Utilizing Integral Neural Networks for Imitation Learning

Y. Kazakov[1] **and Y. Cui**[2]

[2]Department of Computer Science, UCLA

This paper describes the implementation and evaluation of two different policy representations in the field of imitation learning using the OpenAI Gym environments: Acrobot, Lunar Lander, Half Cheetah and Ant. The first representation is based on a standard deep neural network (NN). The second approach uses an Integral Neural Network (INN). INN is characterized by continuous functions and integration operations, which allow to reduce the size of the network while maintaining reliable policy accuracy in selecting agent actions. For the first time, we tested the INN representation and checked its online resizing transformation.

integral neural network | pruning | compression | imitation learning | behavioural cloning | offline reinforcement learning

Correspondence: ***iaroslav.kazakov98@gmail.com, yuchencui@cs.ucla.edu***

Introduction

Latency, performance, memory size and power consumption of ML setup are crucial aspects of it deploying into real world application where it needs to process large dataset in real-time (i.e. self-driving car with a deep model based on one kind of Reinforcement Learning). This also hinders the application of AI in low resource devices (i.e. small edge devices, robots etc.). As a step towards this direction we conduct experiments on pruning techniques. Traditional AI approaches struggle with the complexities of real-world scenarios that require larger and larger sizes of neural networks and massive datasets and computational power. This is where more cognitive computing, inspired by the biological architecture of the brain, emerges as a promising solution (1), (2). The challenge is to determine methods for assessing and enhancing the capacity of a neural network, as well as to effectively manage its size while maintaining control over its accuracy. In this article we for the first time tested Integral Neural Networks (INN) (3) in the role of agent's policy representation in Imitation Learning, exploring how this innovative approach can unlock new possibilities in the field. Performance of this method we demonstrated on the descrete control tasks Acrobot, Lunar Lander and on continuous control tasks Half Cheetah and Ant.

Related works

Online and offline Deep Reinforcement Learning agents encounter serious difficulties in effectively using network parameters. Although deep neural networks have a great potential for representing complex policies, the main challenge is to achieve efficient learning. (4). A crucial aspect of effective learning is the minimization of neural network size, which directly affects latency, performance, and energy consumption. The existing research provides a wide range of methods for pruning and resizing deep neural networks, both with and without retraining. This subject is also examined with a particular emphasis on online and offline reinforcement learning (5). The most widespread approach involves pruning the network after training based on a predefined criterion that evaluates the importance of network parameters to the objective function. Many classical studies on pruning utilized the second derivative information of the loss function (6). One of the most straightforward methods is called magnitude pruning. After training, it involves removing a subset of parameters that fall below a certain threshold and then retraining the remaining ones. Another approach includes regularization-based techniques, which introduce sparsity in the network during the optimization process. Therefore, sparse learning methods, policy pruning, and structural evolution provide a promising way to enhance performance by reducing computational cost (7). A significant challenge is scaling deep neural networks without performance deterioration. In this regard, spectral normalization (8), based on the SVD decomposition of convolution filters, has proven effective in solving this problem by providing a reliable criterion to identify and remove less important channels. Continuous research in these areas is essential to create more efficient and reliable deep reinforcement learning agents capable of solving complex real-world problems.

In this paper, we examine the compression issue in the context of Imitation Learning. We consider the method to manage network's size without requiring a dataset for fine-tuning or retraining. Integral Neural Networks are particularly well-suited for addressing this problem in Reinforcement and Imitation Learning, because structural characteristics of INN, that will be discussed below. We consider an INN as the agent policy function comparing to standard NN. Form of policy was obtained first by TRPO optimization of an expert, followed by imitation training of the INN and NN representations. However, in this case, it does not matter which RL method is used to derive the policy function, since the subject of research is not the specifics of learning, but how much the resulting policy function can be compressed without significant degradation in accuracy.

Method

To explore the feasibility of implementing a policy function through an Integral Neural Network, we employed a pipeline based on behaviour cloning of expert, consisting of the following steps:

1. Optimization of the expert policy using the Trust Region Policy Optimization (TRPO) to derive the optimal policy ($\hat{\pi}(s)$).
2. Aggregation of a dataset comprising (state, action) pairs of observations and actions during the expert rollout testing.
3. Behavior cloning by the searching policy $\pi^{\star}(s)$ that imitates expert $\hat{\pi}(s)$. We trained two $\hat{\pi}(s)$ agents: based on INN and NN with equivalent architectures.
4. Testing of the obtained agents was conducted using the resized INN and NN policies $\pi^{\star}(s)$.

Integral Neural Network

Integral Neural Network (INN) is a type of deep neural networks with continuous layer representation (3). Layers are represented as continuous functions on N-dimensional hypercubes. Discrete transformations are replaced by continuous integration operations.
Let us look to example with dense layer that performs a transform of a vector F_O to a vector F_I as a scalar product. The scalar product can be considered as the integral of the product of two dimensional functions. So, integral representation of the dense layer can be considered as following:

$$F_O(x^{out}) = \int_0^1 F_W(\lambda, x^{out}, x^{in}) \cdot F_I(x^{in}) \cdot dx^{in} \qquad \textbf{(1)}$$

The continuous layer weights are represented by an integrable function:

$$F_W(\lambda, x^{out}, x^{in}) = \sum_{i,j=0}^{i,j=m^{out},m^{in}} \lambda_{i,j} \cdot u(x^{out}m^{out} - i) \cdot u(x^{in}m^{in} - j), \qquad \textbf{(2)}$$

where m^{in}, m^{out} – dimension sizes of chosen grid; $u(x)$ – a kernel function; $\lambda_{i,j}$ – trainable coefficients. The Figure 1 shows the demonstration of the different data flow mechanisms in the dense layer (a) and integral dense layer (b). In the second case, the transformation is not simply the weight matrix, but the sum of the products of functions, each of which is defined in the entire space of the matrix. The centers of the functions are located on the grid with indices i,j. The trainable parameters are the coefficients $\lambda_{i,j}$ that correspond to the impact of each centrer in the grid. This integral representation of layers can also be used in the case of arbitrary dimensions, as well as any other types of layers, such as convolution, pooling, etc.
The main advantages of INNs are that the network can be discretized to an arbitrary size, and there is no need for finetuning. Similar to how sound waves can be sampled at various rates to achieve different sound qualities, INNs can dynamically modify the shapes of data and parameters, thereby affecting the quality of the deep neural network. For example, experiments confirm that the accuracy loss is 2% for the ResNet18 architecture in the Imagenet classification task at a pruning level of 30% (3).
In Appendix I you can see how signal flows inside INN in the example with convolution layers in contrast to classical NN.

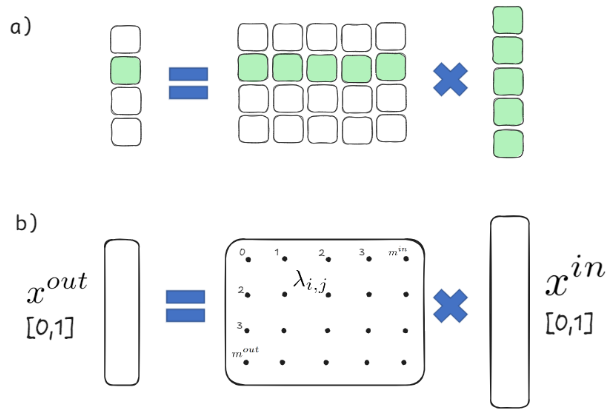


Fig. 1. Demonstration for data flow in dense layer (a) and integral dense layer (b).

Resize operation. When transforming continuous representations using integral operators into discrete weight tensors, it is necessary to select an integration rule and approximate the weights of the integration quadrature q_i:

$$F_O(x^{out}) = \sum_{i=0}^{n} q_i \cdot F_W(x^{out}, x_i^{in}) \cdot F_I(x_i^{in}), \qquad \textbf{(3)}$$

following the trapezoidal rule with

$$q_i = \begin{cases} \frac{\Delta x_1}{2}, & \text{for } i = 0 \\ \frac{\Delta x_i + \Delta x_{i+1}}{2}, & \text{for } i \in [1, ..., m-1] \\ \frac{\Delta x_m}{2}, & \text{for } i = m. \end{cases}$$

The flexibility of the discretization consists of arbitrary choice of m, the size of the grid, which should be compatible with neighbour layers.

Channels permutation. One of the key aspects of creating an INN is minimizing the total variation along a specific dimension of the weight tensor (9). This is accomplished by identifying a permutation that results in the most efficient structure, similar to solving the well-known Traveling Salesman Problem. The permutation process is carried out in such a way that the arrangement of filters in the previous layer corresponds to the ordering of channels in the subsequent layer. Following the permutations model output remains exactly the same.

Experiment

Benchmarks. The paper uses four benchmarks from the OpenAI Gym platform to evaluate the performance of the algorithms and compare their results. These benchmarks are:

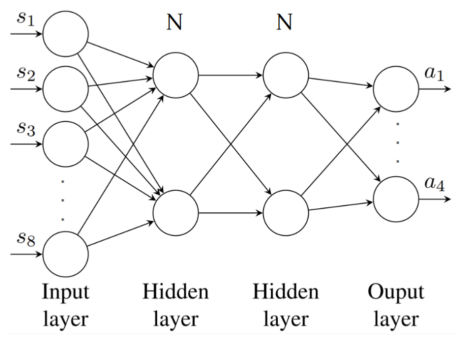


Fig. 2. Neural Network architecture. Policy performance was evaluated as a function of N neurons in hidden layers.

Acrobot: This is a classic control problem where an under-actuated two-link robot (acrobot) must be controlled to swing its end link up and stay in that position (10). The problem we solved for controlling the acrobot is to swing up the joints in a short time. Observation Space is a continuous vector of 6 dimensions. The action space is discrete and consists of 3 possible actions.

LunarLander: In this task, the agent controls a lunar lander spacecraft to safely land on the moon's surface. It must deal with various challenges such as uneven terrain, slopes, and boulders. LunarLander is designed to test the capability of algorithms to handle real-world scenarios with multiple objectives and constraints. The observation space is a continuous vector of 8 dimensions, which includes information about the lander's horizontal position, velocity, angle, and angular velocity, as well as details about the terrain, whether or not the lander is on fire, and the time step. The action space is discrete and consists of 4 possible actions: do nothing, fire left engine, fire right engine, or fire both engines.

HalfCheetah: This benchmark (11) involves controlling a 2D cheetah robot to run as fast as possible. The agent must learn how to coordinate the movement of all four legs to achieve high speeds. HalfCheetah is used to assess the performance of algorithms in continuous control tasks with 17-dimensional state and continuous 6-dimensional action spaces.

Ant: Similar to HalfCheetah, this benchmark requires controlling an ant robot to move forward as quickly as possible. However, the ant has more legs and a more complex body structure, making the control task even more challenging. Ant-v4 is also used to evaluate algorithms in continuous control environments with higher complexity. Observation and Action Space of Ant is a continuous vector of 105 and 8 dimensions,respectively.

These benchmarks provide a diverse set of environments to test and compare the performance on different tasks, starting from simple Acrobot. All problems are implemented using the Gym framework, developed by OpenAI (12). Gym is a toolkit for creating and comparing reinforcement learning algorithms. It offers a diverse range of environments, from Atari games to robotics simulations.

Agent Model. We employed a three-layer perceptron architecture ($8\times100\times100\times4$), depicted in Fig. 2, with N = 100 neurons in each hidden layers. ReLU activation is used in the hidden layers, while the output layer utilizes LINEAR activation. To investigate the impact of network size on policy accuracy, we vary the number of hidden neurons (N) from 100 to lower values.

Expert model training. The details of expert training are provided in Appendix II. About 12 thousand episodes were used for TRPO training for Acrobot, Lunar Lander problems, and 1 million episods for HalfCheetah and Ant.

Data aggregation. To teach the student model through imitation learning, we collected a dataset consisting of (action, state) pairs from 10000 expert episodes that were completed successfully. These data were used for training, while 15 episodes served for validation.

Although it would be beneficial to employ a more sophisticated approach to data aggregation, such as Dagger, which entails multiple iterations of model training and repeated data collection in cases where the student fails using the expert's model, this paper concentrates on evaluating the efficacy of the integral representation of an agent's policy and performance in the context of model compression. Achieving high reward values is not within the scope of our current study.

Imitation learning. Student policy functions (INN and NN) were obtained by supervised learning techniques to minimize the divergence between the learned policy ($\pi^\star(s)$) and the expert ($\hat{\pi}(s)$) demonstrations according to a chosen metric. In the context of the Acrobot and Lunar Lander environment, which features a discrete observation space, cross-entropy was selected as the minimization objective function. For continuous N-dimensional action spaces MSE loss function was chosen. We used the Adam optimizer with an learning rate of 0.001. The training data carried out on 30 epochs. From a technical point of view, training an INN is completely equivalent to training a regular NN. The only difference is that the derivatives of the loss function are calculated not with respect to the weight values, but with respect to the λ coefficients. The same procedure of imitation learning was carried out for the series of NN representations of agent policy function.

Results

Fig. 3 and tables in Appendix III illustrate the correlation between the average reward per episode (AR) and the size of the network, with different sizes corresponding to various numbers of neurons (N) in the hidden layers.

The INN and NN networks with N = 100 were trained separately using the same pipeline and hyperparameter settings. We did not set a fixed seed for the random generator in the torch library and Gym environment and conducted 10 different iterations for training the NN and INN to demonstrate statistical fluctuations during data aggregation, imitation learning, and testing. In addition to AR, the tables include the standard deviation of the reward spectrum. The standard deviation is also presented in Fig. 3 as a shaded band. The last column in the tables shows the p-value of the null hypothesis

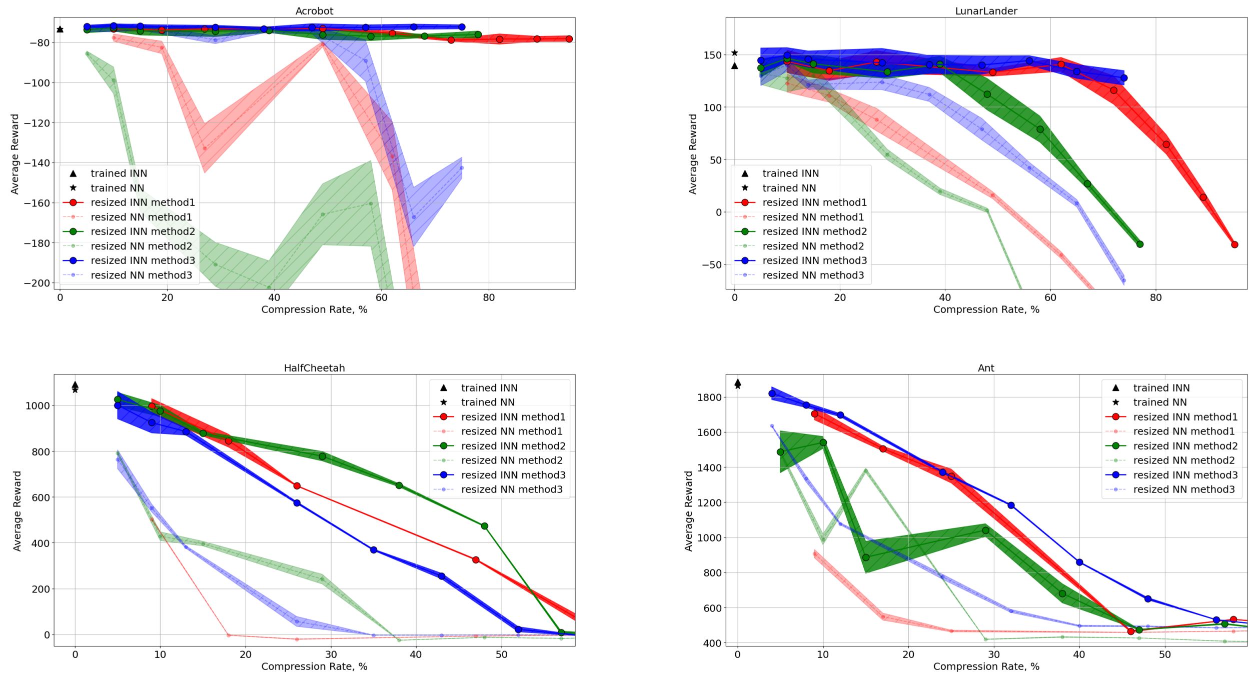


Fig. 3. Average reward plotted against model size for ten random runs of the pipeline in the following environments: Acrobot (top left), Lunar Lander (top right), Half Cheetah (bottom left) and Ant (bottom right).

for comparing the mean values of two NN and INN samples, calculated using the Welch's t-test.

The columns labeled "resized" refer to the tests of networks that have been resized. The process of resizing the networks was carried out using the torchvision.transforms.functional.resize function (13) with bicubic interpolation mode for MLP weights and biases. This approach has proven to be the most effective, thus we present the results based solely on it.

The video demonstration with INN agent model, compressed by 24.3%, played on Ant benchmark is hosted here, while agent playing with original full size INN can be seen in the gif here.

Discussion

The first conclusion we made in the experiment is that INN model is trainable like NN in terms of epoch number and accuracy. Furthermore, INN is parametrized using normalization coefficients λ_i, which form a standart torch.Tensor with grad_fn attribute. This allows applying the standard learning process as for conventional networks. Nevertheless, the additional parametrization and discretization in the forward pass approximately double the learning time.

The second observation in the setup is that INN compression capabilities significantly exceed NN. During the process of changing the size of a neural network, the INN agent maintains relatively high AR values in the case of NN. In simpler tasks, altering the network size does not lead to a degradation of AR down to the level of INN compression of 90–95% (top Figs. 3). This is because the selected network has been chosen with a large margin. Such high performance is achieved at a significant computational cost, due to their strong optimization and generalization abilities in the region of high overparameterization. In more complex tasks (bottom Figs. 3) of continuous multidimensional robot control, reducing the model size inevitably leads to its degradation. However, the integral representation of the agent allows maintaining relatively high AR values.

To illustrate the difference between INN and conventional NN, we can look at the weight matrix on the second dense layer in Fig. 4 for NN (left) and INN (right) used for Lunar Lander problem. The weights in the NN are randomly distributed, and resemble polychromatic random noise. The weights in the INN are distributed over a smoother surface. This behaviour indicates that for this imitation learning task, the size of this layer is excessive and can be reduced.

The Fig. 5 shows the mean absolute values of the one-dimensional Fourier transform (1D FFT) of rows in the two-dimensional weight matrix of the second layer. This analysis provides insights into the frequency content of the weights. NN frequencies are distributed uniformly; however, INN is dominated in low frequency region. Such a spectrum can serve as an indicator of how much the INN differs from the NN and how to choose the right grid for the INN discretization. It is also possible to cut off high frequencies and perform the inverse Fourier transform, thereby performing pruning.

It should be emphasized that INN and current research with model resizing should not be considered as an alternative to manifold pruning algorithms of networks. Training of INN, the subsequent resizing operation, and the transition from a parameterized model to a non-parameterized one can be considered as the first step, followed by sophisticated methods of

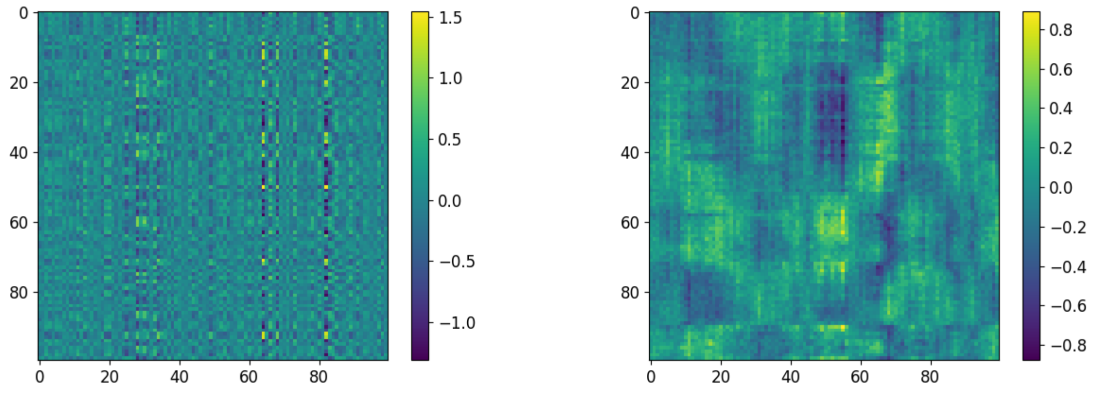


Fig. 4. 2D weight matrix 100×100 of second layer for NN (left) and INN (right) used in Lunar Lander problem.

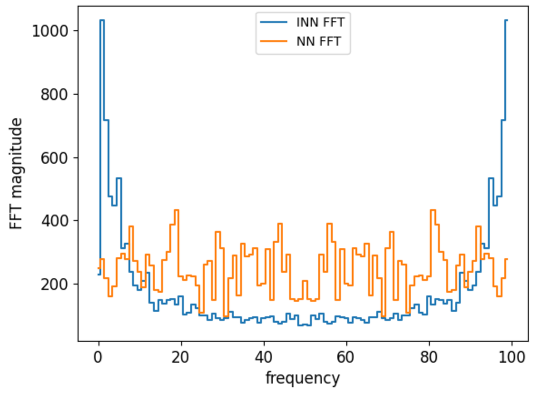


Fig. 5. Mean absolute values of 1D FFT of rows of 2D weight matrix of second layer

structured or unstructured pruning. And unlike pruning, the resizing operation necessarily leads to a reduction in computational cost, because it does not require switching to the representation of sparse matrices.

The main application of resizable network is that it can be adjusted to hardware resources. Limitations on the model size may be related to both the available RAM size on the chip and the required level of performance and/or latency. For example, when driving in harsh conditions or at high speed, the latency of the model response may be critical, and in such a case it is reasonable to use a diminished model.

This ability to control the size of the INN can be useful in other applications. For example, in measuring the size of captured information ("intelligence") in the network. If your goal is to choose a sufficient size for a neural network or layer, you can follow this pipeline: 1. Create abundant architecture with a reserve in the number of channels and the number of neurons per channel; 2. Initialize INN with the chosen NN; 3. Perform INN learning as usual; 4. Compress the INN to size when it starts to degrade, which means that you have reached the limit of the network size you are looking for.

Another promising avenue in the development of artificial intelligence is analog computing, which significantly outperforms digital computing in terms of speed, energy efficiency, radiation resistance, and etc (14), (15). Analog computing and INN are synergistic, as they both operate with continuous functions, serving as activations and transformation kernels, rather than discrete values.

One notable company in this field is POLYN Technology (16), which offers commercial solutions in Neuromorphic Analog Signal Processing. Consequently, INN are closely aligned with the algorithms of future technologies.

Conclusions

This paper proposes a new approach to representing an agent's policy model as an integrated neural network, which shows promise as a potentially effective strategy. By utilizing open AI benchmarks, we have demonstrated that this representation provides flexible control over the degree of network discretization without necessitating retraining across a broad spectrum of model compression levels. This allows users to directly tailor the balance between network size and accuracy. INN can be used as a capability to quantify the "artificial intelligence" inherent within the network, potentially informing architecture selection. Further investigation into the advantages of this approach, including mitigating catastrophic forgetting and addressing noisy environments, will be the subject of future research. Another promising area of research is the implementation of online reinforcement learning with the INN policy function, as well as quantization of INN to reduce computational cost.

Bibliography

1. G. Dellaferrera, S. Woźniak, and G. Indiveri et al. Introducing principles of synaptic integration in the optimization of deep neural networks. *Nat Commun*, 13:1885, 2022. doi: 10.1038/s41467-022-29491-2.
2. R. Iyer, V. Menon, M. Buice, C. Koch, and S. Mihalas. The Influence of Synaptic Weight Distribution on Neuronal Population Dynamics. *PLoS Comput Biol*, 9(10): e1003248, 2013. doi: 10.1371/journal.pcbi.1003248.
3. Kirill Solodskikh, Azim Kurbanov, Ruslan Aydarkhanov, Irina Zhelavskaya, Yury Parfenov, Dehua Song, and Stamatios Lefkimmiatis. Integral neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16113–16122, June 2023.
4. J Obando-Ceron, Pablo Samuel Castro, and Aaron Courville. In deep reinforcement learning, a pruned network is a good network. *ArXiv*, 2024. doi: https://arxiv.org/html/2402.12479v1.
5. Samin Yeasar Arnob, Riyasat Ohib, S. Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *ArXiv*, abs/2112.15579, 2021.
6. Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.07.045.

7. Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural network distiller: A python package for dnn compression research. October 2019.
8. Arjan Matthams, Scarlet Moreno, and Sadia Swanson. Spectral Norm Pruning: A Stability-Driven Approach to Deep Neural Network Compression. working paper or preprint, September 2024.
9. Solodskikh K. Kurbanov A. Torchintegral. https://github.com/TheStageAI/TorchIntegral, 2023. Accessed: 2024-11-08.
10. Richard S. Sutton. Generalization in reinforcement learning: successful examples using sparse coarse coding. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 1038–1044, Cambridge, MA, USA, 1995. MIT Press.
11. Half cheetah. https://gymnasium.farama.org/environments/mujoco/half_cheetah/. Accessed: 2024-11-16.
12. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
13. Torchvision.transforms. https://pytorch.org/vision/main/generated/torchvision.transforms.functional.resize.html. Accessed: 2024-11-05.
14. Yuan Du, Li Du, Xuefeng Gu, Jieqiong Du, X. Shawn Wang, Boyu Hu, Mingzhe Jiang, Xiaoliang Chen, Subramanian S. Iyer, and Mau-Chung Frank Chang. An analog neural network computing engine using cmos-compatible charge-trap-transistor (ctt). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(10):1811–1819, 2019. doi: 10.1109/TCAD.2018.2859237.
15. Olga Krestinskaya, Khaled Nabil Salama, and Alex Pappachen James. Learning in memristive neural network architectures using analog backpropagation circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(2):719–732, February 2019. ISSN 1558-0806. doi: 10.1109/tcsi.2018.2866510.
16. Polyn technology. https://polyn.ai/. Accessed: 2024-11-05.
17. John Schulman. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

Appendix I. Integral CNN.

To more clearly demonstrate the properties of the INN, a convolutional neural network analysis was performed. Thus, the model *ModelMnist* was trained on the MNIST dataset, which consists of three convolutional layers followed by ReLU activation, AvgPool2d and one dense layer. Training was carried out over two epochs. The classification accuracy on the validation dataset was approximately 97%.
Figures 6, 7 show the feature maps for separately trained INN and NN, respectively. Feature maps were collected on the first (a) and second (b) layers before activation. To emphasize internal network transformations, we used the sum of two figures as input with "6" and "7" labels. Feature maps on the first and second layers correspond to filters with dimensions $1 \times 3 \times 3$ and $16 \times 5 \times 5$, respectively. Figures demonstrate that the feature maps of the integral network are arranged in a more continuous manner compared to those of the discrete network. This continuous behaviour allows the user to control the level of discretization, while still allowing the layers to convey the correct spatial structure of the activation tensor.

Appendix II. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) (17) is an important algorithm in RL that ensures large policy updates do not deteriorate performance. We follow the next TRPO pipeline:
The surrogate objective function for the policy can be defined as:

$$L(\theta_{\text{old}}, \theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right] \quad \textbf{(4)}$$

where A is the advantage function for the current policy.
The constraint to ensure the policy does not deviate too much can be defined using the KL divergence:

$$\overline{D}_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta) = \mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\left[D_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta)\right] \leq \delta \quad \textbf{(5)}$$

The overall optimization problem in TRPO can be stated as:

$$\begin{aligned} \text{maximize} \quad & L(\theta_{\text{old}}, \theta) \\ \text{s.t.} \quad & \overline{D}_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta) \leq \delta \end{aligned}$$

In the first and second Taylor expansion we can approximate via g – gradient of L and H – Hessian of $\overline{D}_{KL}$:

$$\begin{aligned} L(\theta_{\text{old}}, \theta) &\approx g^T \cdot (\theta - \theta_{\text{old}}) \\ \overline{D}_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta) &\approx \frac{1}{2}(\theta - \theta_{\text{old}})^{\mathrm{T}} \cdot \mathrm{H} \cdot (\theta - \theta_{\text{old}}). \end{aligned}$$

In these notations, the approximation of the problem is as follows:

$$\begin{aligned} \theta_{\text{new}} &= argmax_\theta \; g^{\mathrm{T}} \cdot (\theta - \theta_{\text{old}}) \\ \text{s.t. } & \frac{1}{2}(\theta - \theta_{\text{old}})^{\mathrm{T}} \cdot \mathrm{H} \cdot (\theta - \theta_{\text{old}}) \leq \delta \end{aligned}$$

This approximate problem can be solved analytically by Lagrangian duality methods, which gives the solution for the weights vector:

$$\theta_{\text{new}} = \theta_{\text{old}} + \sqrt{\frac{2\delta}{g^{\mathrm{T}} \cdot \mathrm{H}^{-1} \cdot g}}\, \mathrm{H}^{-1} \cdot g. \quad \textbf{(6)}$$

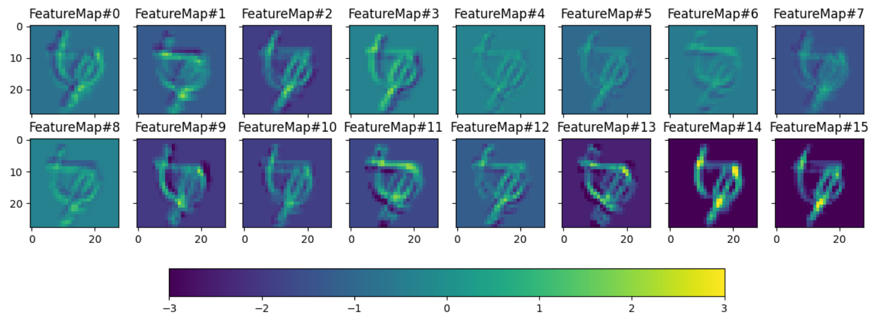
To find $x = \mathrm{H}^{-1} \cdot g$, we solved equation $\mathrm{H}x = g$ by iterative conjugate gradient method.
To be sure that the constraint to KL divergence is satisfied a backtracking line search is used:

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha_j \sqrt{\frac{2\delta}{g^{\mathrm{T}} \cdot \mathrm{H}^{-1} \cdot g}}\, \mathrm{H}^{-1} \cdot g, \quad \textbf{(7)}$$

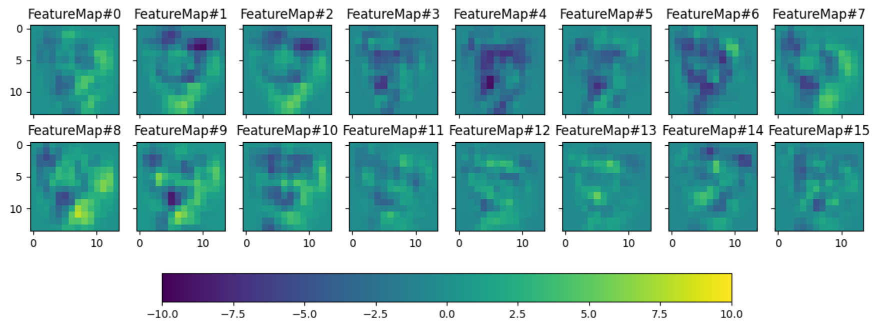
where $\alpha_j = 0.5^j$ is the backtracking coefficient, and j is a minimal positive integer $0..10$ such that satisfies the KL constraint and produces a smaller value of the surrogate objective function (4). If such j was not found, $\alpha = 0$ was used.
The TRPO method is considered more cumbersome, since it requires solving a conditional optimization problem at each step of the algorithm, but it is more efficient in terms of the number of iterations. The optimization process was carried out over 12 thousand episodes for the Acrobot and Lanar Lander problems. And one million episodes are used for more complex problems: Half Cheetah and Ant.
We used 10 iterations for calculation of $\mathrm{H}^{-1} \cdot g$ in (6) by using conjugate gradient method. The value of KL divergence was constrained by the maximum value $\delta = 0.01$.

Appendix III. Tables.

The tables below show the dependencies of Average Reward on the size of the agent's neural network, either in the form of NN or INN. The first line in the tables corresponds to trained versions of the networks. The lines below demonstrate the behavior of compressed versions of neural networks, which are obtained simply by changing the dimensions of the internal layers of the network.

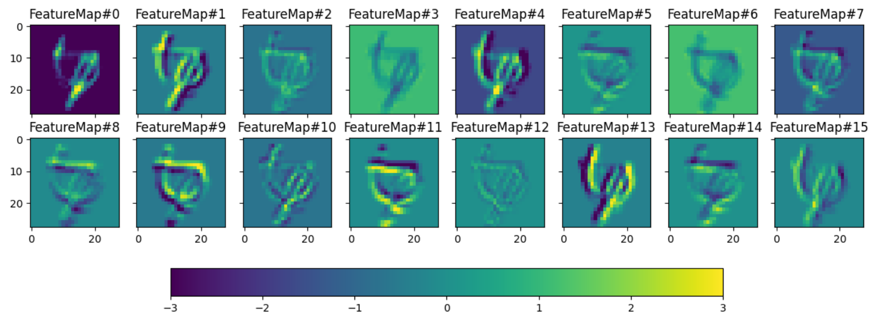


(a)

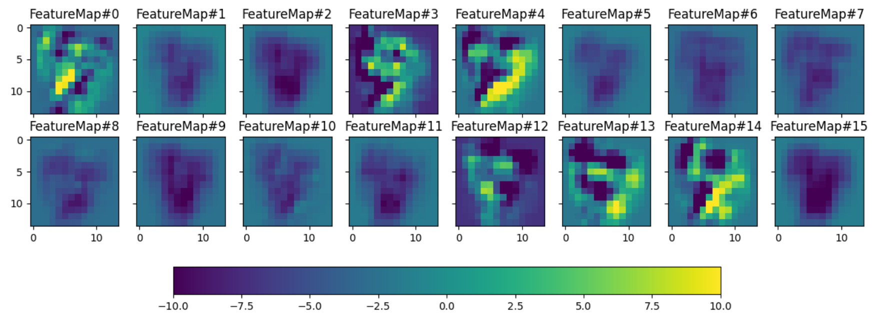


(b)

Fig. 6. Visualization of feature maps after the first (a) and second (b) convolutional layers in Integral *ModelMnist*



(a)



(b)

Fig. 7. Visualization of feature maps after the first (a) and second (b) convolutional layers in Discrete *ModelMnist*

N *on hidden layers*	Compression rate, %	AR, std *(trained INN)*	AR, std *(resized INN)*	AR, std *(trained NN)*	AR, std *(resized NN)*	p-value
100	0.0	-71 , 1		-72 , 1		6.9E-02
95	9.3		-72 , 1		-79 , 2	6.9E-08
90	18.1		-73 , 1		-74 , 2	1.1E-01
85	26.5		-73 , 1		-74 , 2	3.1E-01
70	48.9		-72 , 2		-77 , 3	2.6E-04
60	61.6		-73 , 1		-83 , 10	8.9E-03
50	72.5		-74 , 2		-82 , 2	7.4E-08
40	81.6		-76 , 4		-116 , 49	3.1E-02
30	88.9		-82 , 17		-495 , 11	7.3E-20
20	94.4		-110 , 50		-302 , 150	2.7E-03
10	98.1		-233 , 154		-500 , 0	3.9E-04
5	99.2		-409 , 160		-500 , 0	1.1E-01

Table 1. The dependence of average reward (AR), standard deviation on size of NN/INN in Acrobot-v1 benchmark based on validation runs.

N *on hidden layers*	Compression rate, %	AR, std *(trained INN)*	AR, std *(resized INN)*	AR, std *(trained NN)*	AR, std *(resized NN)*	p-value
100	0.0	152 , 6		152 , 13		8.8E-01
95	9.2		152 , 12		145 , 9	1.5E-01
90	17.9		150 , 9		150 , 6	9.1E-01
85	26.2		149 , 15		147 , 8	7.1E-01
70	48.4		150 , 5		141 , 7	2.5E-03
60	61.0		154 , 4		141 , 11	4.7E-03
50	71.9		154 , 9		138 , 9	4.5E-04
40	81.0		151 , 8		88 , 19	3.6E-07
30	88.4		145 , 10		93 , 23	2.0E-05
20	94.0		144 , 18		-65 , 37	4.4E-10
10	97.9		77 , 48		-526 , 7	9.6E-12
5	99.1		-85 , 91		-524 , 7	8.7E-08

Table 2. The dependence of average reward (AR), standard deviation on size of NN/INN in Lunar Lander v3 benchmark based on validation runs.

N *on hidden layers*	Compression rate, %	AR, std *(trained INN)*	AR, std *(resized INN)*	AR, std *(trained NN)*	AR, std *(resized NN)*	p-value
100	0.0	1093 , 36		1083 , 58		6.6E-01
95	8.8		563 , 159		25 , 54	6.2E-07
90	17.2		355 , 152		-52 , 10	1.3E-05
85	25.2		250 , 128		-50 , 6	3.9E-05
70	46.8		43 , 60		-24 , 3	6.1E-03
60	59.2		-42 , 6		-18 , 3	9.8E-08
50	70.0		-30 , 5		-13 , 2	3.6E-07
40	79.2		-20 , 3		-17 , 1	1.1E-02
30	86.8		-14 , 2		-15 , 1	6.7E-01
20	92.7		-17 , 1		-13 , 1	2.2E-06
10	97.1		-14 , 1		-12 , 1	5.7E-06
5	98.7		-13 , 1		-11 , 1	2.4E-03

Table 3. The dependence of average reward (AR), standard deviation on size of NN/INN in HalfCheetah-v4 benchmark based on validation runs.

N *on hidden layers*	Compression rate, %	AR, std *(trained INN)*	AR, std *(resized INN)*	AR, std *(trained NN)*	AR, std *(resized NN)*	p-value
100	0.0	1818 , 21		1832 , 24		1.7E-01
95	8.5		1235 , 265		855 , 156	1.5E-03
90	16.6		1020 , 356		748 , 195	5.2E-02
85	24.3		1116 , 338		667 , 118	2.2E-03
70	45.3		665 , 157		517 , 28	1.6E-02
60	57.5		517 , 64		493 , 11	2.6E-01
50	68.2		486 , 8		488 , 2	5.1E-01
40	77.5		489 , 8		488 , 1	6.6E-01
30	85.3		491 , 6		490 , 1	4.2E-01
20	91.6		493 , 5		490 , 1	1.3E-01
10	96.5		495 , 5		490 , 1	1.7E-02
5	98.4		496 , 5		490 , 1	6.2E-03

Table 4. The dependence of average reward (AR), standard deviation on size of NN/INN in Ant-v4 benchmark based on validation runs.