Robustness to Network Resizing and Catastrophic Interference in Imitation Learning: Kolmogorov–Arnold and Integral Neural Networks

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

[2]Department of Computer Science, UCLA

The paper examines the issue of catastrophic forgetting and online resizing in the context of imitation learning. The research compares agents based on Multilayer Perceptron Neural Networks (NN), Kolmogorov–Arnold Networks (KAN) and Integral Neural Networks (INN). The study confirms that KAN agents are less prone to catastrophic forgetting compared to NN and INN-based agents. This behavior makes them promising for continuous learning in complex environments with long episodes. These findings contribute to a deeper understanding of catastrophic forgetting and provide valuable insights into the potential of KANs for improving the performance of reinforcement learning algorithms. On the other hand, 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. Code is available at git.

Kolmogorov-Arnold Networks | Catastrophic Forgetting | Continual Learning | Integral Neural Network | Compression | Imitation Learning | Behavioral Cloning

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 network resizing 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 discrete control tasks Acrobot, Lunar Lander and on continuous control tasks Swimmer, Walker2D, Half Cheetah and Ant.

In the rapidly evolving field of artificial intelligence (AI), one of the key challenges is to develop models that can effectively learn and adapt to new data keeping previous datasets information. The results of the paper (4) indicate that the size of a model does not ensure better performance in continual learning. In fact, larger models often find it more difficult to adapt to new tasks, especially in online settings. These findings contradict the idea that large models can inherently reduce catastrophic forgetting. They emphasize the complex relationship between the size of the model and its effectiveness in continual learning.

The success of AI products depends not only on accuracy and specificity but also on factors like continuous learning ability, resistance to catastrophic forgetting, latency, and other aspects. Previously, the authors (5–7) showed that there is a special kind of neural networks called KANs (Kolmogorov–Arnold Networks), which have a remarkable property of being able to naturally work in continual learning without catastrophic forgetting. KAN and MLP have fundamental functional differences. Empirical studies have shown that KAN excels in data fitting and other tasks, demonstrating that even smaller KAN-based models can achieve or surpass the performance of MLP-based models in these areas (8). However, there are still some unexplored aspects of KAN's capabilities, including their behavior under shifted domain conditions.

The objective of this paper is to compare three neural network architectures: KAN, INN and NN (MLP), in the context of model resizing and catastrophic forgetting. To test these architectures, we employed tasks and environments from the field of Reinforcement Learning of robots. In order to expedite the learning process of the models, we conducted experiments utilizing the technique of imitation learning.

Related work

The authors (9) consider dynamic transformer architecture for continuous learning of multi-modal tasks. They proposed new method – Task Attentive Multi-modal Continual Learning (TAM-CL) exploiting dynamic model expansion to sequentially learn tasks. It involves knowledge distillation to extract experience from the past. It enables information exchange between tasks and mitigates the problem of catastrophic forgetting.

Reinforcement and Imitation 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 (10). 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 (11). 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 (12). 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 (13). A significant challenge is scaling deep neural networks without performance deterioration. In this regard, spectral normalization (14), 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.

Pruning often fails to accelerate inference, and only network resizing can be of help. The most basic approach is to train a network with various layer sizes to adapt the network for efficient operation at different scales. However, this technique is computationally complex and limits the precision that can be achieved (15).

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.

Several articles have already been published on the use of KAN in solving RL problems (8),(5). However, for the first time, we created artificial domain shift to see how neural networks are resistant to catastrophic forgetting. We visualize the process of forgetting and explain the reason for this behavior, which is associated with the internal structure of the networks. Our work expands on previous research in several ways. We focus on studying the behavior of neural networks under conditions of artificial domain shifting. This allows us to visualize the process of forgetting and to more deeply understand the mechanisms of catastrophic forgetting and develop methods to overcome it. We explain the cause of such behavior, which is related to the internal structure of the networks.

Method

To explore the feasibility of implementing a policy function through an INN, KAN and traditional MLP, 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 observation and action vectors, collected during the expert rollout testing.
3. Imitation learning of the new agent with policy $\pi^{\star}(s)$ that is supposed to clone expert $\hat{\pi}(s)$. We trained three $\hat{\pi}(s)$ agents: based on NN, INN and KAN. The same three layer architecture was used for the three agent types and for all tested environments.
4. Experiment 1. Obtained NN and INN agents $\pi^{\star}(s)$ were resized and compared with original policies.
5. Experiment 2. To assess the networks' ability for continual learning, the acquired models of NN, INN, and KAN were subjected to additional training on shifted input data. To implement the shift, we add a constant to the vector of the observation state.

Kolmogorov–Arnold Networks

The KAN represent a new paradigm for designing and training neural networks. KANs employ sophisticated learning functions to represent nonlinear input transformations with fewer parameters compared to traditional neural networks (5, 6). In a neural network, activation functions are fixed throughout the network and exist at each node (see Fig. 1). In KANs, however, activation functions exist on edges, which correspond to the weights of a traditional neural network. Each edge has its own activation function that adapts during the learning process.

The universal approximation theorem and the Kolmogorov–Arnold theorem are the theoretical foundations for KANs. The theorem states that any multivariate continuous

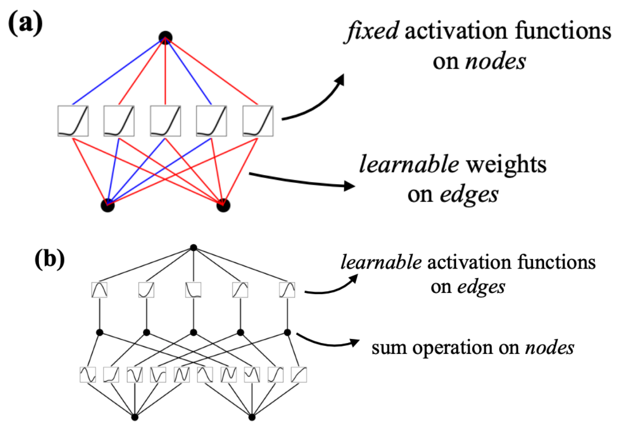

(a)
fixed activation functions on *nodes*
learnable weights on *edges*
(b)
learnable activation functions on *edges*
sum operation on *nodes*

Fig. 1. The illustration of key difference between the structures of standard NN (a) and KAN (b) (16).

function can be expressed as a composition of a finite number of univariate continuous functions. KANs demonstrate better generalization ability and retain knowledge when switching to new tasks or adding new data. Whereas traditional NNs quickly forget old knowledge, because global activations allow propagating changes to distant regions on the backward pass stage.

In this work we used parametrization RELU-KAN proposed by authors (7):

$$R_i(x) = [ReLU(e_i - x) \times ReLU(x - s_i)]^2 \times \frac{16}{(e_i - s_i)^4}, \quad (1)$$

where the last factor is used as normalization such that the maximum value of R_i is 1. The position of the function is defined by the trainable parameters e_i and s_i. Each edge between two neurons of the ReLU-KAN layer can also be expressed by the superposition:

$$\phi(x) = \sum_{i=1}^{G+k} \omega_i R_i(x), \quad (2)$$

with trainable normalization factors w_i, $G+k$ determine the dimension of the superposition with initial values:

$$s_i = \frac{i-k-1}{G}, \; e_i = \frac{i}{G}, \quad (3)$$

which are trained.

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:

$$F_O(x^{out}) = \int_0^1 F_W(\lambda, x^{out}, x^{in}) \cdot F_I(x^{in}) \cdot dx^{in} \quad (4)$$

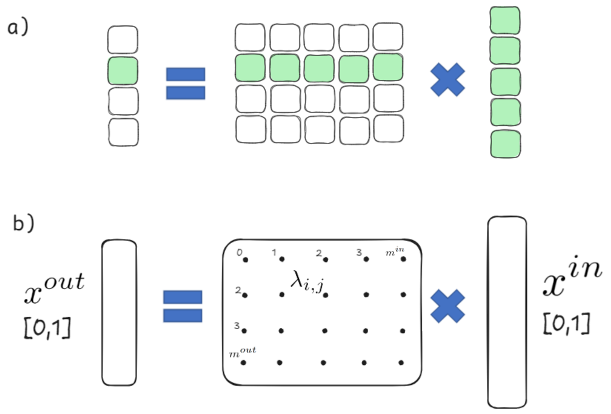


Fig. 2. Demonstration for data flow in dense layer (a) and integral dense layer (b).

The continuous layer weights are represented by an integrable function:

$$F_W(\lambda, x^{out}, x^{in}) = \sum_{i,j=0}^{i,j=m^{out},m^{in}} \lambda_{i,j} \cdot u(x^{out} m^{out} - i) \cdot u(x^{in} m^{in} - j), \quad (5)$$

where m^{in},m^{out} – dimension sizes of chosen grid; $u(x)$ – a kernel function. For efficiency purposes, authors suggested to rely on the cubic convolutional interpolation, which is used for efficient image interpolation on GPUs. $\lambda_{i,j}$ – trainable coefficients. The Figure 2 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 center 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 relativelyarbitrary size, and there is no need for fine-tuning. 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 Supplementary Note 1 you can observe the flow of signals within INN in the example with convolutional layers, as opposed to classical neural networks.

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:

$$F_O(x^{out}) = \sum_{i=0}^{n} q_i \cdot F_W(x^{out}, x_i^{in}) \cdot F_I(x_i^{in}), \qquad (6)$$

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 neighbor 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 (17). 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. The permutation is involved in the training process.

Benchmarks

We used six benchmarks from the OpenAI Gym platform (18) to evaluate the performance of the algorithms and compare their results. Dimensions of used environments are shown in Table 1.

Acrobot: This is a classic control problem where an underactuated two-link robot (acrobot) must be controlled to swing its end link up and stay in that position. The problem we solved for controlling the acrobot is to swing up the joints in a short time.

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.

Swimmer: The aim is to move as quickly as possible towards the right by applying torque to the rotors and utilizing fluid friction.

Walker2D: The objective is to move forward by applying torque to the six hinges that connect the seven body parts.

HalfCheetah: This benchmark 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.

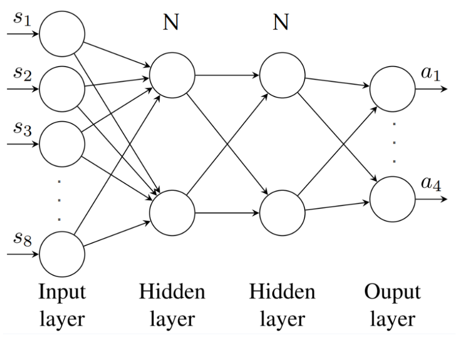


Fig. 3. Neural Network architecture. Policy performance was evaluated as a function of N neurons in hidden layers.

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.

Environment	Continuous observation space (s)	Action space (a)
Acrobot (19)	(6)	discrete (3)
LunarLander	(8)	discrete (4)
Swimmer	(8)	continuous (2)
Walker2d	(17)	continuous (6)
HalfCheetah (20)	(17)	continuous (6)
Ant	(105)	continuous (8)

Table 1. Dimensions of used environments

These benchmarks offer a wide range of environments to test and compare performance across various tasks, beginning with the simple Acrobot and ending with Ant, which has a 105-dimensional state vector.

Experiment

Agent Model. We employed a three-layer perceptron architecture (s×100×100×a), depicted in Fig. 3, 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 Supplymentary Note 2. For TRPO training, 100 thousand episodes were used for Acrobot and Lunar Lander problems. For other tasks, the number of episodes was 1 million.

Data aggregation. To teach the student model through imitation learning, we collected a dataset consisting of pairs (action-state) from 10000 expert episodes that were completed successfully. These data were used for training, while 150 episodes served for validation.

Although it would be advantageous to use a more advanced method of data aggregation, such as Dagger (21), which involves multiple rounds of model training and repeated data collection when the student fails using the expert's model, this paper focuses on assessing the effectiveness of the integral and KAN representation of an agent's policy and performance in the context of model compression and continual learning. Obtaining high reward values is not the goal of our current research.

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 of INN and KAN 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.

Network resizing. All experiments in the paper we performed with the architecture shown in Fig. 3 with layers dimensions: (s = observation shape (state), N, N, a = action shape), where initial N = 100. Dimensions of resized network equal to (s, **N'**, **N'**, a) with **N'** $<$ N. The resizing of networks layers was carried out using standard method – the torchvision.transforms.functional.resize function (22). We used three different InterpolationMode: *BICUBIC*, *BILINEAR*, *NEAREST*.

1. *BICUBIC* interpolation is a method that uses a cubic polynomial to interpolate the values of pixels in the output tensor. This method is more accurate than bilinear interpolation but can sometimes produce blurry results.

2. *BILINEAR* interpolation uses linear interpolation to calculate the new pixel values of resized tensor. This method produces smoother results than nearest-neighbor interpolation but may not be as sharp.

3. *NEAREST* interpolation selects the closest pixel value from the input tensor to determine the new pixel value. This method preserves the sharpness of the original tensor but losses information if we decrease the size.

Which method should be used to resize the weight tensor is an open question. However, intuition suggests that if trained INN have smoother weight distributions than regular networks, then bicubic interpolation would be a more advantageous option when resizing.
It should be emphasized separately that the considered resizing can be applied together with other pruning methods. However, unlike pruning, resizing directly reduces the size of the network and the processing time for input data.

We do not show the results with resizing KAN neural networks. Our attempts to change the connectivity of KAN were unsuccessful, because each edge in it is trained on a specific feature of the original data. Resizing methods for such types of networks seem to be the subject of future research.

Retraining with shifted domain. To test how agents are protected from catastrophic forgetting, we artificially created a learning pipeline where the input data is biased. We added a constant to the vector of observation space such that there is no overlapping between previous and new spaces. That allows to continue learning on shifted domain without change in action space. Such an approach can be implemented in practice when, for example, a robot with optical cameras, after being trained in one room, moves to another room with different lighting.

Results

A. Resizing. Fig. 4 illustrates the correlation between the average return per episode (AR) and the size of the network, with different sizes corresponding to various numbers of neurons (N) in the hidden layers. Red, green and blue colors correspond different resizing types: bicubic, bilinear and nearest, respectively. The compression rate is defined as ratio of models sizes in bytes:

$$\text{Compression Rate} = 1 - \frac{\text{MB(resized model)}}{\text{MB(initial model)}} \quad \textbf{(7)}$$

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.
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 git here.

B. Continual learning. The Fig. 5 shows how AR behaves after we begin training on shifted data for NN, INN and KAN in various environments: Acrobot (top left), Lunar Lander (top right), Swimmer (middle left), Walker2d (middle right), Half Cheetah (bottom left) and Ant (bottom right). The horizontal axis shows the number of events that were used when retraining on shifted data. In fact, this is the number of iterations of backpropagation or weight updates.

Discussion

The first conclusion we made in the research is that INN and KAN models is trainable like NN in terms of epoch number and accuracy and can be used in RL and Imitation Learning. Furthermore, INN and KAN is parametrized using normalization and other coefficients, which form a standard torch.Tensor with grad_fn attribute. This allows applying the

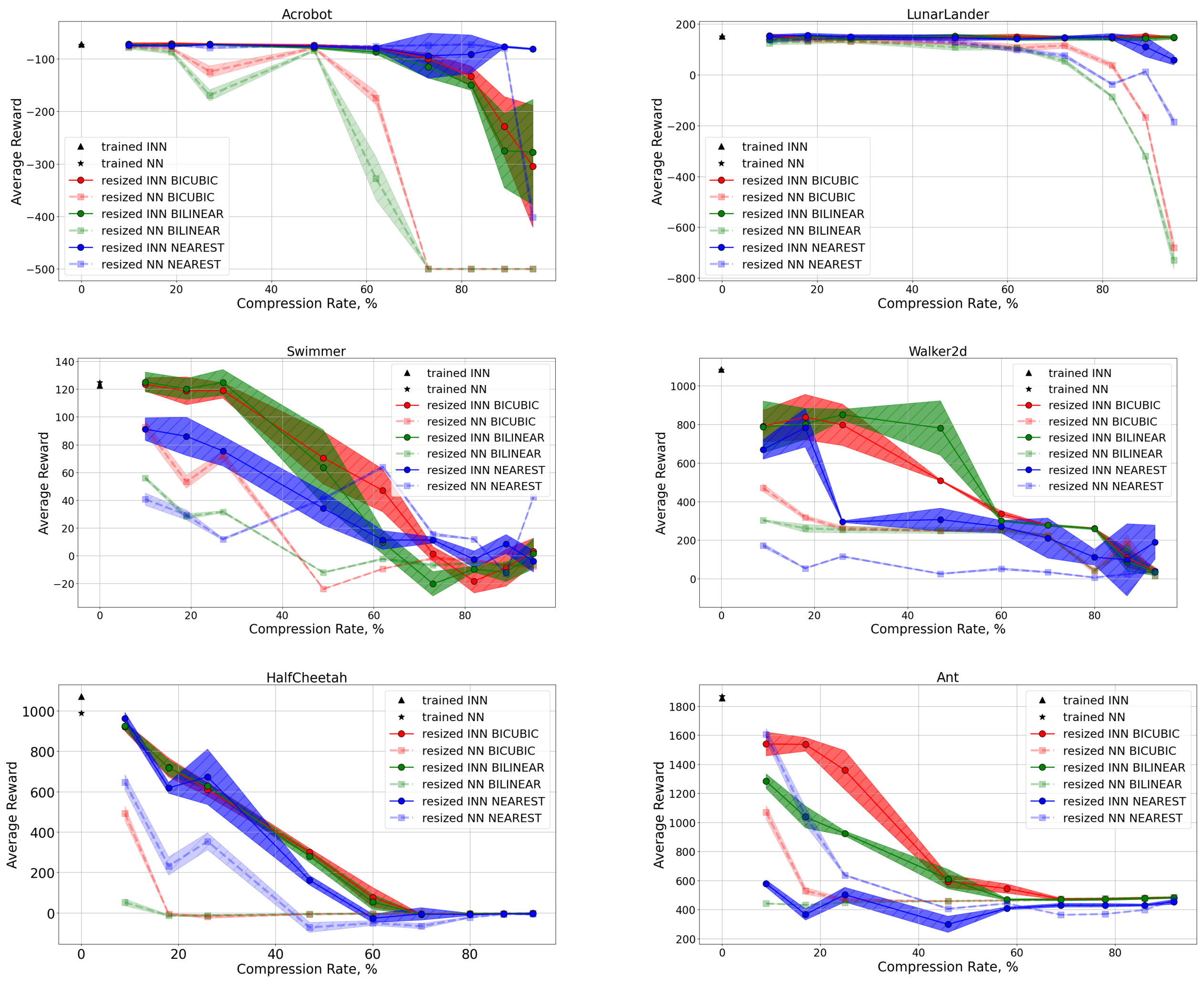


Fig. 4. Average return plotted against model size for ten random runs of the pipeline in the following environments: Acrobot (top left), Lunar Lander (top right), Swimmer (middle left), Walker2d (middle right), Half Cheetah (bottom left) and Ant (bottom right). Red, green and blue colors correspond different resizing types: bicubic, bilinear and nearest, respectively.

standard learning process as for conventional networks. We chose Imitation Learning primarily to save time in model acquisition. From a technical standpoint, KAN and INN can be employed as alternatives to NN in any AI task, including RL. Nevertheless, the additional parametrization and discretization in the forward pass approximately double the learning time for KAN and INN relatively to NN. The AR of the trained agents, calculated during test episodes, is represented by star markers in Figs. 4, 5. There are no significant difference between NN, INN and KAN.

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 comparison with 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. 4). 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 (middle and bottom Figs. 4) 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. We compared three resizing modes: bicubic, bilinear and nearest. None of them demonstrates the best performance in all environments; nevertheless, the bicubic mode operates relatively more stably, and we propose to stick with it.

To illustrate the difference between INN and conventional NN, we can look at the weight matrix on the second dense layer in Fig. 6 for NN (left) and INN (right) trained on 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 as was supposed. This behavior indicates that for this imitation learning task, the size of this layer is excessive and can be reduced.

The Fig. 7 shows the mean absolute values of the one-dimensional Fourier transforms (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.

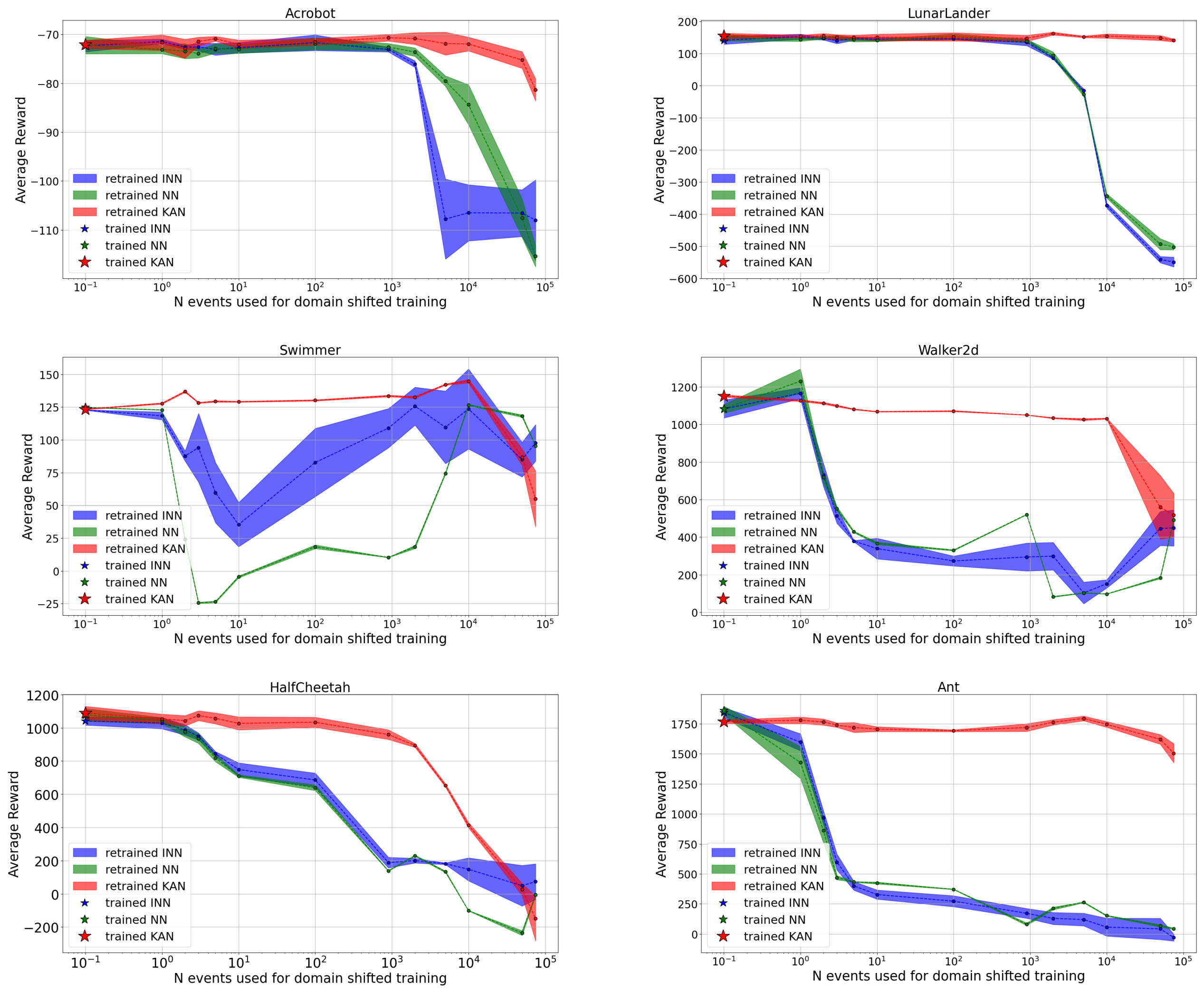


Fig. 5. AR modification during training on data that has undergone domain shift in the following environments: Acrobot (top left), Lunar Lander (top right), Swimmer (middle left), Walker2d (middle right), Half Cheetah (bottom left), Ant (bottom right).

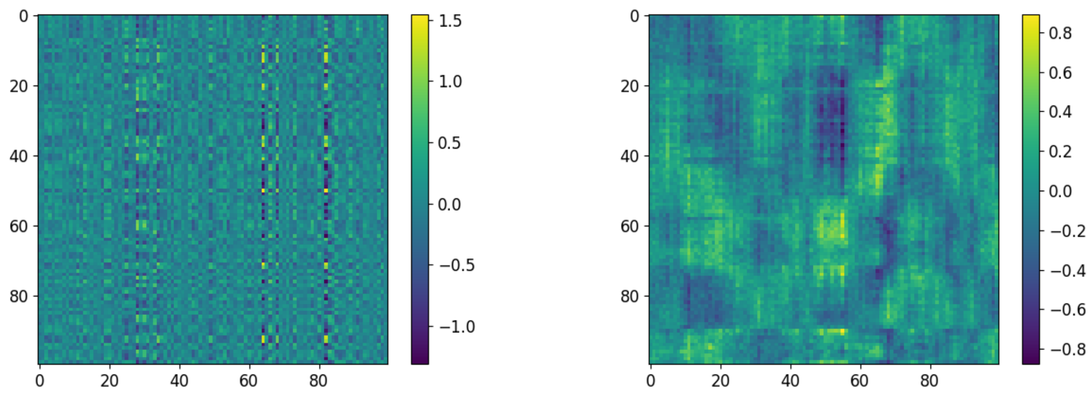


Fig. 6. 2D weight matrix 100×100 of second layer for NN (left) and INN (right) used in Lunar Lander problem.

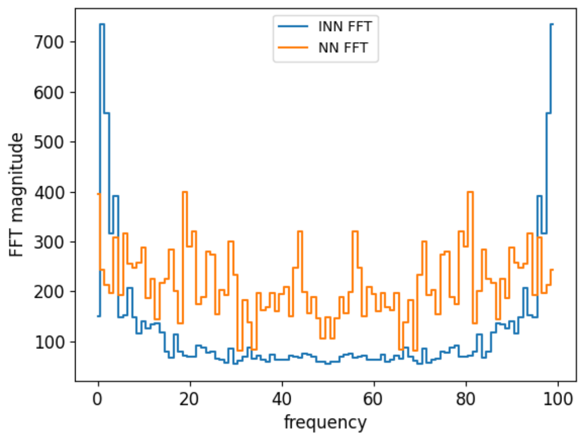


Fig. 7. Mean absolute values of 1D FFT of rows of 2D weight matrix of second layer

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 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.

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.

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 (23), (24). 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 (25), which offers commercial solutions in Neuromorphic Analog Signal Processing. Consequently, INN are closely aligned with the algorithms of future technologies.

In the Fig. 5 we have demonstrated for the first time that KAN-based agent policies are much less prone to catastrophic forgetting in comparison to NN and INN models. This is because learning in a new environment primarily changes the behavior of activations rather than weights, which allows maintaining functionality in the old environment. In all tested environments, the AR drop begins much later in the case of KAN. The catastrophic forgetting for KAN starts to manifest itself after about a thousand iterations of weight updates on the shifted dataset. This interesting property of KAN will be further investigated in future projects. In Supplementary Note 3, we demonstrate the output values of 100 neurons in the 2nd layer for 50 observation states. KAN shows the presence of large outliers after activation, leading to the fact that learning on new data changes weights locally (with large amplitudes in activations).

Conclusion

This paper proposes a new approach to representing an agent's policy models as INN and KAN, which shows promise as a potentially effective strategy. By utilizing open RL Gym benchmarks, we have demonstrated that INN 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. On the orher hand we showed that KAN representation of the agent policy allows the agent to delay the onset of catastrophic forgetting to a later stage.

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. Eunhae Lee. The impact of model size on catastrophic forgetting in online continual learning, 2024.
5. Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks, 2024.
6. Diab W. Abueidda, Panos Pantidis, and Mostafa E. Mobasher. Deepokan: Deep operator network based on kolmogorov arnold networks for mechanics problems, 2024.

7. Qi Qiu, Tao Zhu, Helin Gong, Liming Chen, and Huansheng Ning. Relu-kan: New kolmogorov-arnold networks that only need matrix addition, dot multiplication, and relu, 2024.
8. Haihong Guo, Fengxin Li, Jiao Li, and Hongyan Liu. Kan v.s. mlp for offline reinforcement learning, 2024.
9. Yuliang Cai and Mohammad Rostami. Dynamic transformer architecture for continual learning of multimodal tasks, 2024.
10. 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.
11. Samin Yeasar Arnob, Riyasat Ohib, S. Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *ArXiv*, abs/2112.15579, 2021.
12. 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.
13. Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. Neural network distiller: A python package for dnn compression research. October 2019.
14. 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.
15. Yichen Zhu, Xiangyu Zhang, Tong Yang, and Jian Sun. Resizable neural networks, 2020.
16. A beginner-friendly introduction to kolmogorov arnold networks (kan). https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/. Accessed: 2024-10-30.
17. Solodskikh K. Kurbanov A. Torchintegral. https://github.com/TheStageAI/TorchIntegral, 2023. Accessed: 2024-11-08.
18. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
19. 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.
20. Half cheetah. https://gymnasium.farama.org/environments/mujoco/half_cheetah/. Accessed: 2024-11-16.
21. Akash Haridas, Karim Hamadeh, and Samarendra Chandan Bindu Dash. Dadagger: Disagreement-augmented dataset aggregation, 2023.
22. Torchvision.transforms. https://pytorch.org/vision/main/generated/torchvision.transforms.functional.resize.html. Accessed: 2024-11-05.
23. 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.
24. 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.
25. Polyn technology. https://polyn.ai/. Accessed: 2024-11-05.
26. John Schulman. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

Supplementary Note 1: 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 8, 9 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.

Supplementary Note 2: Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) (26) 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:

$$L(\theta_{\text{old}}, \theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\theta_{\text{old}}}}(s,a)\right] \tag{8}$$

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:

$$\overline{D}_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta) = \mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}}\left[D_{KL}(\pi_{\theta_{\text{old}}}||\pi_\theta)\right] \leq \delta \tag{9}$$

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 $\overline{D}_{KL}$:

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

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

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

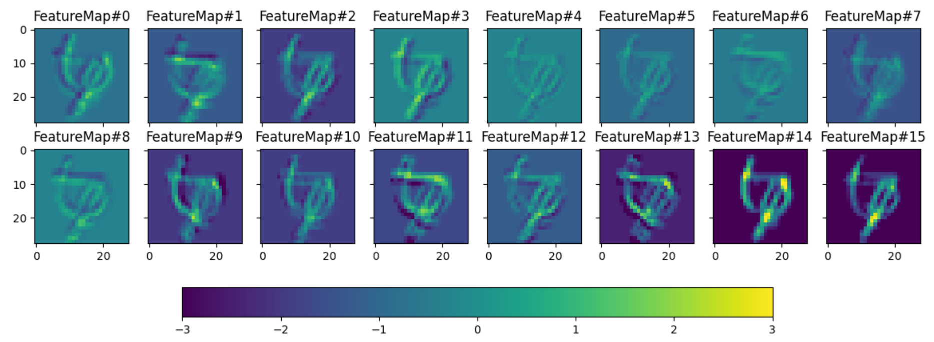
This approximate problem can be solved analytically by Lagrangian duality methods, which gives the solution for the weights vector:

$$\theta_{\text{new}} = \theta_{\text{old}} + \sqrt{\frac{2\delta}{g^{\text{T}} \cdot \text{H}^{-1} \cdot g}}\, \text{H}^{-1} \cdot g. \tag{10}$$

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

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha_j \sqrt{\frac{2\delta}{g^{\text{T}} \cdot \text{H}^{-1} \cdot g}}\, \text{H}^{-1} \cdot g, \tag{11}$$

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 (8). 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. We used 10 iterations for calculation of $\text{H}^{-1} \cdot g$ in (10) by using conjugate gradient method. The value of KL divergence was constrained by the maximum value $\delta = 0.01$.


(a)

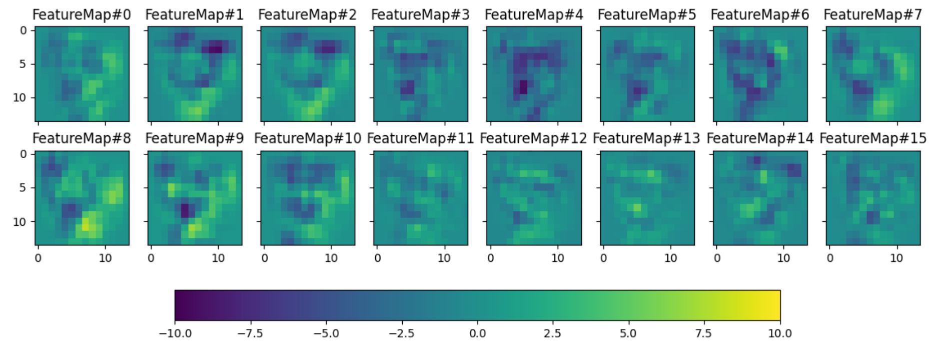
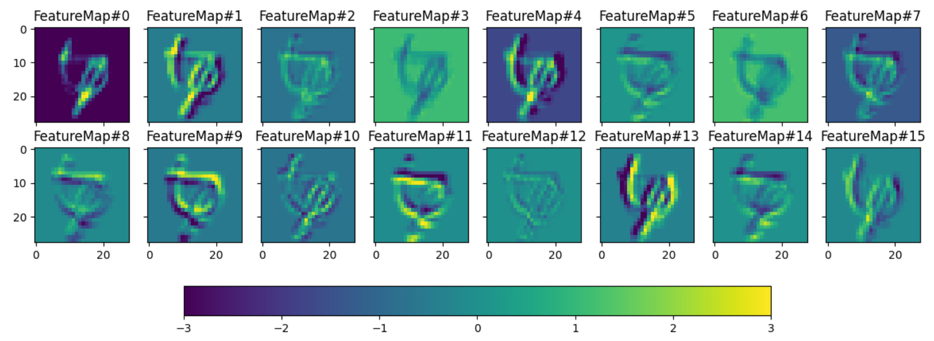

(b)

Fig. 8. Visualization of feature maps after the first (a) and second (b) convolutional layers in **Integral** *ModelMnist*


(a)

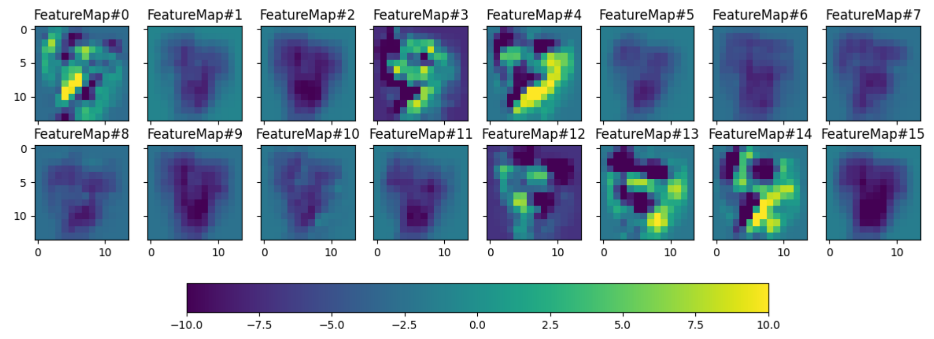

(b)

Fig. 9. Visualization of feature maps after the first (a) and second (b) convolutional layers in **conventional** *ModelMnist*

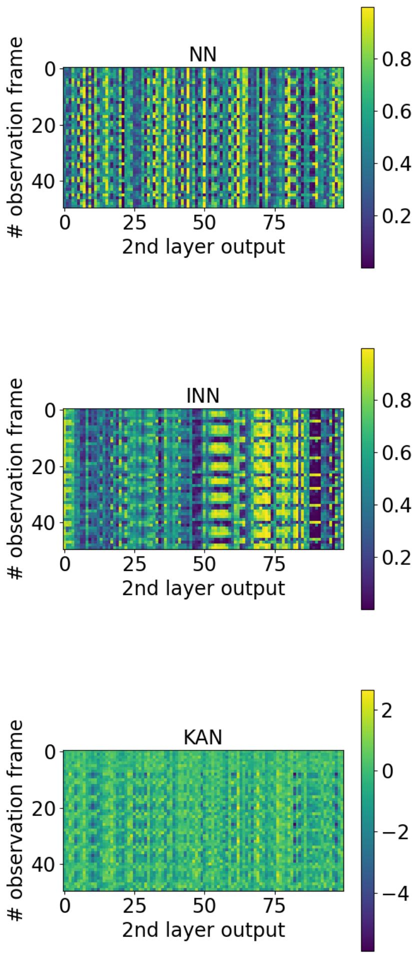


Fig. 10. Visualization of neuron outputs on the second layer of the networks used for agent in Half Cheetah environment. The horizontal axis represents the neuron number on the second layer. The vertical axis indicates observation number.

Supplementary Note 3: Intermediate layer currents.

The Fig. 10 shows the visualization of neuron outputs on the second layer of the networks used. The horizontal axis represents the number of neurons on this layer, while the vertical axis indicates the observation number. We plotted this spectrum for 50 observations in the Half Cheetah environment.

It is evident that the activations of INN and NN are distributed randomly within the range from 0 to 1. In contrast, the potential of KAN neurons after activation on the third layer is distributed much more broadly. For each observation, there are neurons with very large absolute values.

The key difference between INN, NN, and KAN is that backpropagation in KAN is focused on a certain part of neurons with large activation values, which is not the case with INN and NN.