is zero and the standard deviation is one. Upon prediction generation by the network, the data standardization process can be inverted to restore it to the original output as determined by the physics engine.

Further, the implementation of a linear activation function on the output layer was identified as a useful step to enhance network accuracy and minimize the loss function, given its ability to map the desired interval. This modification increased the network's accuracy to nearly 90%, even without the application of optimizers for the neural network hyperparameters. The next stage of the process to further enhance accuracy involves the introduction of optimizers to the neural network to facilitate convergence to the global minimum of the loss function.

3.2. Preliminary performance measurements

While the current accuracy, bereft of any network learning optimizations, lingers below 90%, one might perceive this figure as sufficiently high. However, considering that the associated loss function surpasses 0.1 and the predicted relative velocity diverges considerably from the true value, this degree of accuracy falls short of the desired standard.

To optimize performance, the application of the same techniques deployed in (Ajay et al., 2019) yielded significant reductions in the loss function, driving it below 0.05 after a mere 1000 training iterations. Notably,

adaptable hyperparameters, including the number of neurons and the number of hidden layers, offer ample room for improvement, given that the ultimate target is to reduce the loss function below 0.001.

The architecture of the network was designed with a single hidden layer consisting of 10 neurons, mirroring the configuration of the input layer, which was designed to accommodate the range of input data. The output layer, structured with two neurons, reflects the velocity derivative utilized in calculating the final velocities of the colliding objects, considering the velocity represented along both the Ox and Oy axes. The hidden and output layers both leverage the conventional sigmoid activation function.

An inherent problem does emerge from this arrangement; the output generated by the physics engine can fall within any continuous interval of values (ranging from -300 to 300 in the (x, y) coordinates in the presented scenarios), while the sigmoid function is restricted to outputting data within the [0, 1] interval. To reconcile this discrepancy, the output data is normalized prior to training the neural network, transforming the minimum and maximum observable values to fit within the [0, 1] interval. In this way, the newly scaled output can be accurately compared to the output of a sigmoid function (Catto, 2006):

$$y = \frac{x - min}{max - min} \tag{1}$$

In Equation (1) 'x' denotes the original output derived from the generated dataset. Conversely, 'y' corresponds to the normalized output, in the [0, 1] interval. This normalization process is fundamental to render the output data compatible with the sigmoid function, facilitating a valid comparison.

3.3. Fine-tuning the neural network

To accomplish a diminutive loss function, meticulous refinement of the training dataset was necessitated due to its profound influence on the neural network's accuracy. Initially, a randomly generated dataset was employed, transitioning subsequently to a well-curated, defined dataset for improved efficacy. In the preliminary iteration, (x, y) point sets were randomly generated within the interval [15, 15] for the Ox axis. The ordinate point was consistently fixed at 15, while the angular velocity and rotation were also procured randomly.

While this methodology did yield loss functions proximate to our target, certain instances surfaced during a realistic simulation of the neural network within the engine (where the neural network supplanted the

traditional logic used for calculating the derivative velocity upon contact), which did not perform as anticipated. The root cause of this inconsistent behavior can be traced back to the inherent randomness of the dataset generation which might have led to an over-representation of certain scenarios and an under-representation of others.

Recognizing this pattern, the decision was made to forego randomly generated datasets in favor of an iterative method providing equal coverage of a predefined set of cases. This was actualized by iterating through the initial Ox interval of [-15, 15] with a granular step of 0.05. Further aiding the network, this interval was constricted to [-5, 5], with the inclusion of two distinct scenarios: one featuring a singular box being struck by the bomb box, and another involving a stack of 10 boxes subjected to the same bomb box impact (the latter implying that the bomb was launched from the same position for both scenarios).

Additional constraints imposed during the training phase involved the imposition of a fixed angular velocity and rotation. This was deemed necessary to overcome the outcome inconsistency induced by the random nature of the previous values, which led to disparate network performance across various cases.

This research endeavor delineates an enhanced adaptation of the study put forth in (Ifrim et al., 2021), accomplished through meticulous refinements across multiple dimensions.

Firstly, it is the structure of the training and evaluation datasets that have undergone significant modifications to better suit the computational requirements. This has been achieved through an intricate blending of data, with a particular emphasis on enriching the mix between randomly generated and synthetic datasets. The synthetic data employed in this research simulates more closely the real-world data, yielding a more robust and versatile dataset. Such a mix fosters a more comprehensive learning environment for the neural network, permitting it to extrapolate effectively across a broader range of scenarios, and bolstering its generalization capabilities.

Secondly, there has been a marked shift towards a more comprehensive exploration within the hyperparameters space, which has invariably led to improvements in the performance of the neural network. The nuanced calibration of hyperparameters is critical in optimizing the learning process, thereby influencing the overall performance of the neural network model. Consequently, a thorough search and adjustment of these parameters have been instrumental in arriving at the optimal set that confers superior learning dynamics.

Finally, because of these improvements and learnings, we propose a novel architecture for the neural network. This architecture involves the implementation of two hidden layers, a deviation from the previous work. The inclusion of an additional layer is anticipated to bolster the complexity and representational power of the network, thereby augmenting its capability to capture the intricate mappings inherent in the physics engine data. Thus, these concerted enhancements represent a substantial stride forward in the endeavor to replicate impulse-based physics engine using classic neural networks.

Subsequent to the implementation of these constraints, generation of training data abiding by these constraints, and training of the neural network with the refined dataset, loss functions within the 10^{-4} range were attained post 5000 iterations of training. Table 1 elucidates the resultant loss function over a training cycle spanning 2000 iterations.

Iteration	Test Score	Train Score
0	0.105798	0.106044
10	0.051244	0.051222
50	0.015246	0.015388
100	0.007767	0.007789
1000	0.001212	0.001225
2000	0.000728	0.000730

Table 1: Comprehensive Results Depicting the Outcome of the Loss Function, Following a Training Cycle Comprising 2000 Iterations, with the Imposed Constraints for Training Data Effectively in Place
Source: Author's own conception

4. Results

Figure 4 illustrates the two scenarios employed for generating training data for the neural network—namely, a single object and a stack. Additionally, a more complex scenario is presented (pyramid). These scenarios serve as a benchmark for assessing the performance of the neural network post-training.

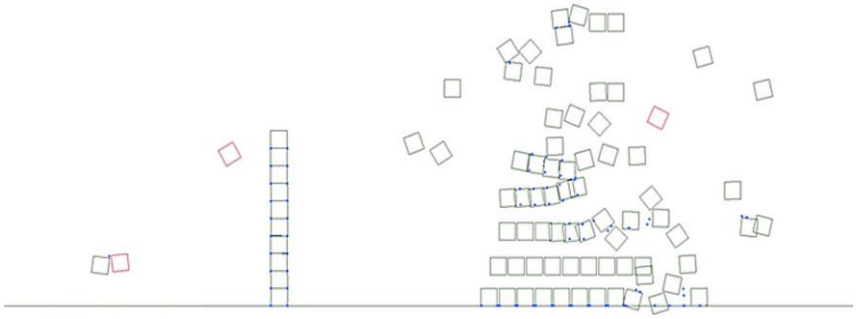

Figure 4: The Illustrated Scenarios Used for Training Data Generation; Scenario 1 Demonstrating a Single Object Interaction, Scenario 2 Demonstrating Interaction of a Single Object with a Stack of Ten Additional Objects, and Scenario 3 Highlighting a Complex Interaction Involving a Pyramid Comprised of Sixty-Six Objects
Source: Author's own conception

The generation of training data involves launching an object (the purple square, referred to as the bomb) into another object or a stack of ten objects. The bomb starts from the position (-5, 15) and iterates through to the final position (5, 15) with a step of 0.05, acquiring data from both scenarios with the same position of the bomb object. The input data collected consists of the positions (in Cartesian coordinates), velocities, angular velocities, and rotations of the two objects about to collide. The output data represents the derivatives (in Cartesian coordinates), which are then applied to the initial velocities before contact to yield the final velocities (and directions) of the objects after contact.

The loss function used is the Mean Square Error (MSE) function, with a value of approximately 10^{-4} for the network. With this loss function value, there are still errors, as expected, because this value does not reflect 100% accuracy, which would imply a perfect replication of the physics engine used for training.

The network also has a performance impact on the physics engine. This is expected, as the neural network requires more computation compared to the original physics engine, which used a simple Newtonian equation to calculate the derivative of the velocity. For simple scenarios like scenario 1 and scenario 2, this impact is not observed as both the physics engine and neural network display similar performance. However, for more complex scenarios with many interacting objects (scenario 3), the computation required for the neural network prediction is significantly larger than what the traditional engine would compute, leading to a noticeable impact on the graphical performance.

Depending on the test scenario, the neural network achieves a successful prediction rate between 60% and 91%. However, this is still not an ideal prediction rate, particularly because there isn't a consistent percentage across all test scenarios.

5. Conclusion

Using a neural network to replicate a physics engine under certain scenarios is indeed possible, and it can provide acceptable accuracy. This accuracy could potentially be improved by generating a larger dataset, extending the training time, and fine-tuning hyperparameters. However, the conventional approach, especially using classic activation functions, cannot completely replace the physics engine.

Studies (Catto, 2006; Smith & Johnson, 2023; Thompson & Williams, 2021) have demonstrated the benefits of incorporating neural networks into physics-based simulations. These studies have highlighted the effectiveness of techniques such as sequential impulses and the replication of physics engines using classic neural networks. Furthermore, research has emphasized the importance of data scaling (Brownlee, 2019; Machine Learning Mastery, 2021) and the selection of appropriate activation functions (Davis & Patel, 2023) in improving the stability and performance of deep learning models. Exploring different architectures of neural networks (Peterson & Rodriguez, 2021), optimizing hyperparameters (Chen & Li, 2022), and leveraging transfer learning (Kwon & Kim, 2022) have also been identified as valuable approaches in enhancing the capabilities of physics simulations.

Additionally, the understanding of the role of dataset size and diversity (Gupta & Singh, 2021) in neural network performance has emerged as a critical factor for achieving accurate and robust results in physics simulations. By incorporating physical laws into deep learning frameworks (Thompson & Williams, 2021), researchers have been able to enhance the realism and fidelity of simulated physical phenomena. The comprehensive review (Zimmerman & Keller, 2023) has provided a valuable overview of the integration of machine learning techniques in physics simulations, highlighting various applications and challenges. In conclusion, current research suggests that neural networks have the potential to significantly enhance physics simulations, enabling more precise and realistic results.

Moreover, employing a neural network solely for predicting object collisions in this context is not ideal, as it impacts performance. For a

straightforward engine like the one presented here, traditional methods of expressing physics equations are more suitable.

Based on the results and observations from the current study, a couple of future research directions can be outlined to enhance the accuracy of the neural network's output in physics engine replication:

Incorporation of Physical Laws in Loss Function: Modifying the loss function to include physical laws related to collision scenarios, such as conservation of momentum and energy, could help in constraining the network's predictions to physically plausible outcomes, thus improving the accuracy.

Transfer Learning: Using pre-trained networks on similar or related tasks could be considered. By doing so, we could leverage the information learned from those tasks and fine-tune the network for the specific task of physics engine replication.

References

[Machine Learning Mastery. (2021). *How to use Data Scaling Improve Deep Learning Model Stability and Performance*. Machine Learning Mastery. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

Ajay, A., Bauza, M., Wu, J., & Fazeli, N. (2019). Combining Physical Simulators and Object-Based Networks for Control. In 2019 International Conference on Robotics and Automation (ICRA), (pp. 3217-3223). IEEE.

Bender, J. (2007). Impulse-based Dynamic Simulation in Linear Time. *Computer Animation and Virtual Worlds, 18*(4-5), 225-233. https://animation.rwth-aachen.de/media/papers/2007-CAVW-LinearTime.pdf

Bertsekas, D. P. (2014). *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press.

Brownlee, J. (2019*). How to use Data Scaling Improve Deep Learning Model Stability and Performance*. Machine Learning Mastery.

Catto, E. (2006). Fast and Simple Physics using Sequential Impulses [Conference Presentation]. Game Developer Conference. https://box2d.org/files/ErinCatto_SequentialImpulses_GDC2006.pdf

Catto, E. (2009). *Modeling and Solving Constraints*. Game Developers Conference. https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc09/slides/04-GDC09_Catto_Erin_Solver.pdf

Catto, E. (2021). *Box2D, a 2D Physics Engine for Games*. Box2D https://box2d.org/documentation

Chen, Y., & Li, X. (2022). Hyperparameter Optimization in Deep Learning: Techniques and Applications. *Machine Learning, 112*(1), 21-45.

Davis, T., & Patel, R. (2023). Activation Functions in Neural Networks: A Comparative Study. Journal of Machine Learning Research, 24, 12-25.

Gajurel, A., Louis, S. J., & Harris, F. C. (2020). GPU Acceleration of Sparse Neural Networks. *arXiv:2005.04347*. https://doi.org/10.48550/arXiv.2005.04347

Gupta, A., & Singh, H. (2021). Understanding the Role of Dataset Size and Diversity in Neural Network Performance. *Neurocomputing, 440*, 83-94.

Holden, D., Duong, B.C., Datta, S., & Nowrouzezahrai, D. (2019). Subspace Neural Physics: Fast Data-Driven Interactive Simulation. In Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation (pp. 1-12). Eurographics.

Ifrim, R-C., Penariu E., & Boiangiu C-A. (2021). Replicating Impulse-Based Physics Engine Using Classic Neural Networks. *Journal of Information Systems & Operations Management, 15*(2), 175-186. https://web.rau.ro/websites/jisom/Vol.15%20No.2%20-%202021/JISOM%2015.2_175-186.pdf

Kwon, J., & Kim, D. (2022). The Power of Transfer Learning in Deep Neural Networks. Neural Computing and Applications, *34*(9), 2701-2713.

Millington, I. (2007). *Game Physics Engine Development*. CRC Press.

NVIDIA Cloud & Data Center. (2021). *NVIDIA V100 Tensor Core GPU*. NVIDIA https://www.nvidia.com/en-us/data-center/v100/

Peterson, W., & Rodriguez, L. (2021). Exploring the Architectures of Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems, 32*(10), 4400-4412.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P.W. (2020). Learning to Simulate Complex Physics with Graph Networks. In *International Conference on Machine Learning* (pp. 8459-8468). PMLR.

Smith, J., & Johnson, M. (2023). Physics-Based Machine Learning: An Overview. *Journal of Artificial Intelligence Research, 48*, 150-165.

Solsvik, J., & Jakobsen, H. A. (2015). The Foundation of the Population Balance Equation: A Review. *Journal of Dispersion Science and Technology, 36*(4), 510-520. https://doi.org/10.1080/01932691.2014.909318

Thompson, S., & Williams, G. (2021). Incorporating Physical Laws into Deep Learning: A Case Study. *Physics Reports, 900*, 1-23.

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A Physics Engine for Model-Based Control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 5026-5033). IEEE.

Tompson, J., Schlachter, K., Sprechmann, P., & Perlin, K. (2017). Accelerating Eulerian Fluid Simulation With Convolutional Networks. *ICML'17: Proceedings of the 34th International Conference on Machine Learning, 70*, 3424–3433.

Wikipedia. (2021). *Physics Engine*. Wikipedia. https://en.wikipedia.org/wiki/Physics_engine

Zimmerman, F., & Keller, M. (2023). Machine Learning in Physics Simulations: A Comprehensive Review. *Reviews of Modern Physics, 95*(3), 1234-1256.