RELATIONAL MACROSTATE THEORY GUIDES ARTIFICIAL INTELLIGENCE TO LEARN MACRO AND DESIGN MICRO

Yanbo Zhang
School of Earth and Space Exploration
Arizona State University
Tempe, AZ 85281, USA
Zhang.Yanbo@asu.edu

Sara Imari Walker
School of Earth and Space Exploration
Arizona State University
Tempe, AZ 85281, USA
sara.i.walker@asu.edu

ABSTRACT

A central focus of science is the identification and application of laws, which are often represented as macrostates that capture invariant properties associated with symmetries. However, complex systems can be challenging to study due to their high-dimensionality, non-linearity, and emergent properties. To address this challenge, we propose the relational macrostate theory (RMT) that defines macrostates in terms of symmetries between mutually predictive observations. Additionally, we have developed a machine learning architecture, MacroNet, that can learn these macrostates and invertibly sample from them, allowing for the design of new microstates consistent with conserved properties. By utilizing this framework, we have studied how macrostates can be identified in systems ranging from simple harmonic oscillators to complex spatial patterns known as Turing instabilities. Our results demonstrate how emergent properties can be designed by identifying the unbroken symmetries that give rise to invariants, bypassing Anderson's "more is different" by showing that "more is the same" in complex systems.

1 INTRODUCTION

Finding laws and using laws is one of the central topics in science. In physics, laws can be represented as invariants like energy, describing what is possible and what is not. Symmetry is a powerful tool for finding such invariants. Noether's theorem (Noether, 1971) states that for systems with conservative forces, every differentiable symmetry corresponds to a macro level conservation law. However, for more general cases, macrostates that link sub-spaces of microstates are required to fill the gap left by Noether's theorem in non-differentiable mappings: for example, rule-behavior, genotype-phenotype, and text-image mappings require a more general method for identifying invariants. In the past, successful laws like Newton's laws of motion worked well because they identified macroscopic properties like mass, reducing the motion of high-dimensional objects to a single measurable scalar quantity. However, finding macrostates for complex systems like biological and technological systems has proven challenging due to their high dimensionality, nonlinear behavior, and emergent properties. This suggests that machine learning may help identify conservation laws, even in non-differentiable systems, by identifying macrostates and the symmetries they conserve.

In the current work, we introduce a relational theory of macrostates, which provides a framework for training artificial neural networks, a complex system themselves, to break the barrier of complexity (Jumper et al., 2021; Pathak et al., 2018; Seif et al., 2021) and to identify macrostates based on symmetries in complex systems. Existing contrastive machine learning methods (Chen & He, 2021; Oord et al., 2018; Mikolov et al., 2013) have applied similar ideas to find lower-dimensional representations for microstates by relations. However, these methods either require large numbers of negative samples or cannot design microstates. Recent work has also focused on creating artificial intelligence that can learn natural laws from data with minimal supervision (Udrescu & Tegmark, 2020; Liu & Tegmark, 2021; Daniels & Nemenman, 2015), but this does not go beyond extracting laws from data. To take the next step, requires artificial intelligence that can learn rules from data and use that knowledge to design new systems that follow those rules. Our machine learning architecture, MacroNet, learns macrostates and designs microstates by using the macrostate theory on general relations and introducing invertibility.

2 Relational Macrostate Theory

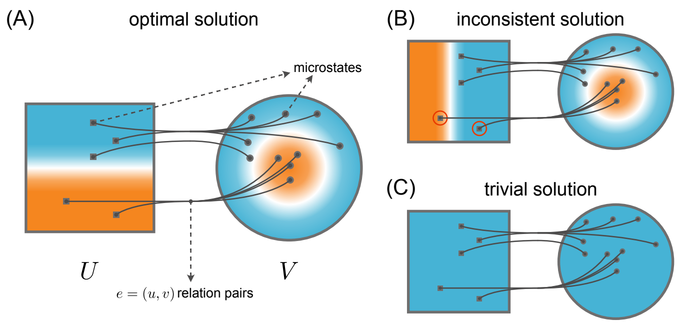


Figure 1: The rectangle and disks represent the space of microstates. The points and links represent the observed microstate pairs (u_i, v_i). The background color illustrates the macrostates. **(A)** shows an optimal solution. **(B)** shows an inconsistent solution. **(C)** shows a trivial but legal solution in which all microstates are coarse-grained to the same macrostate.

There have been various attempts to identify macrostates related to the emergence of regularities in complex systems (Chen et al., 2021; Kipf et al., 2019; Sun et al., 2021), including the causal state theory (Shalizi & Moore, 2003) and causal emergence theory (Hoel, 2017; Chvykov & Hoel, 2020; Comolatti & Hoel, 2022). However, these theories have their limitations. The causal state theory does not take into account microstate similarity, and both theories define macrostates based only on temporal relations. However, not all regularities that we may want to associate with laws involve time. A general theory of macrostates should be defined based on general relations between observations that retain symmetries under mappings, whether they are temporal, genotype-phenotype, word co-occurrence (Mikolov et al., 2013), or other types of mappings. The theory of macrostates we propose is general enough to extend to the symmetry of underlying rules, allowing us to identify sets of rules that produce a specific large-scale behavior (Anderson, 1972).

Mathematically, a macrostate is an ensemble of microstates that belong to the same equivalence class under a mapping φ. If two microstates u and u' have the same behavior (macrostate) under this map, they are in the same equivalence class. We use symmetries to define macrostates based on the relations between microstates $u \in U$ and $v \in V$ as random variables. The *micro-to-micro relation* can be represented as a joint distribution $P(u, v)$, and the *micro-to-macro relation* can be represented as the joint distributions $P(\alpha, v)$ and $P(u, \beta)$, where α and β are macrostates mapped from u and v by φ_u and φ_v. The micro-to-macro relation for a given microstate u_i (or v_i) can be represented as the conditional distribution $P(\beta|u_i)$ (or $P(\alpha|v_i)$). In the most general case, macrostates are defined as:

Definition 1. Two pairs of microstates u_i and u_j (and v_i and v_j) belong to the same macrostate if and only if they have the same micro-to-macro relation:

$$u_i \sim u_j \Leftrightarrow P\left(\beta|u_i\right) = P\left(\beta|u_j\right) \text{ and} \tag{1}$$

$$v_i \sim v_j \Leftrightarrow P\left(\alpha|v_i\right) = P\left(\alpha|v_j\right) \tag{2}$$

This definition creates an equivalence class of symmetries where $u_i \sim u_j$ and $v_i \sim v_j$ (where $\sim$ indicates "is equivalent to" under a symmetry operation). Thus, as in Noether's theorem, we see that the definition of a macrostate entails simultaneously defining a class of symmetry operations, although here our definition is sufficiently general that the system of interest need not necessarily be continuously differentiable. We can approach the definition of macrostates by solving the equation $\varphi_u(u) = \varphi_v(v)$ (see SI A.1.3). This will be included in the loss function for MacroNet's machine learning task. Figure 1C shows an example of trivial solutions, which reminds us to add additional requirements to avoid this. To specify "good" macrostates, we maximize mutual information $I(\varphi(v); \varphi(u))$ at the macroscopic level for a given dimension of macrostates, where (u, v) is sampled from $P(u, v)$.

3 MacroNet Architecture

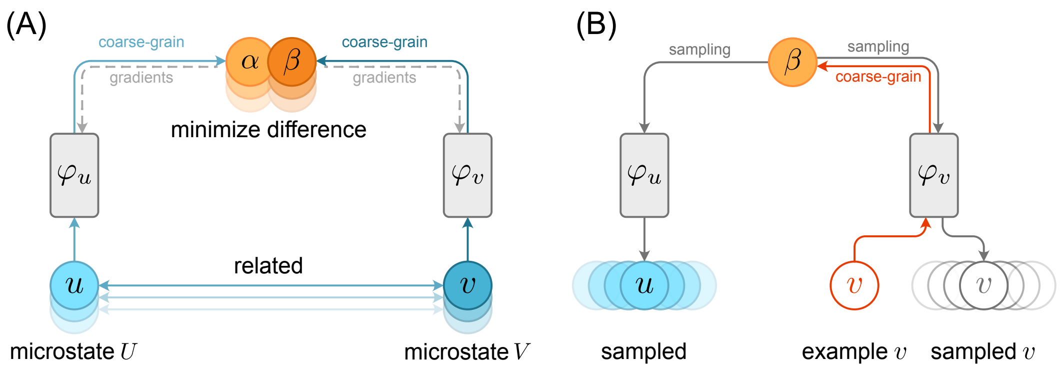


Figure 2: MacroNet neural network architecture. **(A)** Training process using two invertible neural networks to map microstates to the same macrostates. **(B)** Conditional sampling and designing process using a microstate from type V to compute its macrostate and sample microstates in U or V with that macrostate.

To find optimal macrostates, we propose a self-supervised generative model for finding macrostates from observations (Figure 2A). We optimize macrostates to predict other macrostates using neural networks φ_u and φ_v for coarse graining on U and V. The prediction loss is $\mathcal{L}_P = \mathbb{E}_{(u,v)\sim P(u,v)} |\varphi_u(u) - \varphi_v(v)|^2$, where (u, v) are pairs of microstates sampled from the training data. To avoid trivial solutions, we add a distribution loss $\mathcal{L}_D = \mathcal{L}_{D_u} + \mathcal{L}_{D_v}$ to force the output following independent normal distribution, where $\mathcal{L}_{D_u}$ and $\mathcal{L}_{D_v}$ are exactly the training loss in invertible neural networks (Dinh et al., 2014; 2016; Kingma & Dhariwal, 2018). We train the neural networks by combining the prediction loss $\mathcal{L}_P$ with the distribution loss $\mathcal{L}_D$ using a hyperparameter γ to balance the two terms, so the final loss is $\mathcal{L} = \mathcal{L}_P + \gamma\mathcal{L}_D$. This combination allows us to approach the mutual information criterion. We use invertible neural networks to compute the log-determinate of the Jacobian and do sampling efficiently. Given an example microstate v', we can use φ to compute its macrostate β and then invert the neural network to sample microstates v_s (or u_s) with the same macrostate as v'. This enables the design of complex systems without human classification of behavior (Figure 2B).

4 Results

We applied our macrostate theory on Turing patterns using the Gray-Scott model (Gray & Scott, 1984), a reaction diffusion model with four parameters (D_a, D_b, F, k) that can generate complex patterns (see Figure 6A). The model was originally designed to study the pattern formation in biology. By finding macrostates mapping patterns to parameters, we can design related systems by specifying parameters that will yield user-specified patterns. Here, u is the parameter vector. And v is the generated pattern, represented by 64×64 two-channel images.

The trained neural network maps parameters and patterns to each other at the macro level. By giving an example pattern v (Figure 3A), we can sample parameters $u' = \varphi_u^{-1}(\varphi_v(v), z)$ with the same macrostate as v (Figure 3B). The sampled rules (set of four parameters) will generate similar patterns as the example patterns (Figure 3C). This shows that MacroNet can design complex systems by sampling parameters that generate patterns with the same macrostate as the example behavior. MacroNet can also discover microstate ensembles associated with macrostates. Figure 3B shows the distribution of parameters sampled from different macrostates. The sample points with the same color are considered equivalent under the mapping φ, which takes the microstate to a macrostate. Parameters in the same equivalence class (sharing the same symmetry) will lead to patterns with the

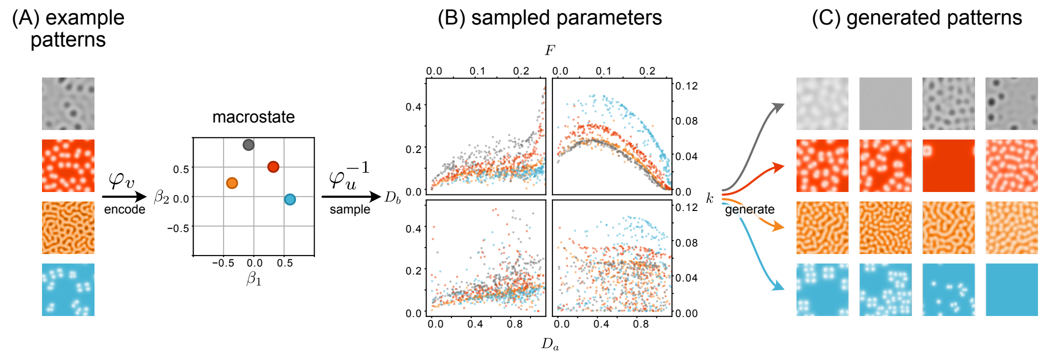


Figure 3: Experiments on Turing patterns. **(A)** By giving an example pattern, we can compute its macrostate by φ_v. The patterns are colorized for distinguishing different experiments. **(B)** Sampled ensembles of corresponding parameters from the macrostates. The points with the sample color are sampled from the same macrostate. **(C)** Using the sampled parameters, we can generate Turing patterns with similar macroscopic shape as the corresponding examples.

same macrostates, so we can sample any parameters along these equivalence curves and generate Turing patterns with the user-specified behavior.

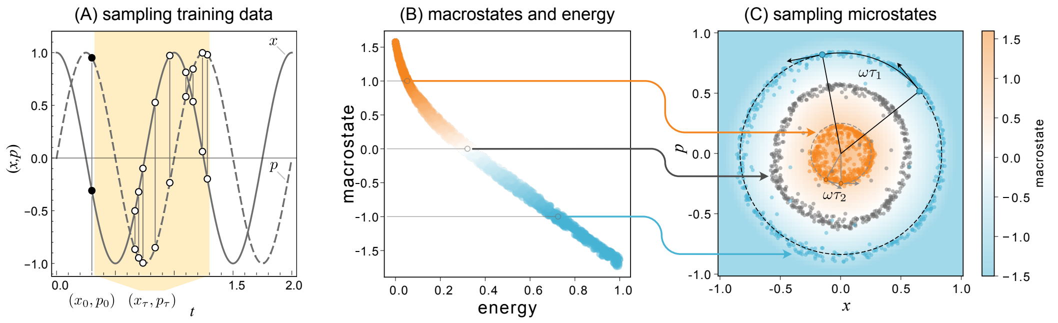


Figure 4: We train a neural network to find invariant quantities in simple harmonic oscillators. **(A)** the (u, v) pairs are sampled from simulations, with a time interval τ sampled from a uniform distribution. **(B)** the neural network learns energy as the invariant quantity. **(C)** conditional sampling microstates from the given macrostates.

We then apply MacroNet so simple harmonic oscillators (SHOs) to identify the symmetry of time translation invariance (energy). We represent the micro-to-micro relation as pairs of past-future position and momentum. The time interval is uniformly sampled (see Figure 5A), so it is impossible for accurate prediction at micro-level. Two neural networks φ_u and φ_v share the same weights to find time invariants. The resulting macrostate is a function of energy (Figure 4B), which indicates we find energy by finding time invariants. Sampling from macrostates (Figure 4C) allows for predicting future macrostates and generating corresponding microstates, even when microscale predictions are impossible. This demonstrates how machine learning combined with a theory for macrostates can find physical concepts from observations, and determine predictable variables in stochastic relations.

5 Discussion

Since Anderson published the seminal paper, *More is Different,* it has been increasingly recognized that complex systems displaying emergent behaviors do not necessarily share the same symmetries as their micro-rules (Strogatz et al., 2022). However, some symmetries may still be present at both the micro and macro levels. Indeed, this is what we see in the experiments presented in this work. Each macrovariable can represent a type of symmetry: for instance, the energy of a simple harmonic oscillator represents the time translation symmetry. This is a general framework for identifying macrostates as maps conserving the symmetries of systems. Hence, given that "more is different" is true in most cases, we can still find examples of macrovariables that behave as "more is the same."

REFERENCES

Philip W Anderson. More is different: broken symmetry and the nature of the hierarchical structure of science. *Science*, 177(4047):393–396, 1972.

Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2019.

Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *arXiv preprint arXiv:2103.04922*, 2021.

Boyuan Chen, Kuang Huang, Sunand Raghupathi, Ishaan Chandratreya, Qiang Du, and Hod Lipson. Discovering state variables hidden in experimental data. *arXiv preprint arXiv:2112.10755*, 2021.

Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 32, 2019.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15750–15758, 2021.

Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

Pavel Chvykov and Erik Hoel. Causal geometry. *Entropy*, 23(1):24, 2020.

Renzo Comolatti and Erik Hoel. Causal emergence is widespread across measures of causation. *arXiv preprint arXiv:2202.01854*, 2022.

Bryan C Daniels and Ilya Nemenman. Automated adaptive inference of phenomenological dynamical models. *Nature communications*, 6(1):1–8, 2015.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

DC Dowson and BV666017 Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

Timothy Dozat. Incorporating nesterov momentum into adam. *ICLR 2016 workshop submission*, 2016.

Peter Gray and Stephen K Scott. Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Oscillations and instabilities in the system a+ 2b$\rightarrow$ 3b; b$\rightarrow$ c. *Chemical Engineering Science*, 39(6):1087–1097, 1984.

Anne Greenbaum and Tim P Chartier. *Numerical methods: design, analysis, and computer implementation of algorithms*. Princeton University Press, 2012.

Erik P Hoel. When the map is better than the territory. *Entropy*, 19(5):188, 2017.

Hong-Ye Hu, Dian Wu, Yi-Zhuang You, Bruno Olshausen, and Yubei Chen. Rg-flow: a hierarchical and explainable flow model based on renormalization group and sparse prior. *Machine Learning: Science and Technology*, 3(3):035009, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Thomas Kipf, Elise Van der Pol, and Max Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

A Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

Ziming Liu and Max Tegmark. Machine learning conservation laws from trajectories. *Physical Review Letters*, 126(18):180604, 2021.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Emmy Noether. Invariant variation problems. *Transport theory and statistical physics*, 1(3):186–207, 1971.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical review letters*, 120(2):024102, 2018.

John E Pearson. Complex patterns in a simple system. *Science*, 261(5118):189–192, 1993.

Alireza Seif, Mohammad Hafezi, and Christopher Jarzynski. Machine learning the thermodynamic arrow of time. *Nature Physics*, 17(1):105–113, 2021.

Cosma Rohilla Shalizi and Cristopher Moore. What is a macrostate? subjective observations and objective dynamics. *arXiv preprint cond-mat/0303625*, 2003.

Steven Strogatz, Sara Walker, Julia M Yeomans, Corina Tarnita, Elsa Arcaute, Manlio De Domenico, Oriol Artime, and Kwang-Il Goh. Fifty years of 'more is different'. *Nature Reviews Physics*, 4 (8):508–510, 2022.

Yu Sun, Gaoke Hu, Yongwen Zhang, Bo Lu, Zhenghui Lu, Jingfang Fan, Xiaoteng Li, Qimin Deng, and Xiaosong Chen. Eigen microstates and their evolutions in complex systems. *Communications in Theoretical Physics*, 73(6):065603, 2021.

Alan Mathison Turing. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52 (1):153–197, 1990.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

A APPENDIX

A.1 DEFINITIONS AND THEORY

A.1.1 DEFINITIONS

Equivalence

Two microstates are equivalent if and only if they belong to the same macrostate. Using $\sim$ to represent equivalence, we have $u \sim u' \iff \varphi(u) = \varphi(u')$. Here φ maps microstates to macrostates.

Relations

We use the inclusive term relation to include most types of paired variables – for instance, co-occurrence pairs, data-label pairs, or past-future pairs, etc. A set of microstate pairs (u_i, v_i) can be mathematically represented by joint distribution $P(u, v)$. This joint distribution represents the entire *micro-to-micro* relations. Given a microstate u_i, we can define its micro-to-micro relation as a conditional distribution $P(v|u = u_i)$ or $P(v|u_i)$.

Since there are two types of data in the paired datasets, we use $\alpha = \varphi_u(u_i)$ and $\beta = \varphi_v(v_i)$ to represent the macrostates of u_i and v_i respectively. For simplicity, we also use φ to represent either of the mappings from microstates to macrostates when there is no ambiguity.

Given the microstates and their macrostates, we can define the entire *micro-to-macro* relation as $P(u, \beta)$ and $P(\alpha, v)$. And the micro-to-macro relation for a certain microstate, say u_i (or v_j), is represented as conditional distributions:

$$P(\beta|u_i) = \int P(\beta|v)P(v|u_i)\mathrm{d}v \tag{3}$$

$$P(\alpha|v_i) = \int P(\alpha|u)P(u|v_i)\mathrm{d}u \tag{4}$$

Here, the $P(\beta|v)$ is a probabilistic representation of φ_v, which is a many-to-one mapping since φ_v is a deterministic mapping.

The *macro-to-macro* relation can also be represented as the distribution $P(\alpha, \beta)$.

$$P(\alpha, \beta) = \iint P(\alpha|u)P(\beta|v)P(u, v)\mathrm{d}v\mathrm{d}u \tag{5}$$

So, we can also define macro-to-macro for certain macrostates as conditional distributions $P(\beta|\alpha_i)$ and $P(\alpha|\beta_i)$. These definitions of relations are illustrated in Figure A1.

(A) entire relations

$$\begin{tikzcd}
\alpha \arrow[r, no head, "P(\alpha,\beta)"] & \beta \\
u \arrow[u, "\varphi_u"] \arrow[ur, no head, "P(u,\beta)"] \arrow[r, no head, "P(u,v)"] & v \arrow[u, "\varphi_v"']
\end{tikzcd}$$

(B) relations for certain states

$$\begin{tikzcd}
\alpha_i \arrow[r, "P(\beta|\alpha_i)"] & \beta \\
u_i \arrow[u, "\varphi_u"] \arrow[ur, "P(\beta|u_i)"] \arrow[r, "P(v|u_i)"] & v \arrow[u, "\varphi_v"']
\end{tikzcd}$$

Figure A1: The relations can be represented by joint distributions and conditional distributions. (A) Joint distributions are used to represent the entire relationship. (B) For a certain microstate u_i or macrostate α_i, we can use conditional distributions to represent its relations.

A.1.2 DEFINITION OF MACROSTATES

Based on the definitions of relations, we can define macrostates based on micro-to-macro relations:

Definition S1: macrostate. Two pairs of microstates u_i and u_j (and v_i and v_j) belong to the same macrostate if and only if they have the same micro-to-macro relation:

$$u_i \sim u_j \iff P(\beta|u_i) = P(\beta|u_j) \text{ and} \tag{6}$$

$$v_i \sim v_j \iff P(\alpha|v_i) = P(\alpha|v_j) \tag{7}$$

The macrostate solutions should be self-consistent. Figure 1A shows a consistent solution, as an example, $u_1 \sim u_2$ because $P(\beta|u_1) = P(\beta|u_2)$, and $v_1 \sim v_2$ because $P(\alpha|v_1) = P(\alpha|v_2)$. However, Figure 1B shows an inconsistent solution: The microstates in red circles are all mapped to the orange macrostate in V and therefore should not be mapped to different macrostates in U because each microstate should belong to only one macrostate. they have the same micro-to-macro relation.

Another solution is that all the microstates are mapped to the same macrostate (see Figure 1C). This kind of solution will not provide any meaningful information about the systems under study.

Therefore, in addition to the definition S1, we propose an information criterion to specify "good macrostates". That is, given a certain number (or dimension) of macrostates, the information criterion imposes the constraint of maximizing the mutual information $I(\alpha;\beta)$ at the macrostate.

A.1.3 FROM DEFINITION TO OPTIMIZATION OBJECTIVE

In relational macrostate theory, macrostates are defined in terms of relations. As such they are defined in a circular manner: the macrostate of a microstate is determined by the macrostates of its related microstates. Since the macrostates in U are defined by the macrostates in V, and macrostates in V are defined by U, we need to optimize the mapping from micro-to-macro to find informative and consistent solutions that allow identifying macrostates associated to symmetries. Based on our definition of macrostates, we can do continuation on the definition by introducing distance functions D_1 and D_2. The continuous version of the definition becomes:

$$D_1[\varphi_u(u_i), \varphi_u(u_j)] = D_2[P(\beta|u_i), P(\beta|u_j)], \tag{8}$$

$$D_1[\varphi_v(v_i), \varphi_v(v_j)] = D_2[P(\alpha|v_i), P(\alpha|v_j)]. \tag{9}$$

When these two equations are perfectly satisfied, this reduced to the original macrostate definition. When choosing D_1 be square Euclidean distance and D_2 be 2-Wasserstein distance (Dowson & Landau, 1982), we can verify the following formula is a solution for our macrostate definition:

$$\varphi_u(u_i) \approx \varphi_v(v_i). \tag{10}$$

More specifically, the solution is:

$$\varphi_u(u_i) - \varphi_v(v_i) \sim \mathcal{N}(0, \Sigma), \tag{11}$$

where (u_i, v_i) is sampled from $P(u, v)$, and tr$(\Sigma) \ll 1$. Using $P(\alpha|u_i)$ and $P(\beta|v_i)$ to represent $\varphi_u(u_i)$ and $\varphi_v(v_i)$ as distributions, we have:

$$P(\beta|u_i) = \int_v P(\beta|v)P(v|u_i)\mathrm{d}v \tag{12}$$

$$= \int_v P(\alpha + \delta|u_i)P(v|u_i)\mathrm{d}v \tag{13}$$

$$= P(\alpha + \delta|u_i) \tag{14}$$

where $\delta \sim \mathcal{N}(0, \Sigma)$ and $tr(\Sigma) \ll 1$. Here we replaced $P(\beta|u)$ by $P(\alpha + \delta|u_i)$ because $\varphi_u(u_i) \approx \varphi_v(v_i)$, or $\alpha_i \approx \beta_i$. So, we can find that $P(\beta|u_i)$ and $P(\alpha|v_i)$ are both normal distributions with low standard deviations. For normal distributions X and Y, the 2-Wasserstein distance has a simple form:

$$W_2(X, Y)^2 = |\mu_x - \mu_y|^2 + \text{tr}\left(\Sigma_x + \Sigma_y - 2(\Sigma_x \Sigma_y)^{1/2}\right), \tag{15}$$

So, the definition becomes:

$$|\varphi_u(u_i) - \varphi_u(u_j)|^2 = |\mathbb{E}(\varphi_v(v_i) - \varphi_v(v_j))|^2 + \text{tr}(\Sigma_i + \Sigma_j - 2(\Sigma_i \Sigma_j)^{1/2}), \tag{16}$$
$$|\varphi_v(v_i) - \varphi_v(v_j)|^2 = |\mathbb{E}(\varphi_u(u_i) - \varphi_u(u_j))|^2 + \text{tr}(\Sigma_i' + \Sigma_j' - 2({\Sigma_i}' \Sigma_j')^{1/2}), \tag{17}$$

Since $\Sigma \ll 1$, we can abandon the trace term and remove the expectations:

$$|\varphi_u(u_i) - \varphi_u(u_j)|^2 \approx |\varphi_v(v_i) - \varphi_v(v_j)|^2, \tag{18}$$
$$|\varphi_v(v_i) - \varphi_v(v_j)|^2 \approx |\varphi_u(u_i) - \varphi_u(u_j)|^2, \tag{19}$$

The formulas still hold when substitute $\varphi_u(u_i) \approx \varphi_v(v_j)$ into it. So, we can verify that $\varphi_u(u_i) \approx \varphi_v(v_j)$ is a solution for our definition. This solution can be approximated by minimizing the distance between $\varphi_u(u_i)$ and $\varphi_v(v_i)$. There may exist other more general but more complex solutions. However, this simple approach shows good performance in experiments.

A.2 Methods

A.2.1 Invertibility and distribution control

Technically, our framework requires two key features in the neural network for learning φ: the ability to perform conditional sampling and ability to control the distribution of its outputs. Fortunately, the invertible neural networks (INNs) cover both features. The invertibility makes conditional sampling possible. And the distribution control feature makes it possible to avoid trivial solutions without a large number of negative samples (in (Chen et al., 2020), 65536 negative samples are used).

In a broad definition, the INNs can be classified into two types: flow-based models (Dinh et al., 2014; 2016; Chen et al., 2019), and models that are trained to be invertible such as InfoGAN (Chen et al., 2016). The flow-based model, including the coupling models such as RealNVP (Dinh et al., 2016), NICE (Dinh et al., 2014), and ResNet-based models such as invertible residual networks (Behrmann et al., 2019) and ResFlow (Chen et al., 2019). The flow-based models have two common designs: first, they are guaranteed to be invertible, no matter how well they have been trained. Second, they are easy to compute determinants of Jacobians.

With the information of determinants of Jacobians, the probability density of the output can be computed by the “change of variable” theorem (Dinh et al., 2016), hence we can control the distribution of output. Here, for simplicity, let’s just consider an extreme case: if a linear matrix that maps a three-dimensional manifold to a zero, one, or two-dimensional manifold that is embedded in three-dimensional space. Then, the rank of the matrix must be two or lower. Hence, the determinant of Jacobian will be zero. So, by avoiding having zero determinants of Jacobians, we can avoid the dimension collapse, hence avoiding trivial solutions.

Another type of INNs is the models that are trained to be invertible. Such models should also have the two features as flow-based models: invertibility, and distribution control. InfoGAN (Chen et al., 2016) architecture is an example that follows the requirements. Compared to vanilla GANs, the InfoGAN is simply doing two different things: 1) splitting the input noise into two parts c and z. 2) add a Q network that can reconstruct the c information, i.e., $Q[G(c, z)] \to c$, where G is the generator. The inverse of InfoGAN is trained, it can *partially* inverse the process of $G : (c, z) \to x$ by using $Q : x \to c$, while the z information is lost. This loss will not affect our macrostate framework, because we can map microstates to macrostates by $Q : u \to \alpha$, and sample microstates from macrostates by $G : (\alpha, z) \to u$. The ability of distribution control is achieved by the reconstruction process and discriminator together. Given that discriminator exists, if c is sampled from a distribution P and $z \sim \mathcal{N}(0, 1)$, then $G(c, z)$ will follow the data distribution. Since Q is trained to predict c by the generated samples, as an inverse process, $Q(x \sim P_{data})$ will follow the distribution of P. By controlling the distribution, InfoGAN can also avoid trivial solutions.

Our experiments have all been trained on flow-based models. We are making this choice for three reasons: 1) flow-based models are guaranteed to be invertible. and 2) flow-based models are not likely to have mode collapse problems, while GAN based models often have such problems. This is critical if we want to design microstates. 3) flow-based models make the experiments more concise. However, the InfoGAN structure can still be useful when we need a high expressivity because it can use more different neural network structures.

A.2.2 INVERTIBLE NEURAL NETWORKS

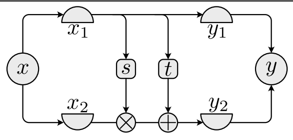
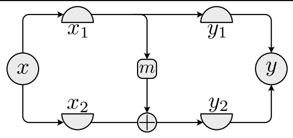
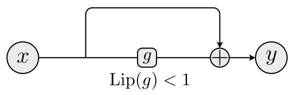
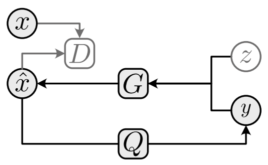
name	structure	forward	inverse
RealNVP	 x_1 y_1 x s t y x_2 y_2	$y_1 = x_1$ $y_2 = x_2 s(x_1) + t(x_1)$	$x_1 = y_1$ $x_2 = (y_2 - t(y_1))/s(y_1)$
NICE		$y_1 = x_1$ $y_2 = x_2 + t(x_1)$	$x_1 = y_1$ $x_2 = y_2 - t(y_1)$
ResFlow		$y = x + g(x)$	$x = \lim_{n\to\infty} x_n$ $x_n = y - g(x_{n-1})$ if Lip$(g) < 1$
InfoGAN		$y = Q(\hat{x})$	$\hat{x} \sim G(y, z)$

Table A1: illustrations for multiple versions of invertible neural networks (INNs).

Table A1 compares different types of INNs. The forward and inverse column shows the mapping from input x to output y, and y to x.

A.2.3 COARSE-GRAINING AND SAMPLING

The flow-based models require the output and input to have the same dimensions for invertibility. So, to do coarse-grain and up sampling, we need to adopt a special way to change dimensions.

(Hu et al., 2022) provided a multi-scale architecture, which let the network abandon dimensions: $f : x \to (y, z)$, where z is the abandoned dimensions, and y can be used to do supervised or self-supervised training. In this way, we can reduce the dimension and do coarse graining. In the forward process, given a N-dimensional input, the output will be splitted into two variables $\alpha^{(D)}$ and $z^{(N-D)}$, where the superscripts show their dimensions. Only α will be trained to satisfy $\varphi_u(u_i) = \varphi_v(v_i)$. To make it clear, we use φ to represent the mapping from u to α, and use Φ to represent the mapping from u to (α, z).

However, z is not totally ignored. Since we also want to do conditional sampling, the distribution of z should also be trained to be an independent normal distribution. So, the Jacobian of φ is computed by Φ so we can include z. When doing conditional sampling, given the macrostate $\alpha^{(D)}$ or $\beta^{(D)}$, we sample a $z^{(N-D)}$ to compute $\Phi^{-1}(\alpha, z)$. The coarse-graining and sampling process are summarized in Table A2.

	forward	**training**
coarse-graining	$\Phi(u^{(N)}) = (\alpha^{(D)}, z^{(N-D)})$, where $z^{(N-D)}$ is the abandoned dimensions. And $\varphi(u) = \alpha$.	both α and z will be trained to follow independent normal distribution. z will *not* be stored since we know its distribution after training. And α will be trained as a macrostate.
sampling	$\varphi^{-1}(\alpha) = \Phi^{-1}(\alpha^{(D)}, z^{(N-D)}) = u^{(N)}$, where $z^{(N-D)}$ is sampled from an independent normal distribution.	—

Table A2: details of coarse-graining and sampling process.

Since (α, z) is trained to be independent normal distributions, the $P(z|\alpha)$ should also follow normal distribution. With this feature, we are able to do conditional sampling of u from $P(u|\varphi(u) = \alpha)$.

A.2.4 Training tricks

The flow-based models have limitations of expressivity (Bond-Taylor et al., 2021) since their Jacobian and dimensions are restricted. A common way to overcome this problem is to have more layers of INNs, for example, the Glow model (Kingma & Dhariwal, 2018) uses nearly one hundred layers to do generative tasks on the CIFAR10 dataset (Krizhevsky, 2009). However, for some tasks which have very low dimensions, more layers cannot provide results that are good enough. To solve this problem, we propose two useful tricks for different situations.

Noisy kernel trick

The expressivity problem can often be overcome by adding more layers of INNs (Bond-Taylor et al., 2021). However, our experiments show that when the input dimension is too low, adding layers will not help. While extending neural networks wider can significantly improve the performance. To extend the neural network of INN, we need to extend the input dimension by concatenating the original input with additional random variables:

$$u' = [u, x], x \sim N^d(0, 10^{-3}) \tag{20}$$

With this method, we can add d dimensions to the inputs. Here, the u is the original input, and x is the appended input, which is sampled from a standard normal distribution. Note that x has to be sampled from a d-dimensional distribution instead of zeros. This is because the flow-based model will be trained to map inputs to an independent normal distribution. However, if we append inputs by zeros, the input itself will be a lower dimensional manifold, which makes it impossible to be mapped to an independent normal distribution and leads to unstable training. We found that 10^{-3} is a good standard deviation that is small enough to reduce the interference from noise, and large enough to avoid the explosion of log-Jacobian. Since this method is increasing the input dimensions with gaussian noise, we call it "noisy kernel". The additional dimensions will increase the expressivity of flow-based models, which will lead to better performance. Table A3 shows noisy kernels can significantly improve the performance on the simple harmonic oscillator task.

However, the added noise will also have side effects on sampling. The additional dimensions in z will add noise to the output when doing sampling (see Table A3). So, we only suggest using noisy kernels when necessary, for example, when the dimension is too low.

One-side INN structures

In many cases, only one side of microstates needs to be sampled. In such a case, we only need to let one of two networks (i.e., φ_u or φ_v) be invertible. The other network is not necessary to be an

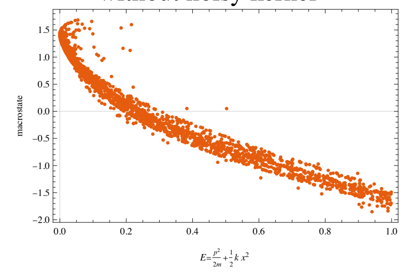
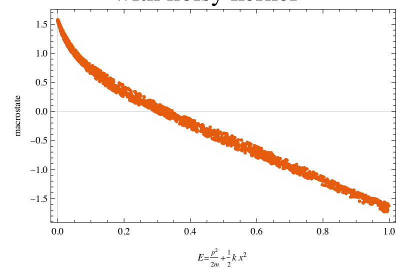
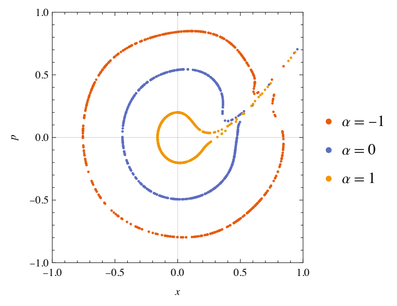
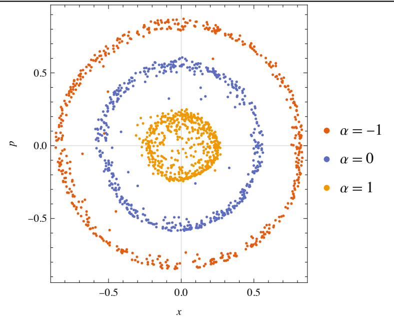
	without noisy kernel	with noisy kernel
A		
B		

Table A3: The noisy kernel can improve the performance when the input dimension is too low. However, noisy kernels may make the sampling noisy.

INN. This makes the optimization much easier since the free-form neural networks will have higher expressivity. We adopted this method in finding macrostates of Turing patterns.

Putting batch normalization at the last layer

The common practice in neural networks often puts the linear layer as the last layer. In MacroNet, although we have the distribution term to avoid trivial solutions, we still find that putting the invertible batch normalization layer (Dinh et al., 2016) as the last layer (or before the last resize layer) will improve the performance. This may be caused by the potential tradeoff between the prediction loss and the distribution loss, which could skew the distribution of macrostates away from gaussian distribution. This trick cannot omit the importance of the distribution loss. Even when the macrostates have a standard deviation of one, the macrostates can still be low-dimensional manifolds that lack information.

A.3 Experiments

A.3.1 Linear dynamical systems

A linear dynamical system can be represented as a differential equation (Strogatz, 2018):

$$\frac{\mathrm{d}\vec{x}}{\mathrm{d}t} = M\vec{x} \tag{21}$$

where M is a $n \times n$ matrix. n is the dimension of vector $\vec{x}$. So, when the system has different $\vec{x}$, the $\mathrm{d}\vec{x}/\mathrm{d}t$ will be different. Different matrices will lead to different behaviors, such as attractor, limit cycles, rotations or saddles (see Figure A2).

So, there exist many-to-many mappings between the matrix and trajectory:

1. one-to-many: For the same matrix M, depending on initial states, the trajectories can be different. For instance, given $M = I$, the trajectories can move to the right or left if the initial state $x_0 = (1, 0)$ or $(-1, 0)$.
2. many-to-one: Also, even with different matrices, the trajectories can be the same when the initial state is properly chosen. For instance, when $M_1 = I$, and M_2 be a permutation

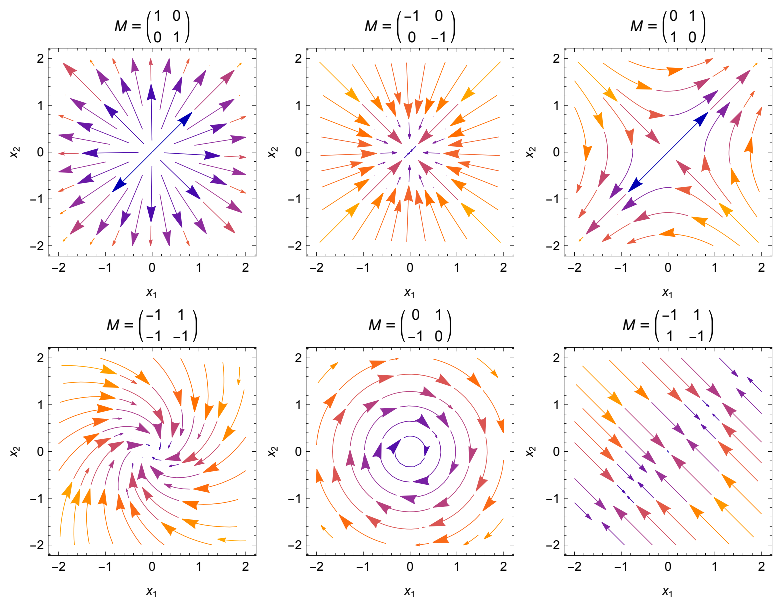


Figure A2: The behavior of the linear dynamical system is changing with the matrix M.

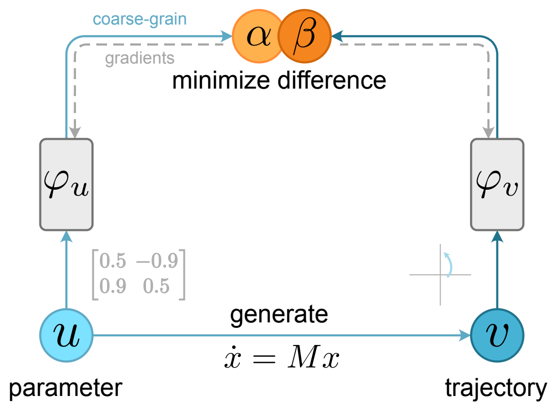


Figure A3: The training process of finding macrostates from linear dynamical systems. Both the parameters and trajectories are coarse-grained to two-dimensional macrostates.

matrix that permutes between dimension 1 and 2, their trajectories can be the same when the initial state $x_0 = (\xi, \xi)$, where $\xi > 0$.

For such many-to-many mapping situations, our macrostate theory and machine learning method can help us design the matrices for given trajectories. Here we define the macrostates on the parameter-trajectory pairs. The parameter is a 2×2 matrix M and the trajectory is a $n \times 2$ tensor $x_{0:n-1} = [x_0, x_1, ..., x_{n-1}]$ to represent the evolution with the initial state x_0, where $n = 8$. We coarse-grained both sides to a 2-dimensional space as the macrostate (see Figure A3).

The training data is generated by an algorithm. For each $(u = M, v = x_{0:n-1})$ pair, the M is firstly sampled from an independent normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$. Then the trajectory is generated by the dynamic $\mathrm{d}x/\mathrm{d}t = Mx$, where the initial state x_0 are sampled uniformly and independently in a 2-dimensional space $U^2(-1, 1)$.

The training takes 2000 epochs, and each epoch has 512 samples with a batch size of 256. We use Adam optimizer (Kingma & Ba, 2014) to train the model. The learning rate is 10^{-3} and the weight

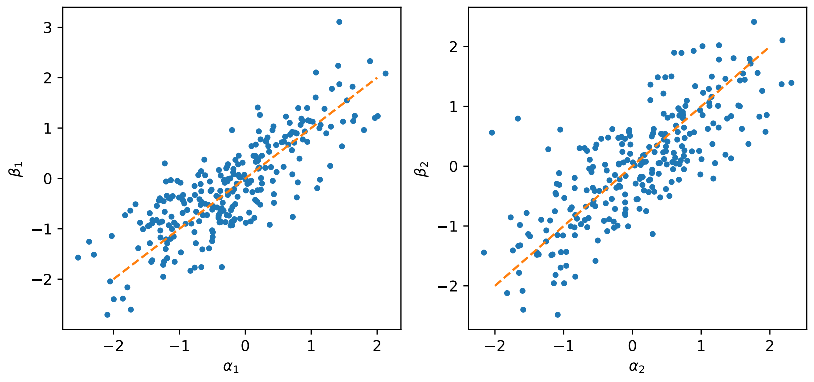


Figure A4: The trained neural network on linear dynamic systems is capable of predicting the macrostate. Here, each point represents a (α_i, β_i) pair. Here, $\alpha_i = \varphi_u(u_i)$ and $\beta_i = \varphi_v(v_i)$. By jointly compare α_i and β_j, we can quantify the performance of the predictions on macrostates. In the view of mutual information, when all points are on the curve $\beta = \alpha$, the mutual information $I(\alpha; \beta)$ is maxmized.

decay is 10^{-5}. We let $\gamma = 0.1$ to balance the prediction loss and distribution loss. Figure A4 shows the scatter plot of macrostates (α, β), which indicates the accuracy of prediction at macrostates.

After training, we can do two things: given a trajectory s_e as "example behavior", use φ_v^{-1} to sample other trajectories that have the same macrostate as s_e. Or, given a trajectory, use φ_u^{-1} to sample parameters that can generate this trajectory with certain initial states. Here we show the sampling with different example behaviors (represented by $x_{0:n-1}$, illustrated by red trajectories) in Figure A5.

Neural network architecture

The neural network maps the parameters and trajectories to a two-dimensional space as the macrostates. To improve the performance, we use noisy kernels to improve the performance. For the parameter side, we use a noisy kernel to increase the dimension from 4 to 8. For the trajectory side, we use a noisy kernel to increase the dimension from 16 to 32. The noises for each additional dimension are independently sampled from $\mathcal{N}(0, 10^{-3})$. The details of the structure of the neural networks are in Table A4.

A.3.2 Simple harmonic oscillator

There is an important special case of the macrostates. When the relation is built on temporally connected microstates, the neural network is predicting future macrostates, which is similar to the contrastive predictive learning (Oord et al., 2018), but adding the conditional sampling ability. Furthermore, if we force the two neural networks to share the same parameter, then it is learning time invariant quantities. Here we use simple harmonic oscillators (SHOs) as an example. The Hamiltonian of SHOs is:

$$H(x, p) = \frac{p^2}{2m} + \frac{1}{2}kx^2, \tag{22}$$

where $p = mv$ is the momentum, x is the position, m is the mass, and k represents the elasticity of the spring. In this experiment, we let $m = 1$ and $k = 1$ in all cases for simplicity. So, the solution is:

$$x_t = A\cos(t + \phi),\ p_t = -A\sin(t + \phi), \tag{23}$$

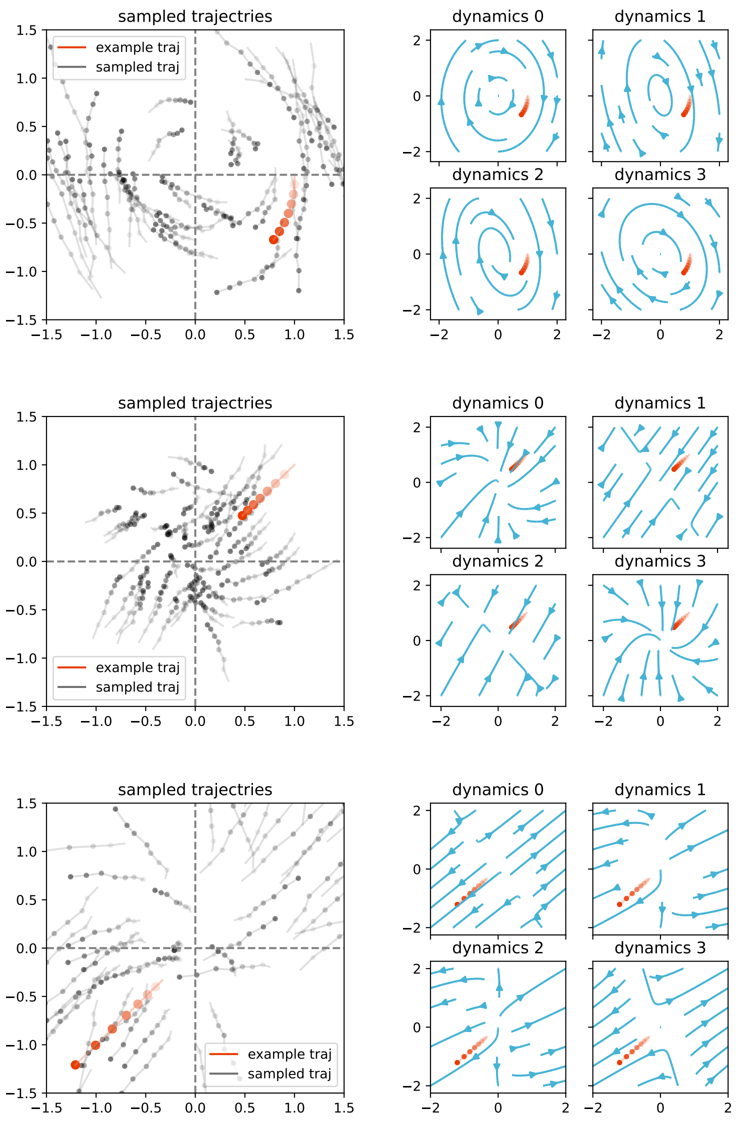


Figure A5: Red lines show the example trajectories. And gray dotted lines show the sampled microstates of trajectories. The blue vector lines represent the dynamics of sampled parameters.

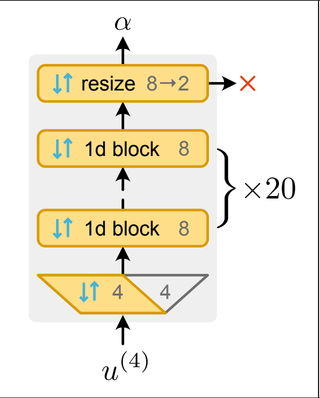
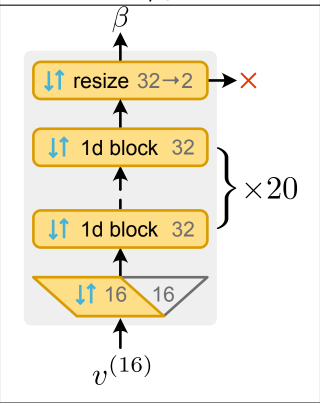
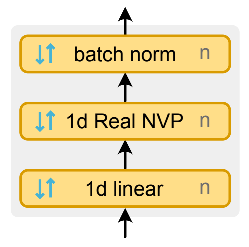
φ_u	φ_v	↓↑ 1d block n
 α ↓↑ resize 8→2 ↓↑ 1d block 8 ↓↑ 1d block 8 ×20 ↓↑ 4 4 $u^{(4)}$	 β ↓↑ resize 32→2 ↓↑ 1d block 32 ↓↑ 1d block 32 ×20 ↓↑ 16 16 $v^{(16)}$	

Table A4: Here we adopted the noisy kernels, represented by trapezoids. For φ_u, the 4-dimensional microstates input are increased to 8 dimensions by the noisy kernels. After that, there are 20 one-dimensional INN blocks (indicated by the ↓↑ icon). At the end, we simply abandon 6 dimensions to get a 2-dimensional output. The one-dimensional INN block is composed of a linear INN (Kingma & Dhariwal, 2018), a RealNVP 1-dimensional layer, and an invertible batch normalization layer (Dinh et al., 2016).

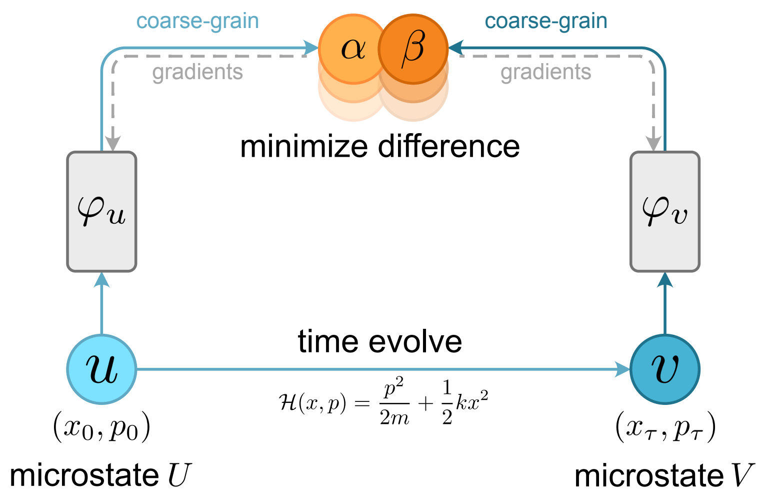


Figure A6: The training process of finding invariants as macrostates from simple harmonic oscillators.

where A depends on the initial energy, $A = \sqrt{x_0^2 + p_0^2}$. And ϕ is the initial phase, $\phi = \arctan(p_0/x_0)$. The microstate of simple harmonic oscillator is (x_t, p_t). To find an invariant quantity, we require the macrostate of $u = (x_0, p_0)$ should as close as the macrostate of $v = (x_\tau, p_\tau)$, where τ follows the uniform distribution $\mathcal{U}(0, 2\pi)$ (shown in Figure A7A). Since τ is a random variable, predicting microstate (x_τ, p_τ) is not possible. However, the macrostate can be predictable. The training architecture is shown in Figure A6.

We use 2048 samples of (u, v) pairs to train the neural network. The training takes 200 epochs with a batch size of 256. We use NAdam optimizer (Dozat, 2016) to optimize the neural network. The learning rate is 5×10^{-3}. The learning rate decreases by 0.1 in each 60 epochs. To balance the prediction loss and distribution loss, we choose $\gamma = 0.5$.

Figure A7B shows the invariant quantity found by our neural network has a clear and monotonous relation to the energy. We can also sample the microstates (x, p) from given invariant by implement

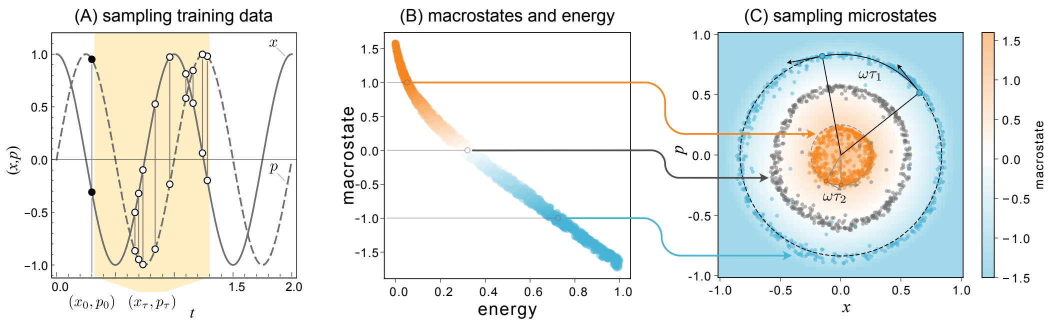


Figure A7: With a simple harmonic oscillator, we train a neural network to find invariant quantities as a special case of macrostates. **(A)** the (u, v) pairs are sampled from simulations, where $u = (x_0, p_0)$ (the black dots) and $v = (x_\tau, p_\tau)$. The τ is sampled from a uniform distribution $\mathcal{U}(0, 2\pi)$. The white dots in the yellow region show a sampling example of v. Due to the randomness of τ, it is impossible for accurate prediction at microstate. **(B)** the neural network learns energy as the invariant quantity. The x-axis is the energy of microstates computed by the physical theory of SHOs discovered by humans, and the y-axis is the macrostate discovered by the neural network. They show a monotonical relation, which implies the successful identification of energy by the neural network. **(C)** conditional sampling microstates from $P((x,p)|\varphi(x,p) = \alpha_i)$, where the α_i are the given macrostates. The results approximate equal energy surfaces, denoted by the dashed circles. Note that the noise in the sampling is a side effect of the noisy kernel trick we use here. The background color also shows the learned macrostate mapping as a field.

$\varphi^{-1}(\alpha)$. The results show that the neural network can sample a ring in (x, p) space (Figure A7C), which is exactly the solution of $p^2 + x^2 = H$.

Neural network architecture

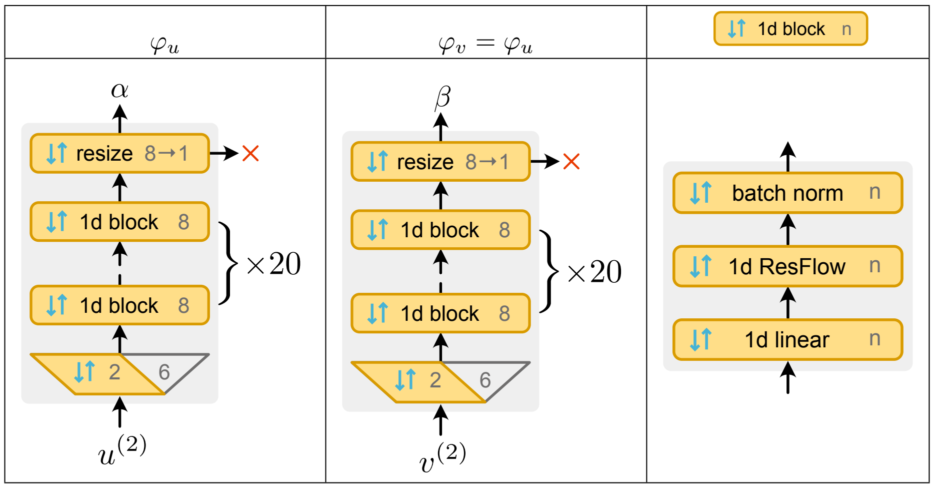


Table A5: The neural network structure for finding macrostates of simple harmonic oscillators. To find invariant quantity, the φ_u and φ_v have the same structure and share the same weights. We replaced the RealNVP layer with the ResFlow layer (Chen et al., 2019) to get better performance.

Since the dimension of the microstate is two, we use a noisy kernel to increase it to eight dimensions. The noise follows the distribution of $\mathcal{N}^6(0, 10^{-3})$. We also use residual flow (Chen et al., 2019) as the basic block to increase the expressivity. The details of the neural network are shown in Table A5. Note that here we let φ_v shares the same weight as φ_u.

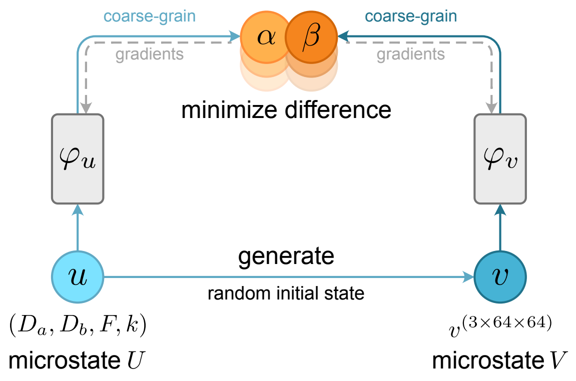


Figure A8: The training process of finding macrostates from Turing patterns. The neural networks maps the parameter (D_a, D_b, F, k) and patterns $v^{(3\times64\times64)}$ to macrostates in a two-dimensional space.

A.3.3 Turing patterns

The Turing patterns are two-dimensional patterns generated by reaction-diffusion models (Turing, 1990). By changing the parameter of the model, the reaction-diffusion model can generate many different types of patterns (Pearson, 1993). In this experiment, we use macrostate theory to find the macrostate of the patterns and parameters. Then, we sample parameters that can generate certain types of patterns.

Here we use the Gray-Scott model (Gray & Scott, 1984) as the reaction-diffusion model. In this model, there are two types of chemical components, their densities are represented as the density fields a and b. The dynamics is represented by the following differential equations:

$$\frac{\partial a}{\partial t} = D_a \nabla^2 a - ab^2 + F(1-a), \tag{24}$$

$$\frac{\partial b}{\partial t} = D_b \nabla^2 b + ab^2 - (F+k)b, \tag{25}$$

where D_a, D_b, F and k are four positive parameters that determine the behavior of the system. So, a microstate u here is a vector of the four parameters, i.e., $u = (D_a, D_b, F, k)$. And the microstate v is the pattern generated based on the parameters. When initializing the a, b as 64×64 grids, each elements are independently sampled from the uniform distribution $\mathcal{U}(0, 1)$. We approximate the differential equation on a $2 \times 64 \times 64$ tensor by using Euler method (Greenbaum & Chartier, 2012) with step size $\mathrm{d}t = 0.1$.

We only sample (u, v) pairs that have meaningful structure in the v matrix and omit the cases where v is a blank image (all elements in v have the same value) with no structure. Using this method, we sample 1024 pairs of microstates. The training architecture is shown in Figure A8.

We trained the neural network 1000 epochs with NAdam optimizer. The learning rate is 10^{-3}. To help the training converge, we reduce the learning rate by 0.5 every 128 epochs. To balance the prediction loss and distribution loss, we let $\gamma = 0.1$.

Since we do not want to sample the pattern v, we only let φ_u be invertible, and let φ_v be a free form neural network. This will make φ_v has higher expressivity and easier to be optimized. The φ_u uses 5 invertible blocks and one resize block to reduce the dimension from 4 to 2. Each invertible block contains an invertible linear layer, a Real-NVP layer, and a batch normalization layer. The φ_v is a convolutional neural network that maps $3 \times 64 \times 64$ tensor to a two dimensional vector. Note that the channel is changed from 2 to 3 by the mapping $(a, b) \to (a, b, (a+b)/2)$ to make it have better visualization and easier to do data augmentations, while not losing or alter any information. The detailed neural network structure is shown in Table A6.

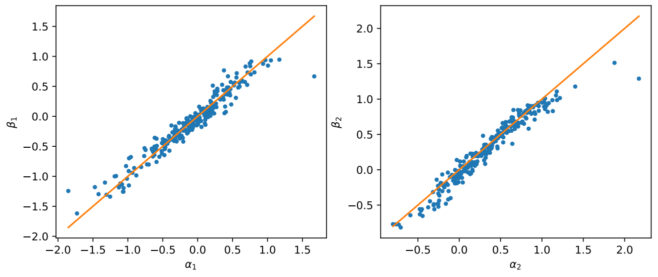


Figure A9: The trained neural network on Turing patterns is capable of predicting the macrostate. Here, each point represents a (α_i, β_i) pair. Here, $\alpha_i = \varphi_u(u_i)$ and $\beta_i = \varphi_v(v_i)$. Here we map the microstates to a two-dimensional space, so we compare the macrostates on each dimension, represented as α_i and β_i.

Figure A9 compares the macrostates mapped from parameters (α) and macrostates mapped from patterns (β). Most points are laying on the $\alpha = \beta$ line, which indicates that this trained neural network made good predictions at macrostate and having high mutual information $I(\alpha; \beta)$.

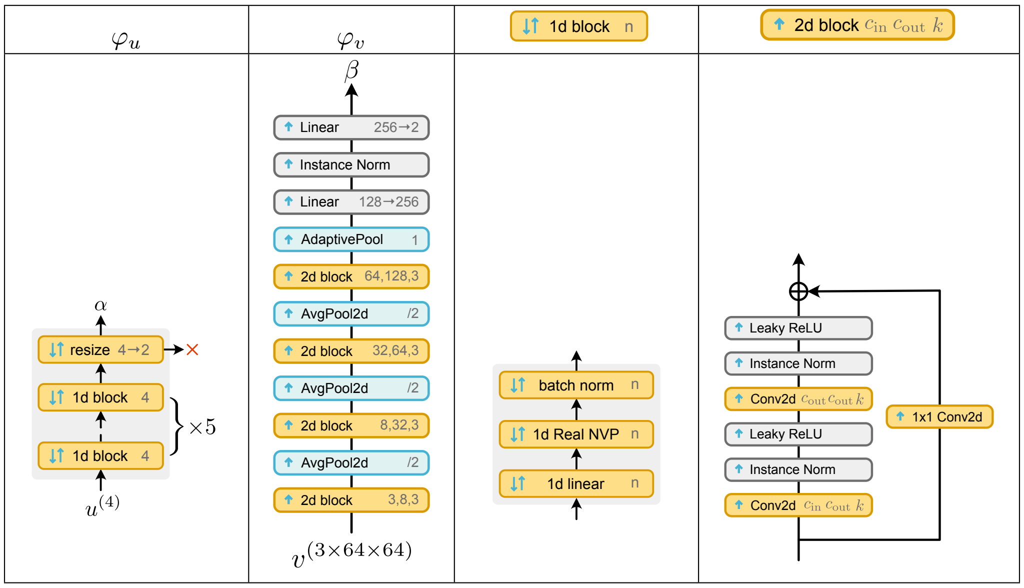


Table A6: The neural network structure for finding macrostates of Turing patterns. For the parameter side (φ_u) we use a 5-layer INN to get a 2-dimensional output. For the pattern side (φ_v), since generation is not needed, we use a free-form neural network to get a 2-dimensional output.