Politecnico di Torino

Master's Degree in Physics of Complex Systems

Scale-Invariant Neural Networks for Percolation

Luca Camagna

Supervisors:
Sergio Chibbaro, Cyril Furtlehner, François Landes
Université Paris-Saclay, LISN

Internal Supervisor:
Alfredo Braunstein
Politecnico di Torino

Acknowledgements

I would first like to express my sincere gratitude to my supervisors, **Sergio Chibbaro** and **Cyril Furtlehner**, for their guidance and, most importantly, for the patience they have shown me after a rather rough start.

I owe my deepest thanks to my parents, for their constant support and understanding of a son who has always been a bit too silent and reserved.

To my dear friends, **le scigne**, thank you for always managing to make me laugh, no matter how far apart we are scattered around the world.

To **quelli di PCS**, thank you for two years filled with company, laughter, and unforgettable memories shared during this madness of a master's degree.

And finally, to **Pietro, Milo, Jacopo, Andrea, and Matteo**, thank you for your unwavering support and for standing by me through all my emotional ups and downs.

Thank you all for being part of this journey.

Abstract

This thesis explores the integration of Renormalization Group theory with Machine Learning to develop a multiscale classifier for predicting square lattice site percolation. The model processes binary lattices of varying sizes, mimicking the iterative RG flow, and demonstrates improved performance when trained on mixed-size data compared to fixed-size training. Key results show that the classifier achieves high accuracy (90–95%) in phase classification, with the arithmetic averaging first coarse-graining (AFC) method proving most effective. The learned coarse-graining rules resemble sigmoidal functions, consistent with theoretical RG expectations, and provide estimates of the percolation threshold (0.569–0.589) close to the known theoretical value. By bridging RG theory with machine learning, this framework offers a physics-informed method for studying critical phenomena.

Contents

I Introduction

Percolation theory is a branch of statistical physics and probability theory that studies how a fluid (or any influence like current, disease, or information) moves and spreads through a medium [4]. At its core, it describes the emergence of long-range connectivity in a random network, exemplified by the sudden formation of a spanning cluster in a lattice as the occupation probability of the sites of the network crosses a critical threshold p_c. This phenomenon exhibits universal scaling laws, making it an example of critical behavior. Traditional analytical and numerical approaches, such as Monte Carlo simulations, have long been employed to study these properties [16].

In recent decades, the renormalization group (RG) formalism has emerged as a transformative tool for analyzing systems near criticality [9]. By iteratively coarse-graining microscopic details, which are proven to be negligible near criticality, while preserving macroscopic observables, allows one to tackle such systems and study their behaviour around the critical point. However, practical implementations of RG, particularly in systems lacking exact solutions, often rely on harsh approximations [18].

Lately some attention has been posed on the performance of machine learning (ML) architectures for the study of critical behaviour. Although some architectures prove to successfully find the percolation threshold [14] and the full percolation probability [5], here we focus on models which are rooted in physics, and specifically the RG approach, addressing the interpretability, or lack thereof, of the former. Moreover, in the statistical physics context, many problems of interest related to critical phenomena are associated to fractal structures, containing information at all scales. This results in notoriously difficult data to capture with standard ML methods. Determining for instance on which side of a second order phase transition is a given critical system requires deciphering subtle signatures over a large spectrum of scales. In the case of 2D percolation at hand, it has been observed [1] that standard tools of image recognition are unable to reliably detect a spanning cluster close to transition for finite systems.

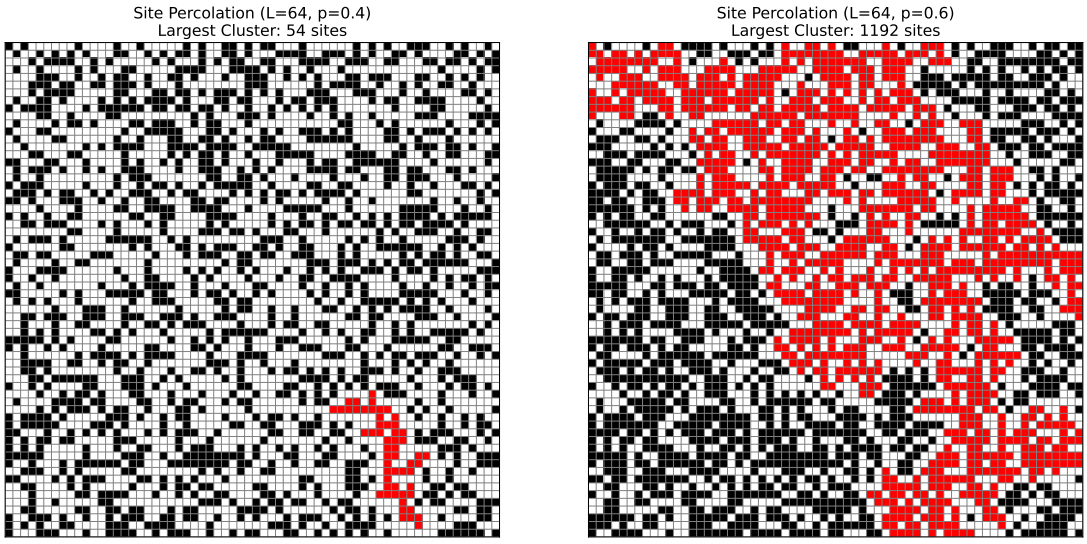
This thesis bridges the conceptual rigor of RG with the adaptive power of machine learning (ML), proposing a neural network architecture that learns effective coarse-graining rules directly from percolation lattices, with the goal of predicting whether a lattice percolates or not, inspired from the large-cell-RG approach [16]. Once a successful and interpretable model is built, it can be also exploited to find the percolation threshold.

In Section II we begin by formalizing Percolation Theory and its relevance under the lenses of critical phenomena. Then, in Section III we outline the principles of real space RG and we detail the design of the classifier mimicking the iterative coarse-graining of RG. We contrast models trained on single lattice sizes with those exposed to multiscale data (i.e. squares of side lengths $3^2, 3^3, 3^4$), demonstrating how mixed training enhances both stability and generalizability, propose different approaches on how to handle the binary data, test the finite size effects of the results, and introduce a model with multiple labels related to the directions of percolation. In Section IV, we report hints of future work, both short-term and long-term.

II Percolation Theory

A fundamental concept in probability theory, *percolation* addresses the following question: what is the probability that a path exists through a random network from one side to another? It is used to model a wide range of phenomena, including the flow of water through soil [8], the spread of diseases in populations [17], the conductivity of random mixtures of materials [15], and the stability of networks [7].

All of the theory summarised below is adapted from [16].



(a) $p < p_c$; no spanning cluster

(b) $p > p_c$, spanning cluster

Figure 1: Comparison of results with and without clustering

There exist different formulations of the problem and here we shall focus on the following.

Square lattice site percolation: *Consider the square lattice given by the set of nodes* $V = \{1, 2, \dots, L\}^2$. *Each node* (i, j) *can be independently* accessible *or not, determined by the Bernoulli random variable*

$$s_{ij} = \begin{cases} 1 & ; \quad \textit{w.p. } p \\ 0 & ; \quad \textit{w.p. } 1-p \end{cases} . \tag{II.1}$$

where p *is called* permeability. *The network is called* percolating *if there exists a connected component, commonly referred to as* cluster*, spanning the entire lattice (see Figure 1)*

This simple model exhibits surprisingly rich behavior, particularly near the percolation threshold where long-range connectivity first emerges.

Percolation transition: *Let* $\Pi(p, L)$ *be the probability that a spanning cluster exists in a network of dimension* L *and permeability* p. *The* percolation threshold *is the permeability* p_c *at which an* infinite cluster *spans the* infinite network*, namely [6]*

$$[0, 1] \ni p_c : \lim_{L\to\infty} \Pi(p, L) = \Theta(p - p_c) \tag{II.2}$$

Note that the limit $L \to \infty$ *is tantamount to replacing the set of nodes with* $V = \mathbb{Z}^2$.

This is neither a statistical mechanics phase transition nor a bifurcation, but rather a mathematical phase transition analogous to phenomena like the traffic flow transition [13].

When a system undergoes a phase transition, its properties can change dramatically with small variations in control parameters. Second-order phase transitions, like the percolation transition, occur continuously without sudden jumps. Near the critical point (the percolation threshold p_c), the system exhibits scale-invariant behavior where clusters of all sizes appear, and physical quantities follow power-law distributions (see II.1 and II.2)

A crucial quantity for describing the critical behaviour of the percolation transition is the following.

Cluster number: *The probability* $n_s(p)$ *that an* arbitrary *site is the left-hand end of a cluster of size* s.

In full generality, given g_{st} the number of cluster configurations of size s and perimeter t, then

$$n_s(p) = \sum_t g_{st} p^s (1-p)^t$$

Unfortunately, the quantity g_{st} not only does not feature a closed formula in two dimension (or any finite dimension greater than $d = 1$, for that matter), but is also computationally hard to find. Nonetheless, starting from this probability, other "observables" can be defined. For example, we may ask the following question: how large is a cluster to which an *arbitrary* occupied site belongs? Calling such random variable $\mathcal{S}$, then

$$w_s \equiv \mathbb{P}(\mathcal{S} = s | s_{ij} = 1) = \frac{\mathbb{P}(\mathcal{S} = s, s_{ij} = 1)}{\mathbb{P}(X_{ij} = 1)} = \frac{s n_s(p)}{p}$$

Mean cluster size: *We refer to*

$$S(p) \equiv \mathbb{E}[\mathcal{S}] = \frac{\sum_s s^2 n_s(p)}{p}$$

as the mean cluster size. *To such quantity, the critical exponent γ is associated*

$$S(p) \propto |p - p_c|^\gamma \quad ; \quad p \to p_c$$

To complete the second order phase transition picture, an order parameter which changes continuously at $p = p_c$ is needed.

Strength: *The probability $P(p)$ that an arbitrary site belongs to the* infinite *cluster. The critical exponent β is defined by*

$$P(p) \propto |p - p_c|^\beta$$

The order parameter $P(p)$, which serves as a proxy for the *normalized average size of the largest cluster*, goes to zero continuously as p approaches p_c from above.[1] Moreover, note that

$$\begin{aligned} p &= \mathbb{P}(s_{ij} = 1) \\ &= \sum_s \mathbb{P}(s_{ij} = 1 \mid (i,j) \text{ is part of an } s\text{-cluster}) \mathbb{P}((i,j) \text{ is part of an } s\text{-cluster}) + P(p) \\ &= \sum_s s n_s(p) + P(p) \end{aligned}$$

II.1 Scaling ansatz

As previously mentioned, there's no exact solution for the distribution $n_s(p)$ in two dimensions. However, we can postulate a scaling form for the cluster number distribution:

$$n_s(p) \sim s^{-\tau} f\left((p - p_c) s^\sigma\right) \quad ; \quad s \gg 1 \quad ; \quad p \to p_c$$

where:

- τ and σ are critical exponents to be determined
- the scaling function $f(z)$ satisfies:

$$f(z) \sim \begin{cases} \text{const} & \text{for } |z| \ll 1 \Leftrightarrow s \ll s_\xi \\ \text{rapid decay} & \text{for } |z| \gg 1 \Leftrightarrow s \gg s_\xi \end{cases}$$

  where $s_\xi \propto |p - p_c|^{-1/\sigma}$ defines the characteristic cluster size.

[1] The largest cluster isn't necessarily the percolating one, but it isn't hard to imagine that this holds true for $L \to \infty$. Hence we can effectively compute numerically the average size of the largest cluster, and divide by L^2 to find $P(p) = \langle S_\infty \rangle / L^2$.

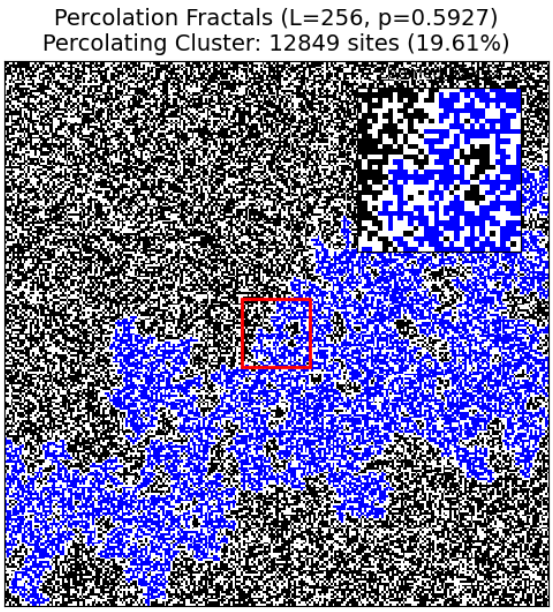


Figure 2: Realization of a percolating cluster for $p \approx p_c$ and zoomed-in view, manifesting scale invariance

This ansatz leads to several important consequences. First, at criticality ($p = p_c$), the distribution becomes a pure power law:

$$n_s(p_c) \sim s^{-\tau}$$

The exponents τ and σ are related to previously introduced critical exponents through scaling relations. For instance, the mean cluster size $S(p)$ can be expressed as

$$S(p) = \frac{1}{p}\sum_s s^2 n_s(p) \sim \int s^{2-\tau} f\left((p-p_c)s^\sigma\right)\, \mathrm{d}s \propto |p-p_c|^{-\gamma}$$

Changing variables to $u = |p - p_c|s^\sigma$ yields the scaling relation

$$\gamma = \frac{3-\tau}{\sigma}\,.$$

Similarly, the strength of the infinite cluster $P(p)$ can be related to the scaling function

$$P(p) = p - \sum_s s n_s(p) \sim |p-p_c|^\beta \tag{II.3}$$

giving the relation

$$\beta = \frac{\tau-2}{\sigma}\,.$$

The scaling function $f(z)$ contains additional universal information about the system. Its precise form is difficult to determine analytically, but it has been studied extensively through series expansions and Monte Carlo simulations. Remarkably, while the exponents τ and σ depend on dimensionality, the qualitative form of the scaling function remains similar across different lattice types in the same dimension.

II.2 Fractal behaviour

The structure of a percolating cluster in large networks is far from simple. Indeed, near the percolation threshold p_c, the geometry of the largest cluster departs significantly from that of a regular object. It is neither line-like nor filled, but forms a pattern that lacks a characteristic length scale.

In order to grasp this quantitatively, we need to define a length scale in the first place. As customary in statistical physics [11], let's introduce the following.

Correlation function and correlation length: *the probability $g(r)$ that site (i,j) is in the same cluster as that of a site at position (ℓ,k), with $r \equiv |\boldsymbol{r}_{ij} - \boldsymbol{r}_{\ell k}|$, is referred to as correlation function. The length scale ξ over which correlations decay is called correlation length.*

The correlation length ξ defines a *length scale* for the system. Indeed, sites distant $r \gg \xi$ are likely not part of the same cluster and vice versa. As the occupation probability p approaches the critical value p_c, the correlation length diverges:

$$\xi \sim |p - p_c|^{-\nu}$$

This divergence signals the absence of any finite cutoff to the range over which sites may belong to the same connected component.

A diverging length scale has two important consequences. First, it implies that the system becomes *self-similar across scales*: clusters exist on all length scales up to the size of the system, and no characteristic scale dominates. This is precisely what defines a fractal structure.

Second, a diverging ξ marks the presence of a continuous phase transition. When ξ is finite, the system looks locally random and fragmented. As $\xi \to \infty$, fluctuations occur on arbitrarily large scales, and a macroscopic cluster spanning the entire system emerges.

In the absence of any basic length scale, all the relevant functions become power laws. For example, at $r \ll \xi$ (which is always true at $p \approx p_c$), the scaling ansatz in dimension $d = 2$ entails [10]

$$g(r) \propto r^{-2\beta/\nu}$$

II.3 Finite size scaling

All of the previous theoretical results hold nicely for $L \to \infty$. However, for both theoretical approaches and simulations purposes, it is crucial to understand how a finite network behaves. Intuitively, we can already expect that for $L \gg \xi$ nothing changes (meaning the characteristic length scale is still ξ), while for $L \gg \xi$ then criticality is governed by the length scale L. In general, quantities behaving like power laws naturally emerge from scaling laws (and viceversa)

$$X(L,\xi) \propto |p - p_c|^{-x} \iff X(L,\xi) \propto \begin{cases} \xi^{x/\nu} & L \gg \xi \\ L^{x/\nu} & L \ll \xi \end{cases}$$

hence studying X *as a function of the system size* at criticality yields information on the ratio x/ν.

An interesting curve is how the percolation probability changes as a function of the size, i.e. $\Pi(p,L)$. For $L \in \{3,4,5\}$ the exact percolation probability can be computed numerically by enumerating all 2^{L^2} configurations $\mathbf{s}_i \in \{0,1\}^{L\times L}$. The percolation probability is

$$\Pi(p,L) = \sum_{\mathbf{s}\in\{0,1\}^{L\times L}} \mathbb{I}[\text{s percolates}] \prod_{ij} p^{s_{ij}} (1-p)^{1-s_{ij}}$$

where $\mathbb{I}[\text{s percolates}]$ is the indicator function that equals 1 if the lattice percolates (has a connected path of occupied sites between either opposite boundaries), and s_{ij} denotes the state of site (i,j) as in eq. (II.1).

For $L \geq 6$, it estimates $\Pi(p,L)$ using Monte Carlo sampling with the following approach:

- Generate Lattices: for each (L,p), we create an $L \times L$ boolean grid where each site $s_{ij} \in \{0,1\}$ is accessible with probability p.
- Detect percolation: We check for a spanning cluster connecting the top and bottom edges. A simple way is to perform a breadth-first search (BFS, [12]) from all open sites in the top row, marking visited sites and stopping if any visited site reaches the bottom row. This finds connected components on the lattice.

- Monte Carlo averaging: for each (L, p) we repeat the lattice generation and connectivity check many times to estimate the percolation probability as the fraction of trials that percolate, i.e.

$$\hat{\Pi}(p, L) \approx \frac{1}{N} \sum_{k=1}^{N} \mathbb{I}[\mathbf{s}_k \text{ percolates}] = \Pi(p, L) + \mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$$

- Finite-Size scaling: we use the known critical threshold $p_c \approx 0.59724$ and critical exponent $\nu = 4/3$. We plot $\hat{\Pi}(p, L)$ vs p for each L, and then plot the same data against the scaled variable $x \equiv (p - p_c)L^{1/\nu}$.

Theory predicts the curves for different L should collapse onto a universal function of x. Indeed, the simulation (Algorithm 1) produces Figure 3. As L increases, the curves approach a step function, hence the phase transition as defined in eq. II.2.

It is worth mentioning that if we restrict to percolation in a specific direction (top-down or left-right), for *rectangular* $a \times b$ there exists a recursive approach of time complexity $\mathcal{O}(\max\{a, b\}^3 2^{\min\{a,b\}})$ for the percolation probability $\Pi(p, a, b)$, which is explored in Section D.

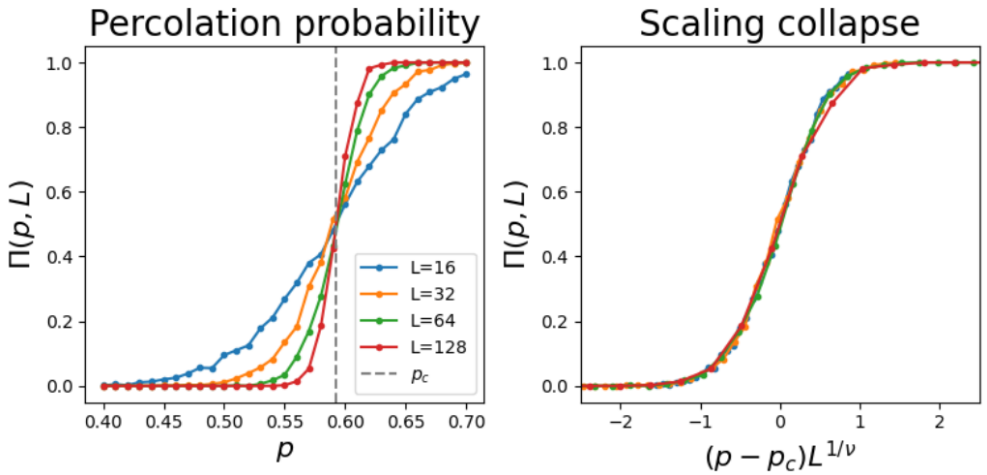


Figure 3: $\Pi(p, L)$ for $L \in \{16, 32, 64, 128\}$. For each value of p and L, the probability $\Pi(p, L)$ has been estimated with 1000 samples.

III RG-Inspired Classifier

In the context of supervised machine learning [2], a *classifier* is a model designed to assign input data to one of several predefined categories. For example, in percolation studies, a classifier might be trained to decide whether a given lattice configuration belongs to the percolating or non-percolating phase. The model learns this task by analyzing labeled examples and adjusting its internal parameters to minimize classification errors. Classifiers can range from simple linear models to deep neural networks with many layers of abstraction. In recent years, ML techniques have garnered attention for their potential to analyze critical behavior in complex systems. Notably, various ML architectures have been applied to percolation models. For instance, [14] showed that regression-based ML models, such as gradient boosting and random forests, can accurately predict the percolation threshold across diverse network structures by leveraging structural features of the networks. Similarly, [5] employed convolutional neural networks to estimate both the percolation threshold and the full percolation probability in two-dimensional lattice systems.

Despite these results, these structures lack interpretability. Many of these ML models function as "black boxes," trading off interpretability for complexity. Moreover, critical phenomena in statistical physics often involve fractal structures and scale invariance,

characteristics that standard ML models may struggle to capture. For example, accurately determining the phase of a system near a second-order phase transition requires detecting subtle, scale-spanning features, hence a task that can be challenging for conventional ML architectures. In the context of two-dimensional percolation, [1] observed that standard image recognition tools, including CNNs, fail to reliably detect the presence of a spanning cluster near the percolation threshold in finite systems.

Given these considerations, our focus shifts toward integrating ML approaches with physics-based methodologies, particularly the RG framework.

III.1 Renormalization Group approach

Let us briefly formalize the RG approach for percolation as the theoretical foundation of the model.

The renormalization group is an approach to the problem of critical phenomena which has been developed in the 60's–70's [18]. It is mainly a collection of concepts and tools, which in some cases crystallize in a well–defined approach. The name renormalization group is possibly unfortunate, since the transformations used do not form a mathematical group and the renormalization in the sense of quantum field theory is not an essential ingredient, in spite of certain technical analogies. The essential ingredient is *coarse–graining*, which induces a flow in the space of parameters of the model under investigation (namely the permeability p in our case).

In the real-space RG approach to site percolation, the lattice is partitioned into blocks (cells) of size $b \times b$. Each block is then replaced by a single "super-site" whose occupation probability p' is determined by a coarse-graining rule applied to the original sites within the block. This procedure defines a transformation $p' = R_b(p)$ that, when applied iteratively, defines a flow for the value p.

As previously mentioned, the lack of a length scale (i.e. $\xi \to \infty$) signals a phase transition. The fixed point(s) $p^* = R_b(p^*)$ of the transformation signals the phase transition. Indeed, after one RG step, the correlation length transforms as

$$\xi' = \frac{\xi}{b} \tag{III.1}$$

therefore the fixed point is equivalent to $\xi = \xi/b$. We conclude then

- $\xi = 0 \iff p^* = 0 \vee p^* = 1$
- $\xi \to \infty \iff p^* = p_c$

that is to say, the non-trivial fixed point of this transformation corresponds to the critical percolation threshold.

As noted in Section II.2, near the critical point p_c the correlation length diverges as $\xi \propto |p - p_c|^{-\nu}$, hence after an RG step equation (III.1) becomes

$$\begin{aligned} \xi' &= \text{const. } |p' - p_c|^{-\nu} = \text{const. } |R_b(p) - p_c|^{-\nu} \\ &= \frac{\xi}{b} = \text{const. } |p - p_c|^{-\nu} \end{aligned}$$

which can be solved for

$$\left(\frac{|R_b(p) - p_c|}{|p - p_c|} \right)^{-\nu} = \frac{1}{b}$$

which, in the limit $p \to p_c$ becomes **a way to estimate the critical exponent** ν

$$\nu = \frac{\ln b}{\ln \frac{\mathrm{d}R_b}{\mathrm{d}p}\Big|_{p=p_c}} . \tag{III.2}$$

III.2 Foundation of the classifier

The implemented classifier embodies a machine-learned realization of the real-space RG approach. Formally, the model operates on lattice configurations $\mathbf{s}^{(0)} \in \{0,1\}^{L\times L}$ where $L = b^k$ represents the block side length, for some integers k, and b. The core computational procedure consists of an iterative coarse-graining transformation of the matrix $\mathbf{s}^{(0)}$ into a scalar by recursively applying the following procedure:

1. **Block decomposition**: At each RG step t, the lattice $\mathbf{s}^{(t)} \in \mathbb{R}^{L_t\times L_t}$ is partitioned into non-overlapping blocks of size $b \times b$, yielding $N_t = (L_t/b)^2$ patches:

$$\mathcal{P}^{(t)} = \left\{ \mathbf{p}_{i,j}^{(t)} \in \mathbb{R}^{b\times b} \mid 0 \le i,j < L_t/b \right\}$$

2. **Learned renormalization rule**: Each block $\mathbf{p}_{i,j}^{(t)}$ is flattened to a vector $\mathbf{v}_{i,j}^{(t)} \in \mathbb{R}^{b^2}$ and transformed by a parameterized function $f_\theta : \mathbb{R}^{b^2} \to [0,1]$ which **maps the whole patch into a scalar**

$$\mathbb{R} \ni s_{i,j}^{(t+1)} = f_\theta(\mathbf{v}_{i,j}^{(t)})$$

   where f_θ is implemented as a neural network

$$f_\theta(\mathbf{v}) = \sigma(\mathbf{W}_2 \cdot \mathrm{ReLU}(\mathbf{W}_1\mathbf{v} + \mathbf{b}_1) + b_2)$$

   with learnable parameters $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, b_2\}$, and σ denoting the sigmoid activation. **Unless otherwise specificed, the neural network is comprised of $n = 64$ neurons and $\ell = 1$ layers.**

3. **Coarse-grained lattice**: The transformed scalars $s_{i,j}^{(t+1)}$ constitute **the coarse-grained lattice new lattice** $\mathbf{s}^{(t+1)} \in \mathbb{R}^{(L_t/b)\times(L_t/b)}$:

$$\mathbf{s}^{(t+1)} = \begin{pmatrix} s_{0,0}^{(t+1)} & \cdots & s_{0,N-1}^{(t+1)} \\ \vdots & \ddots & \vdots \\ s_{N-1,0}^{(t+1)} & \cdots & s_{N-1,N-1}^{(t+1)} \end{pmatrix}, \quad N = L_t/b$$

This transformation is applied iteratively for $T = \lfloor \log_b L \rfloor$ steps, until the lattice $\mathbf{s}$ is coarse-grained into a scalar value $q = s^{(T)}$.

The parameters θ are optimized using a training dataset $\mathcal{D} = \{(\mathbf{s}_i, y_i)\}_{i=1}^N$, where the lattices are generated by creating a boolean grid where each site $s_{ij} \in \{0,1\}$ is accessible with permeability p and

$$\begin{aligned} y_i &= \mathbb{I}[\mathbf{s}_i \text{ percolates}] \\ &= 1 - (1 - \mathbb{I}[\mathbf{s}_i \text{ top-down percolates}])(1 - \mathbb{I}[\mathbf{s}_i \text{ side-to-side percolates}]) \\ &= 1 - (1 - \tau_i^{(1)})(1 - \tau_i^{(2)}) \end{aligned} \tag{III.3}$$

by optimizing what's known as **binary cross-entropy loss**

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} [y_i \log q_i + (1 - y_i)\log(1 - q_i)] \tag{III.4}$$

which is a fundamental objective function in machine learning for binary classification tasks, as it quantifies the discrepancy between predicted probabilities and true binary labels. The optimization task is performed via Stochastic Gradient Descent [3]

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}_B(\theta_t)$$

where $\mathcal{L}_B(\theta_t)$ is the loss in eq. III.4 but computed on a batch of size B of the dataset $\mathcal{D}$.

The two types of percolation (top-down and side-to-side) in eq. (III.3) have been associated to boolean variables $\tau^{(1)}$, $\tau^{(2)}$ and have been made explicit to highlight that, for the time being, the architecture doesn't distinguish between the two (see Section III.5 for an extension of the model).

The model f_θ learns an effective block transformation R_b that maps local configurations to renormalized probabilities (for the sake of clarity, the interpretation of such probabilities shall wait). Unlike analytical RG where R_b is prescribed, here R_b emerges from data-driven optimization while maintaining the RG structure through the iterative coarse-graining procedure.

The learned function f_θ takes vectors in $\mathbb{R}^{b^2}$ as input. However, as noted in Section III.1, the analytical coarse-graining rule maps probabilities into probabilities. Hence, in order to extrapolate an estimate $\hat{R}_b(p)$ of the rule $R_b(p)$ from f_θ we can project it as

$$\hat{R}_b(p) = f_\theta(p\mathbf{1}) \quad ; \quad \mathbf{1} \equiv (1, 1, \dots, 1) \in \mathbb{R}^{b^2}$$

For what concerns the classification task, the final prediction is given by

$$\hat{y} = \mathbb{I}\left[q > \frac{1}{2}\right] .$$

The pseudocode for the foundation of the classifier is reported in Algorithm 2.

III.2.1 Handling the first layer

The RG approach, formally speaking, maps a probability p into another $p' = R_b(p)$, hence it can be argued that feeding the neural network directly configurations $\mathbf{s} \in \{0, 1\}^{L\times L}$ is counter-intuitive. Therefore three variations for the first step (i.e., the one dealing with the binary input data) are explored and compared throughout this report:

- **NFC approach: N**o **F**irst **C**oarse-graining at all, hence the algorithm explained above
- **AFC approach:** generating the first coarse-grained lattice as

  $$s^{(1)}_{i,j} = \left\langle \mathbf{p}^{(0)}_{i,j} \right\rangle = \frac{1}{b^2} \sum_{\ell,k} (p^{(1)}_{i,j})_{\ell,k}$$

  hence performing arithmetic **A**verage as a **F**irst **C**oarse-graining of the binary values
- **PFC approach:** generating the first coarse-grained lattice as in AFC, then mapping the values onto the corresponding **P**ercolation probability using the function $\Pi(p, b)$, meaning

  $$s^{(1)}_{i,j} = \Pi\left(\left\langle \mathbf{p}^{(0)}_{i,j} \right\rangle, b\right) = \Pi\left(\frac{1}{b} \sum_{\ell,k} (p^{(1)}_{i,j})_{\ell,k}, b\right)$$

  computed numerically as in Section II.3.

In Figure 4 a sketch of how a lattice is handled by the classifier is shown.

III.3 Fixed-size training and testing

At first we provide the network with samples $\mathcal{D}_{b,k} = \{(\mathbf{s}_i, y_i)\}_{i=1}^N$ with $\mathbf{s} \in \{0, 1\}^{b^k\times b^k}$. We train the network multiple times, compare different configurations of (b, k), by using each of the three approaches described in Section III.2.1.

In Table 1 we report the results with $L = 3^2$ and $L = 3^3$, in the three approaches. For each run, the number of samples is $N_{\text{train}} = 10^3$, learning rate is $\eta = 10^{-3}$, number of epochs is $E = 10$ and batch size $B = 10$. The accuracy is simply computed as the

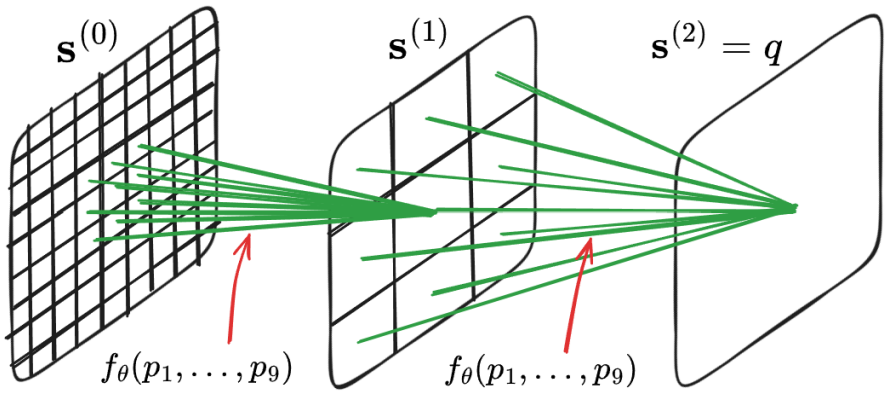


Figure 4: Sketch of the handling of a lattice by the classifier

percentage of missclassified lattices in a freshly generated test dataset $\mathcal{D}_{\text{test}} = \{\mathbf{s}_i, y_i\}_{i=1}^{N_{\text{test}}}$, i.e.

$$\epsilon_{\text{test}} \equiv \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbb{I}[y_i = \hat{y}_i]$$

with $N_{\text{test}} = 100$.

For what concerns the learned rule $\hat{R}_b(p)$, we note that three situations arise (in Figure 5 an output example is provided, however a full report can be found in Appendix B), as the network learns one of the following

1. monotonically increasing sigmoid
2. monotonically decreasing sigmoid
3. curve with a minimum and a maximum

In all of the three cases, the classifier works well in determining the phase of lattices whose size is the same as training. However, when generalizing to different size, only the monotonically increasing sigmoid shows good results.

We shall address the elephant in the room: how does a decreasing curve work so well? The answer lies in the neural network structure: the output value q can be (not formally) expressed by the operation

$$q = s^{(T)} = \left(\bigcirc_{i=2}^{T} f_\theta\right)(f_1(\mathbf{s}))$$

where $f_1(\mathbf{s})$ has been made manifest since it depends on which approach has been chosen for the first layer. If f_θ is monotonically decreasing, with each composition the learned rule "flips" back and forth between increasing and decreasing. Therefore, a neural network trained on size b^k which learns the decreasing curve generalizes well to powers $b^{k'}$ with $|k' - k| \geq 2$. The third kind of curve could then be a combination of the former two.

In Table 2 we report the estimate of the critical point. For the same arguments as above, both the first and second kind of curves yield the same estimate of p_c. Indeed, the intersection of a sigmoid $S(x)$ with the function $(S \circ S)(x)$ occurs trivially at the fixed point x^*

$$S(x^*) = S(S(x^*)) \Longrightarrow S^{-1}(S(x^*)) = S^{-1}(S(S(x^*)) \Longrightarrow S(x^*) = x^*\,, \tag{III.5}$$

[2]. However, the AFC approach provides a much better estimate of p_c (although still not satisfactory), and therefore it shall be the preferred method henceforth.

[2]The function $S^{-1}(x)$ is well-defined since $S : [0, 1] \to [0, 1]$ is a bijection.

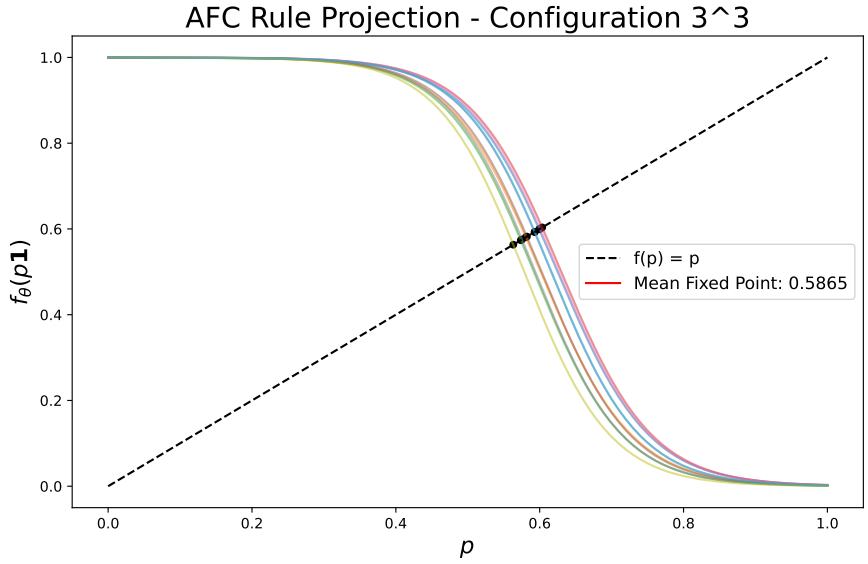


Figure 5: 10 different instances of learned rule $\hat{R}_b(p) = f_\theta(p\mathbf{1})$, trained on $N = 10^3$ lattices of side length $L = 3^3$, learning rate $\eta = 10^{-3}$, batch size $B = 10$, and number of epochs $E = 10$. For a full picture, we refer to Appendix B.

Table 2 also presents the accuracy of the model when presented with critical data, i.e. lattices generated with $p \sim \mathcal{U}(0.55, 0.65)$. However at this stage this is mostly reported for completeness since, as described in Section III.1, the RG approach works best as the size k increases. Indeed, the more steps the coarse-graining procedure requires, the closer the rule $\hat{R}_b(p)$ will be to the fixed point. If anything, a low accuracy is nothing but a different way to interpret the finite-size effects of the curve $\Pi(p, L)$ for low L.

Of course there are further refinements which can be implemented to improve the results. First of all, as customary for ML models, we could track the evolution of the loss $\mathcal{L}(\theta)$ and accuracy ϵ_{test}, in order to ensure proper convergence and fine-tune the amount of neurons and layers in the neural network. Moreover, given the struggle of ML architectures with fractal data, another safety measured is having a larger training sample around the critical point could prove to be beneficial. Nonetheless they will be explored in the following section, where a more reliable method is established.

b^k \ ϵ_{test}	NFC	AFC	PFC
3^2	0.353 ± 0.375	0.907 ± 0.025	0.916 ± 0.025
3^3	0.909 ± 0.026	0.921 ± 0.019	0.919 ± 0.200
3^4	0.326 ± 0.402	0.950 ± 0.021	0.960 ± 0.016

Table 1: Fraction of successfully classified lattices of various sizes, generated with $p \sim \mathcal{U}(0, 1)$ and evaluated across the three approaches (NFC, AFC, PFC). The network was trained solely on size 3^3 lattices (highlighted in cyan); 3^2 and 3^4 rows (highlighted in yellow) test its ability to generalize to smaller and larger sizes. Each entry is the average over 10 runs.

Metric \ Approach	NFC	AFC	PFC
$\hat{p}_c$	0.47345 ± 0.02651	0.557560 ± 0.01265	0.63453 ± 0.02908
ϵ_{test}	0.766 ± 0.022	0.781 ± 0.027	0.761 ± 0.024

Table 2: Estimate of the percolation threshold and fraction of successfully classified lattices of the same size as that of training (here 3^2) with $p \sim \mathcal{U}(0.55, 0.65)$ and in the three approaches. Each entry is the average over 10 runs.

III.4 Mixed-size training

Training the network with lattices of mixed size, i.e. $k \in \{2, 3, 4\}$, reveals results which are much more stable. Here we shall focus on AFS approach, which proved to be most successful for critical data in the previous section.

Convergence parameters are as above, but now we shall fine-tune the number of training samples N, number of parameters $|\theta|$ in the neural network, and epochs E. Namely, in Section C we report some examples of the evolution of train and test loss, for different ratios $|\theta|/N$ and for $b \in \{3, 4, 5\}$.

In Figure 6 and Table 3 we note that the curves are all of the first kind, and the accuracy across the board is improved with respect to the previous case.

$b = 3$		$b = 4$	
b^k	ϵ_{test}	b^k	ϵ_{test}
3^4	0.959 ± 0.0250	4^4	0.972 ± 0.0260
3^5	0.962 ± 0.0340	4^5	0.968 ± 0.035
3^6	0.969 ± 0.0310	4^6	0.962 ± 0.040
3^7	0.960 ± 0.0800	4^7	0.988 ± 0.014

Table 3: Fraction of successfully classified lattices of various sizes, with b^k sites and $b \in \{3, 4\}$, generated using $p \sim \mathcal{U}(0, 1)$. The neural networks were trained on a mixture of lattice sizes with $k \in \{2, 3, 4\}$ (i.e., up to b^4 sites). Only the $k = 4$ rows correspond to sizes seen during training; the $k \in \{5, 6, 7\}$ rows test generalization to larger lattices. Each entry is the average over 15 runs, each with $N_{\text{test}} = 100$.

For what concerns critical data, $N/2$ training samples have been generated with $p \in (0.55, 0.65)$. **This policy shall be maintained throughout the whole report henceforth.**

For a better estimation of p_c, each entry corresponds to an average 50 instances of learned rule (which are also the curves in Figure 6), albeit sacrificing testing over larger powers due to computational limitations. In Table 4 we note that the value $\hat{p}_c$ is a better estimate than before.

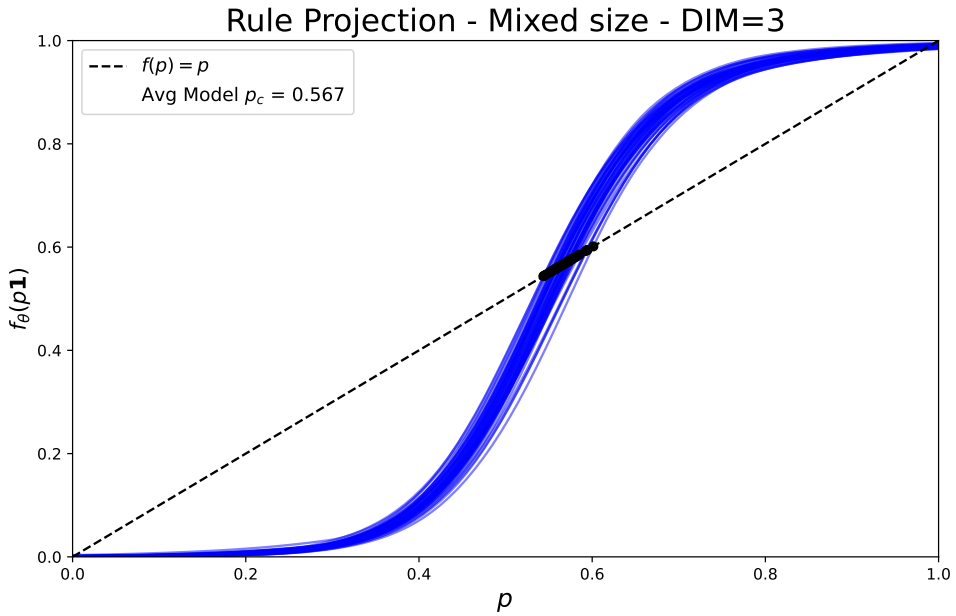
$b = 3$			$b = 4$			$b = 5$		
b^k	$\hat{p}_c$	ϵ_{test}	b^k	$\hat{p}_c$	ϵ_{test}	b^k	$\hat{p}_c$	ϵ_{test}
3^2	0.569	0.795 ± 0.050	4^2	0.577	0.785 ± 0.057	5^2	0.589	0.768 ± 0.057
3^3		0.780 ± 0.058	4^3		0.826 ± 0.050	5^3		0.838 ± 0.045
3^4		0.852 ± 0.045	4^4		0.881 ± 0.037	5^4		0.848 ± 0.042
3^5		0.895 ± 0.036	4^5		0.889 ± 0.034	5^5		0.851 ± 0.032

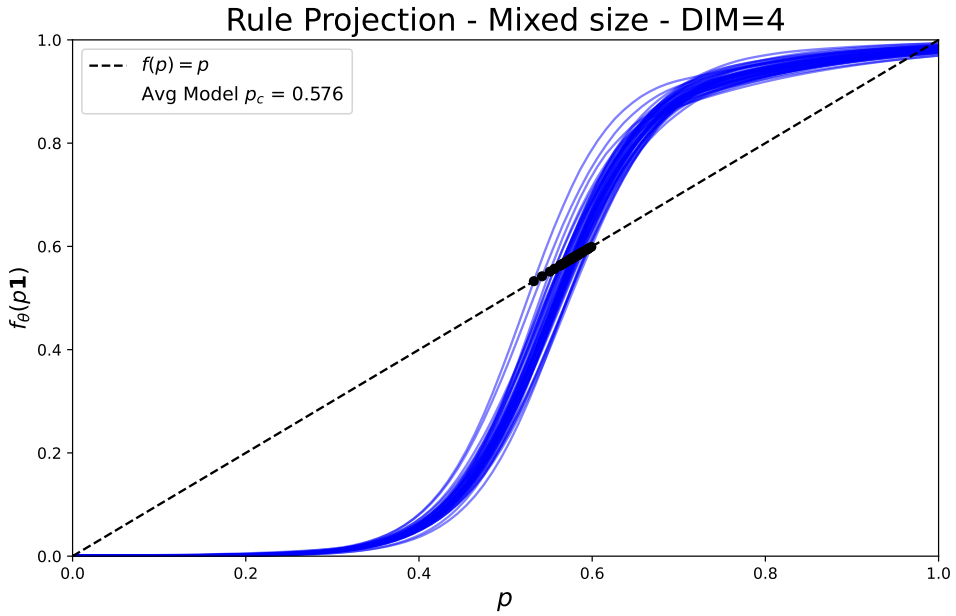
Table 4: Estimated percolation threshold $\hat{p}_c$ (one per base b) and classification accuracy ϵ_{test} with standard deviation, for various lattice sizes b^k. Neural networks were trained on a mix of lattice sizes with $k \in \{2, 3, 4\}$; these rows (highlighted in cyan) correspond to sizes used in training. The $k = 5$ row (highlighted in yellow) shows generalization performance on unseen, larger lattices. Each value is averaged over 50 runs, with $N_{\text{test}} = 100$.

As already anticipated in Section III.3, the accuracy ϵ_{test} improves as k increases: this is due to the fact that the RG approach holds true for $k \to \infty$. The larger k is, the

more the function $f_\theta(\mathbf{p})$ approaches the step function $\Theta(p - \hat{p}_c)$, hence **the bottleneck in performance is the estimate $\hat{p}_c$ itself, which improves as b increases.**

This can be ascribed to the fact that, since there's no distinction between the direction of percolation as noted in Section III.2, a proposed RG scheme for square lattices is *large-cell renormalization* [16], where the coarse-graining rule is directly **the percolation probability of the $b \times b$ sublattice**: $R_b(p) = \Pi(p, b)$. As the name suggests, the method improves as b increases, which is what occurs for the classifier as well.





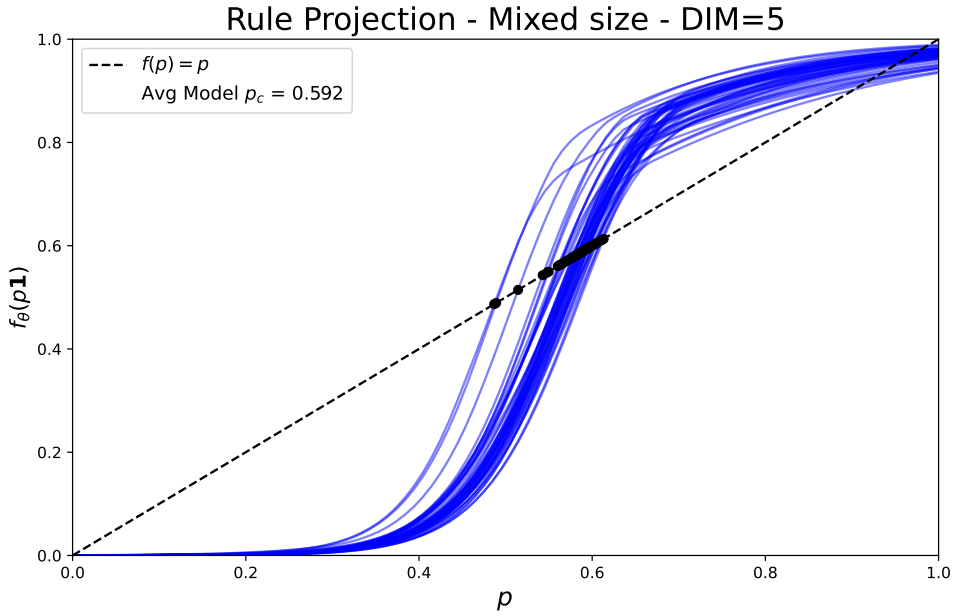


Figure 6: Projections $f_\theta(p\mathbf{1})$, learned by training with lattices of mixed size and with a denser sample around the critical point. The estimation $\hat{p}_c$ improves with base side length b, although the standard deviation σ_{p_c} is noticeable, especially for $b = 5$.

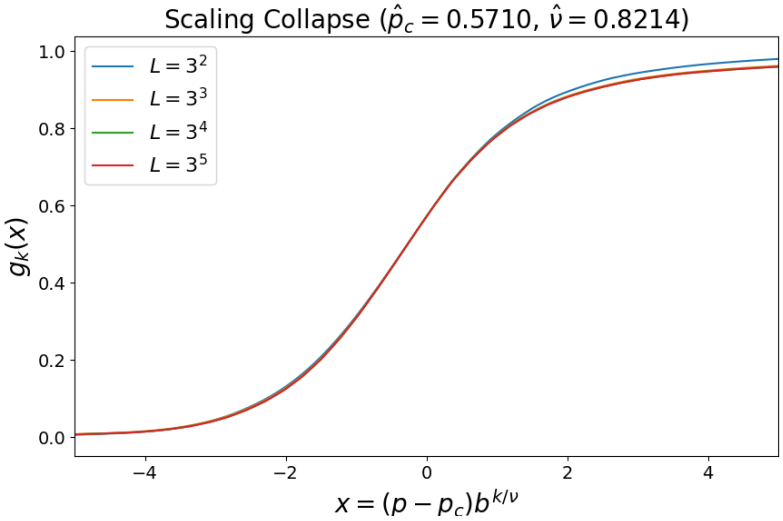


Figure 7: Scaling collapse of the functions $g_k(x)$ defined in eq. (III.6) for $k \in \{1, \dots, 6\}$. The parameters in f_θ have been optimized with learning rate $\eta = 10^{-3}$, epochs $E = 10$, number of training samples $N = 2 \cdot 10^4$, and batch size $B = 10$.

III.4.1 Comparison with the scaling function

There is already strong suspicion that the classifier is learning directly the percolation probability curve of the sublattice. In order to grasp further properties of $\hat{R}_b(p)$, we can test the **scaling collapse** by plotting the functions

$$g_k(x) \equiv \left[\bigcirc_{i=1}^{k} f_\theta(p\mathbf{1})\right](x) \quad ; \quad x \equiv (p - \hat{p}_c)b^{k/\nu} \tag{III.6}$$

for $k \in \mathbb{N}$ and rescale them as explained in Section II.3. Indeed, the function $g_k(x)$ pertains to the coarse-graining of a lattice of side length $L = b^{k+1}$. The result is reported in Figure 7, where $b = 3$ has been chosen.

In addition to that, we are also going to follow a different visualization approach:

- generate N_{sample} lattices of side length b^k for values of $p \in (0.4, 0.8)$
- compute the average prediction $\langle q \rangle$ of the neural network
- plot the curve $\langle q \rangle$ vs $x = (p - \hat{p}_c)b^{k/\nu}$
- repeat for $k \in \mathbb{N}$

The results are reported in Figure 8 and this visualization method shall be exploited once more in the following section.

In both cases it is not a surprise that the curves intersect the (estimated) fixed point $\hat{p}_c$, due to eq. (III.5). However, the true critical exponent $\nu = 4/3$ has been used to produce the plots, which explain the deviation of the rescaled functions from the collapse. The immediate fix is to use eq. (III.2) to find an estimate $\hat{\nu}$ from the learned rule $\hat{R}_b(p)$, and then use the rescaled variable $x = (p - \hat{p}_c)b^{k/\hat{\nu}}$.

III.5 Directional percolation

We note that in Section III.2, the model doesn't distinguish between the directions of percolation. Let's now generalize the model to keep track individually of the two possible ways in which two opposite sides of a square can be connected by a cluster. Denoting them by $i \in \{1, 2\}$ and introducing the labels $\tau^{(1)}, \tau^{(2)}$ the goal now is to learn a coarse-graining rule $f_\theta : \mathbb{R}^{2b^2} \to \mathbb{R}^2$. During the coarse-graining procedure performed by the network, to each cell of the original lattice we associate 2 values, effectively turning it into

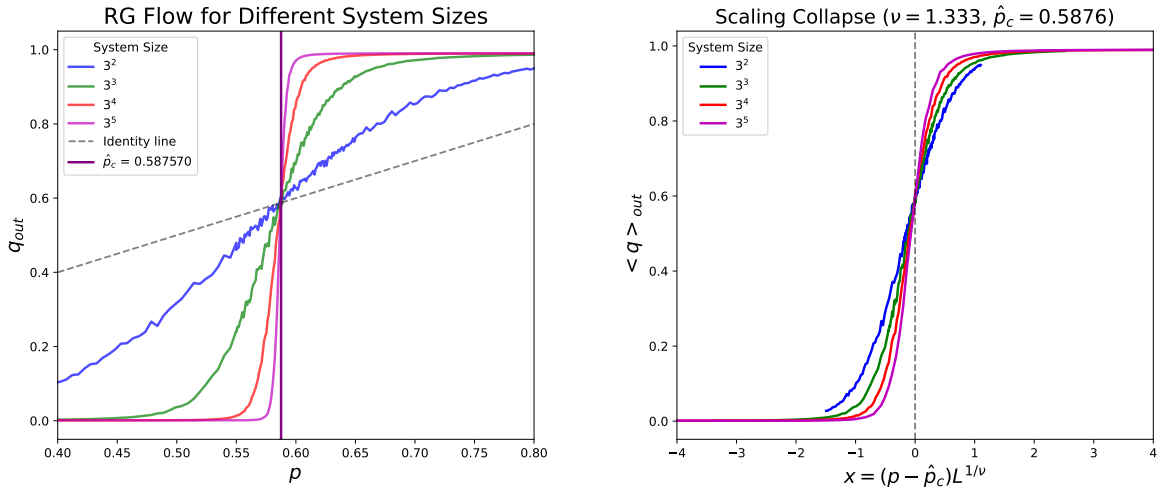


Figure 8: Scaling collapse of the prediction q from a neural network trained with learning rate $\eta = 10^{-3}$, epochs $E = 8$, number of training samples $N = 2 \cdot 10^3$, and batch size $B = 10$. Each point $\langle q \rangle (x)$ is the average over $N_{\text{samples}} = 500$

a **hyperlattice**. The pipeline of Section III.2 is maintained, meaning that at step t of the coarse-graining procedure we have a hyperlattice $\mathbf{s}^{(t)} \in \mathbb{R}^{L_t \times L_t \times 2}$, then we proceed with a block decomposition

$$\mathcal{P}^{(t)} = \left\{ \mathbf{p}_{i,j}^{(t)} \in \mathbb{R}^{b \times b \times 2} | 0 \leq i, j < L_t/b \right\}$$

and via SGD the parameters θ are learned in order to compute

$$\mathbb{R}^2 \ni \mathbf{s}_{i,j}^{(t+1)} = f_\theta(\mathbf{p}^{(1)}, \mathbf{p}^{(2)}) \quad ; \quad \mathbf{p}^{(i)} \in \mathbb{R}^2$$

and to form the lattice $\mathbf{s}^{(t+1)} \in \mathbb{R}^{(L_t/b) \times (L_t/b) \times 2}$.

The loss function now reads

$$\mathcal{L}(\theta) = -\frac{1}{2N} \sum_{i=1}^{N} \sum_{k=1}^{2} \left[\tau_i^{(k)} \log q_i^{(k)} + (1 - \tau_i^{(k)}) \log\left(1 - q_i^{(k)}\right) \right]$$

Among the output vector values $\mathbf{q} = (q^{(1)}, q^{(2)})$, the relevant ones are the first two, since they are related to top-down and left-right connectivity, meaning the actual directions for which the percolation transition occurs.

For what concerns visualization of the rule, the problem isn't straightforward anymore since f_θ now maps to $\mathbb{R}^2$ instead of scalars. However, from the prediction $\mathbf{q}$ we can compute a scalar prediction in the same fashion of eq. (III.3)

$$\mathcal{Q} \equiv q^{(1)} + q^{(2)} - q^{(1)} q^{(2)}$$

and then visualize the RG flow with the method explained in Section III.4.1, i.e. by plotting $\langle \mathcal{Q} \rangle$ vs p for different system sizes, averaged over many runs. One may argue that the correct quantity to plot is $\mathcal{Q} = 1 - \prod_{i=1}^{6}(1 - q^{(i)})$. However, as previously mentioned, the RG approach is supposed to work only for the directions in which the theory holds, i.e. $q^{(1)}$ and $q^{(2)}$. Extending the model to include $q^{(3)}, \ldots, q^{(6)}$ would require a completely different approach, not deep-rooted in iterative coarse-graining.

IV | Conclusion

This report has explored the integration of renormalization group (RG) theory with machine learning to develop a scale-invariant classifier for square lattice site percolation. By

designing neural networks that learn coarse-graining rules inspired by the RG framework, we have created models capable of predicting percolation states across multiple scales while maintaining physical interpretability.

Our key findings demonstrate that:

- The arithmetic averaging first coarse-graining (AFC) approach outperforms other initialization strategies, achieving 90-95% accuracy in phase classification across lattice sizes
- Mixed-size training (using lattices of side lengths b^2, b^3, and b^4) significantly enhances model stability and generalizability compared to fixed-size training, as well as achieving almost 90% accuracy when classifying fractal data already at side length $L = 4^4$
- The learned coarse-graining rules consistently converge to sigmoidal functions resembling theoretical RG transformations
- The hybrid ML-RG approach provides reasonable estimates of the percolation threshold p_c (0.569-0.589 depending on block size b), approaching the theoretical value of 0.5927

The multiscale architecture effectively captures the fractal nature of critical percolation configurations by processing information hierarchically through successive coarse-graining steps. This addresses a fundamental limitation of conventional CNNs in detecting spanning clusters near criticality. Visualizations of the learned rules confirm that the network approximates the RG flow, with the sigmoidal shape reflecting the characteristic probability renormalization observed in theoretical RG treatments.

Future work could explore:

- Adjusting the range of p values over which the critical lattices are generated, based on the side length b^k. Indeed, lattices show fractal properties when p is in the range such that $\Pi(p, b^k)$ sufficiently departs from 0 or 1.
- As introduced in Section III.5, informing the network of the directions of percolation. This type of neural network will likely require more neurons n and layers ℓ than the one used so far.
- Investigating deeper architectures with attention mechanisms to better capture long-range correlations, althought it could prove challenging to maintain interpretability.
- Applying the framework to other critical phenomena exhibiting scale invariance.
- Changing the learning structure fundamentally by using message-passing algorithms instead of gradient-based learning

By maintaining a direct connection to RG theory while exploiting the flexibility of neural networks, this work provides a foundation for physics-informed machine learning approaches to critical systems. The resulting models offer both predictive power and interpretability, bridging a gap between theoretical physics and data-driven methods.

Bibliography

[1] Djénabou Bayo, Andreas Honecker, and Rudolf A. Römer. "The percolating cluster is invisible to image recognition with deep learning". In: *New Journal of Physics* 25.11 (2023). Also available as arXiv:2303.15298 [cond-mat.dis-nn], p. 113041. DOI: 10.1088/1367-2630/ad0525. URL: https://doi.org/10.1088/1367-2630/ad0525.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] Léon Bottou, Frank E Curtis, and Jorge Nocedal. "Optimization methods for large-scale machine learning". In: *arXiv preprint arXiv:1606.04838* (2018). URL: https://doi.org/10.48550/arXiv.1606.04838.

[4] S. R. Broadbent and J. M. Hammersley. "Percolation processes". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 53.3 (1957), pp. 629–641. DOI: 10.1017/S0305004100032680.

[5] Shu Cheng et al. "Machine Learning Percolation Model". In: *Annals of Physics* 435 (2021), p. 168439. DOI: 10.1016/j.aop.2021.168439. arXiv: 2101.08928 [cond-mat.dis-nn].

[6] Kim Christensen. "Percolation theory". In: *Imperial College London* 1 (2002), p. 87.

[7] Reuven Cohen et al. "Resilience of the internet to random breakdowns". In: *Physical review letters* 85.21 (2000), p. 4626.

[8] Allen Hunt, Robert Ewing, and Behzad Ghanbarian. *Percolation theory for flow in porous media*. Vol. 880. Springer, 2014.

[9] Leo P. Kadanoff. "Scaling Laws for Ising Models Near T_c". In: *Physics Physique Fizika* 2.6 (1966), pp. 263–272. DOI: 10.1103/PhysicsPhysiqueFizika.2.263. URL: https://journals.aps.org/ppf/abstract/10.1103/PhysicsPhysiqueFizika.2.263.

[10] Harry Kesten. "Scaling Relations for 2D-Percolation". In: *Communications in Mathematical Physics* 109.1 (1987), pp. 109–156. DOI: 10.1007/BF01205674. URL: https://link.springer.com/article/10.1007/BF01205674.

[11] Donald A. McQuarrie. *Statistical Mechanics*. Accessed via Internet Archive. Sausalito, California: University Science Books, 2000. ISBN: 9781891389153. URL: https://archive.org/details/StatisticalMechanics_201709.

[12] Edward F. Moore. "The Shortest Path Through a Maze". In: *Proceedings of an International Symposium on the Theory of Switching, Part II*. Bell Telephone System. Technical publications. Monograph, vol. 3523. Originally published by Bell Telephone System; length: 8 pages. Cambridge, MA: Harvard University Press, 1959, pp. 285–292.

[13] Kai Nagel and Michael Schreckenberg. "A cellular automaton model for freeway traffic". In: *Journal de physique I* 2.12 (1992), pp. 2221–2229.

[14] Siddharth Patwardhan et al. "Machine Learning as an Accurate Predictor for Percolation Threshold of Diverse Networks". In: *arXiv preprint arXiv:2212.14694* (2022). DOI: 10.48550/arXiv.2212.14694. arXiv: 2212.14694 [physics.soc-ph].

[15] Ari Sihvola, Sonja Saastamoinen, and Kari Heiska. "Mixing rules and percolation". In: *Remote Sensing Reviews* 9.1-2 (1994), pp. 39–50.

[16] Dietrich Stauffer and Ammon Aharony. *Introduction to percolation theory*. Taylor & Francis, 2018.

[17] Zhi-Jie Tan et al. "Epidemic spreading in percolation worlds". In: *Physics Letters A* 300.2-3 (2002), pp. 317–323.

[18] Kenneth G. Wilson. “Renormalization Group and Critical Phenomena. I. Renormalization Group and the Kadanoff Scaling Picture”. In: *Physical Review B* 4.9 (1971), pp. 3174–3183. DOI: 10.1103/PhysRevB.4.3174. URL: https://link.aps.org/doi/10.1103/PhysRevB.4.3174.

A | Pseudocode and GitHub Repository

All of the results and figures reported in this document can be recreated using the code found in this repository.

Below we report pseudocode for Figure 3 and for the foundation of the classifier in Section III.2.

Algorithm 1 Finite-Size Scaling of 2D Percolation

Input: Lattice sizes $L \in \{16, 32, 64, 128\}$, probabilities $p \in [0.4, 0.7]$, samples $N = 1000$, $p_c = 0.59275$, $\nu = 4/3$

Output: Percolation probability $\Pi(p, L)$ and scaling collapse $\Pi((p - p_c)L^{1/\nu})$

function GENERATELATTICE(L, p)
  return Boolean grid $s \sim \text{Bernoulli}(p)^{L \times L}$
end function
function CHECKPERCOLATION(s)
  Initialize queue Q and visited grid same shape as s
  for each column j in top row **do**
    if $s[0, j]$ is occupied **then**
      Mark $(0, j)$ as visited and enqueue into Q
    end if
  end for
  while Q not empty **do**
    Dequeue (i, j) from Q
    if $i = L - 1$ **then**
      return True
    end if
    for each neighbor (i', j') of (i, j) **do**
      if (i', j') in bounds, occupied and unvisited **then**
        Mark as visited and enqueue
      end if
    end for
  end while
  return False
end function
function ESTIMATEPI(L, p, N)
  $C \leftarrow 0$
  for $k = 1$ to N **do**
    $s^{(k)} \leftarrow$ GENERATELATTICE(L, p)
    if CHECKPERCOLATION($s^{(k)}$) **then**
      $C \leftarrow C + 1$
    end if
  end for
  return C/N
end function

Algorithm 2 Pseudocode for the foundation of the classifier

Input: Block size b, coarse-graining steps T, training samples N, batch size B, epochs E
Output: Trained PercolationModel f_θ

LATTICE GENERATION
function GENERATE_LATTICE(size, p)
  return `np.random.choice`$([0,1], \text{size} = (size, size), p = [1-p, p])$
end function
PERCOLATION CHECK
function CHECK_PERCOLATION($\mathbf{s}$)
  Label connected components (4-connectivity)
  return $\mathbb{I}[\exists$ spanning cluster$]$
end function
MODEL DEFINITION
procedure PERCOLATIONMODEL
  rule $\leftarrow$ Sequential(
    Linear$(b^2 \to 64)$, ReLU(),
    Linear$(64 \to 1)$, Sigmoid()
)
end procedure
FORWARD PASS
function MODEL_FORWARD($\mathbf{s}$, T)
  while $H \geq b$ and $W \geq b$ and $t < T$ **do**
    patches $\leftarrow$ unfold$(\mathbf{s}, \text{kernel_size} = b)$
    patches $\leftarrow$ reshape$(-1, b^2)$
    $\mathbf{s}' \leftarrow f_\theta$(patches)
    $\mathbf{s} \leftarrow$ reshape$(\mathbf{s}', (B, C, H/b, W/b))$
  end while
  return squeeze$(\mathbf{s})$
end function
TRAINING
function TRAIN(N, L)
  $\mathcal{D} \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** N **do**
    $p \sim \mathcal{U}(0,1)$
    $\mathbf{s} \leftarrow$ generate_lattice(L, p)
    $y \leftarrow$ check_percolation$(\mathbf{s})$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}, y)\}$
  end for
  $\theta \leftarrow$ SGD$(f_\theta, \text{lr} = 0.001)$
  $\mathcal{L} \leftarrow$ BCELoss()
  for epoch $= 1$ **to** E **do**
    for batch $\sim \mathcal{D}$ **do**
      $\mathbf{s}_{\text{batch}}, \mathbf{y}_{\text{batch}} \leftarrow$ batch
      $\hat{\mathbf{y}} \leftarrow f_\theta(\mathbf{s}_{\text{batch}})$
      loss $\leftarrow \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}_{\text{batch}})$
      ∇_θloss $\leftarrow$ backward()
      $\theta \leftarrow \theta - \eta \nabla_\theta$loss
    end for
  end for
  return f_θ
end function

B Learned rule for fixed size training

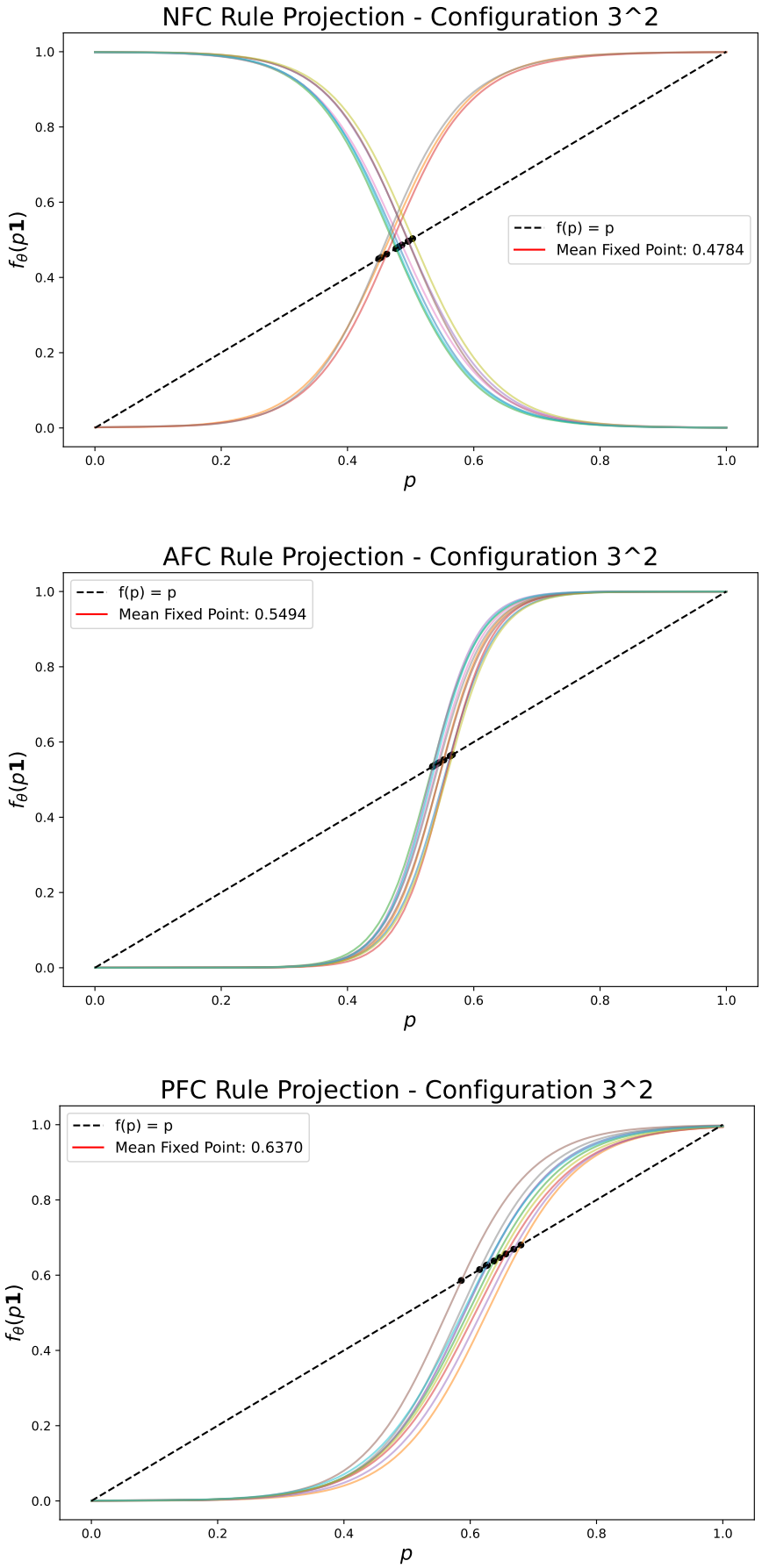
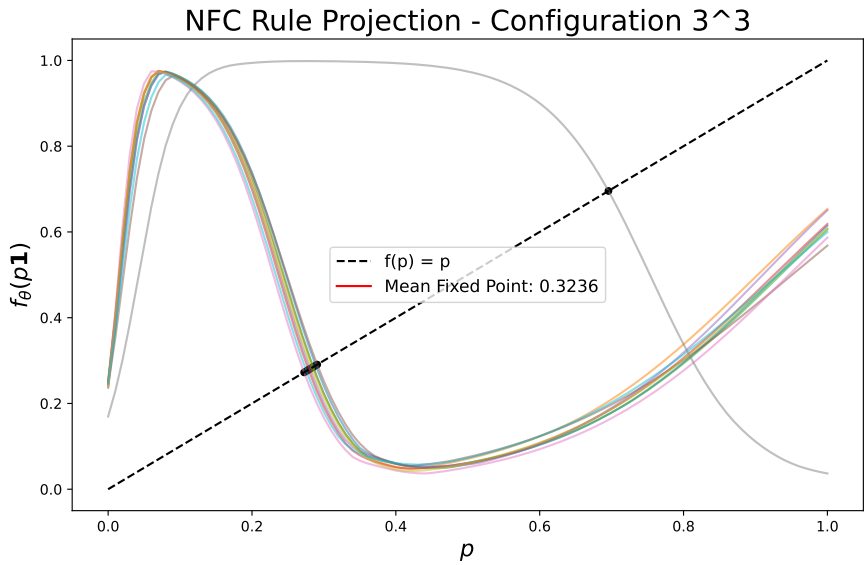
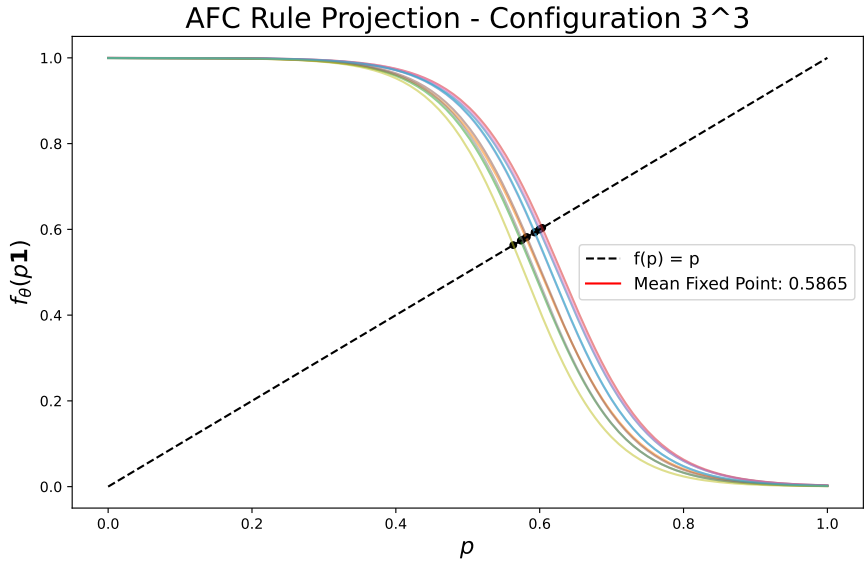


Figure 9: Projection $f_\theta(p\mathbf{1})$ for all methods and size 3^2. For each run, the number of samples is $N = 10^4$, learning rate is $\eta = 10^{-3}$, number of epochs is $E = 10$ and batch size $B = 10$.





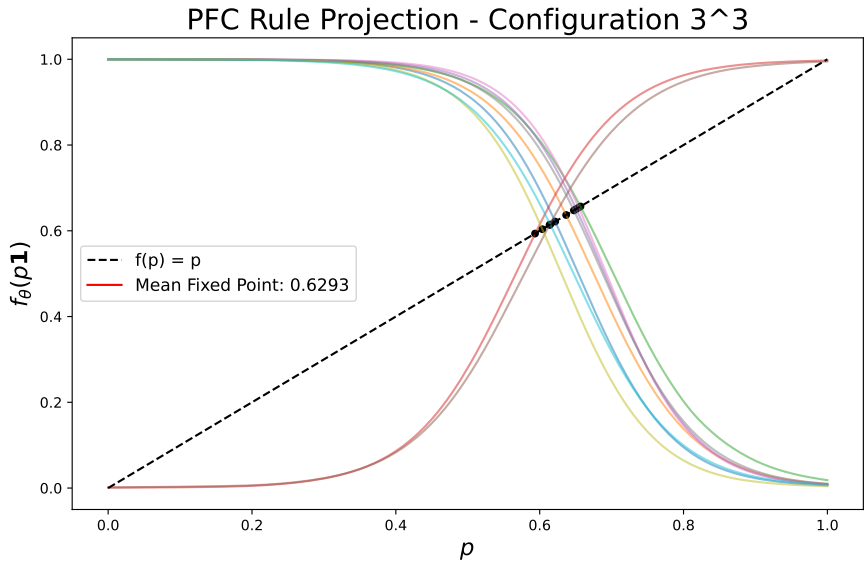


Figure 10: Projection $f_\theta(p\mathbf{1})$ for all methods and size 3^3. For each run, the number of samples is $N = 10^4$, learning rate is $\eta = 10^{-3}$, number of epochs is $E = 10$ and batch size $B = 10$.

C Train and test error

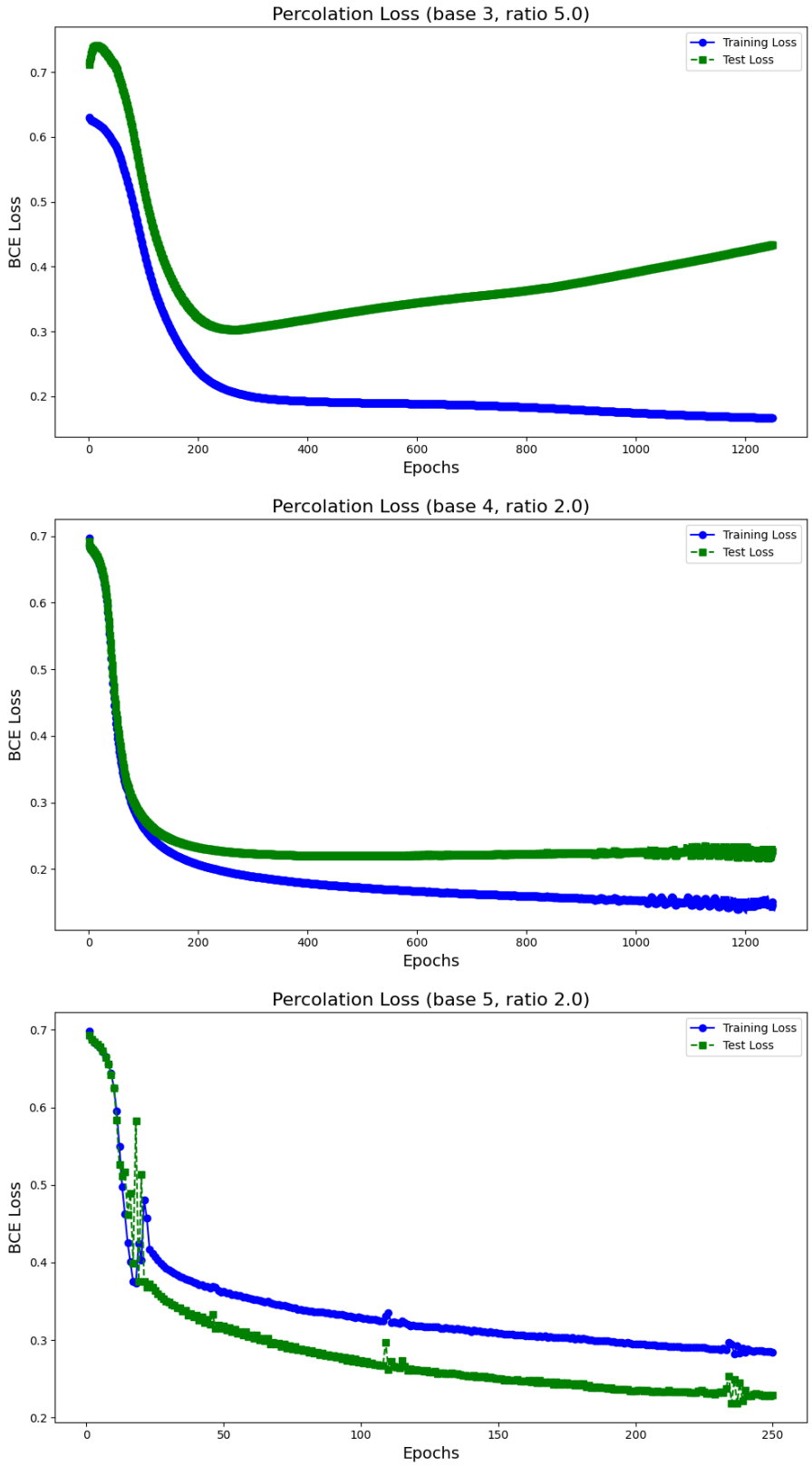


Figure 11: Train and test error as a function of the number of epochs, for $b \in \{3, 4, 5\}$. One ratio $|\theta|/N$ is shown per base b.

D | Recursive approach for $\Pi(p, n, m)$ with $m = 3$

Let Y_n be the random variable "the lattice percolates up to layer n". Such a layer can present itself in 2^3 different configurations, represented by an index $i \in \{1, \dots, 8\}$ (Figure 12).

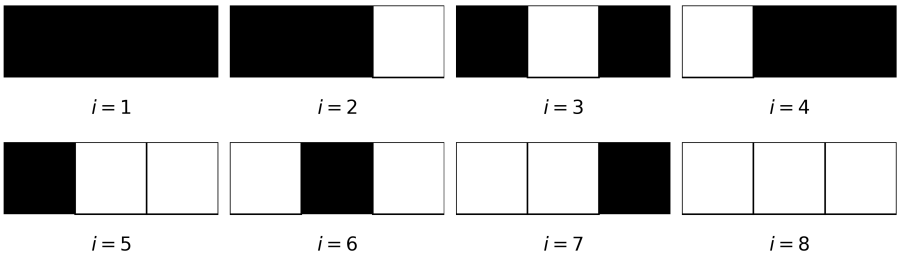


Figure 12: The 8 configurations of a three-cell lattice.

We can exploit them to establish the equation

$$\Pi(p, n, 3) = \mathbb{P}(Y_n = 1) = \sum_{i=1}^{8} \mathbb{P}(Y_n|i)\mathbb{P}(i)$$

For the sake of simplicity, we can rewrite it by defining $f_n \equiv \mathbb{P}(Y_n = 1)$ and $g_n^{(i)} \equiv \mathbb{P}(Y_n = 1|i)$ to write

$$f_n = p^3 g_n^{(1)} + p^2(1-p)(g_n^{(3)} + g_n^{(4)} + g_n^{(5)}) + p(1-p)^2(g_n^{(5)} + g_n^{(6)} + g_n^{(7)})$$

as $g_n^{(8)} = \mathbb{P}(Y_n = 1|i = 8) = 0$. The probabilities $\{g_n^{(i)}\}_{i=1}^8$ satisfy the following recursive system of equations

$$\mathbf{g}_n = \mathbf{M}\mathbf{g}_{n-1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \mathbf{g}_{n-1} = \mathbf{M}^n \mathbf{g}_1$$

with the initial conditions

$$g_1^{(i)} = p^{n_i}(1-p)^{3-n_i} \quad ; \quad n_i = \sum_{j=1}^{3} X_{ij}\,.$$

The matrix power $\mathbf{M}^n$ can be computed in $\mathcal{O}(\log n)$ multiplications, each taking $\mathcal{O}(n^3)$ operations, although the matrix $\mathbf{M}$ itself requires $\mathcal{O}(2^m)$ operations.