Using Machine Learning to Classify Phase Transitions

Morten Hjorth-Jensen[1]

Department of Physics and Center for Computing in Science Education, University of Oslo, Norway[1]

CEACM Flagship School: Machine Learning in Physical Sciences: Theory and Applications, May 26-30, 2025

What is this about?

These notes, with pertinent exercises cover the following topics.

1. Phase Transitions & Critical Phenomena: Definitions and key concepts (order parameters, critical points, first vs second order).
2. Spin Models: 2D Ising model and the q-state Potts model (examples of phase transitions).
3. Data Generation: Monte Carlo simulations for sampling spin configurations across temperatures.
4. Unsupervised Learning (PCA): Principal Component Analysis to visualize phase separation without labels.
5. Supervised Learning (CNN): Convolutional Neural Networks for classifying phases from raw configurations.
6. Generative Models (VAE): Variational Autoencoders for latent representation learning and critical anomaly detection.
7. Comparisons: Interpretability and performance trade-offs between PCA, CNN, and VAE.

Where do I find the material?

All the material here can be found in the PDF files, codes and jupyter-notebooks at the above **doc** folder, see the **pub** subfolder, link to be added

AI/ML and some statements you may have heard (and what do they mean?)

1. Fei-Fei Li on ImageNet: **map out the entire world of objects** (The data that transformed AI research)
2. Russell and Norvig in their popular textbook: **relevant to any intellectual task; it is truly a universal field** (Artificial Intelligence, A modern approach)
3. Woody Bledsoe puts it more bluntly: **in the long run, AI is the only science** (quoted in Pamilla McCorduck, Machines who think)

If you wish to have a critical read on AI/ML from a societal point of view, see Kate Crawford's recent text Atlas of AI.

Here: with AI/ML we intend a collection of machine learning methods with an emphasis on statistical learning and data analysis

Types of machine learning

The approaches to machine learning are many, but are often split into two main categories. In *supervised learning* we know the answer to a problem, and let the computer deduce the logic behind it. On the other hand, *unsupervised learning* is a method for finding patterns and relationship in data sets without any prior knowledge of the system.

An important third category is *reinforcement learning*. This is a paradigm of learning inspired by behavioural psychology, where learning is achieved by trial-and-error, solely from rewards and punishment.

Main categories

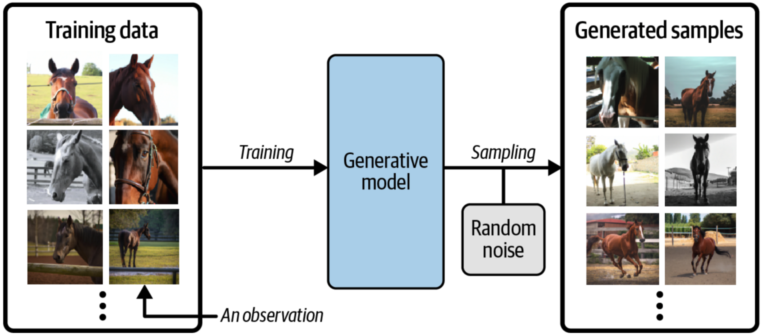
Another way to categorize machine learning tasks is to consider the desired output of a system. Some of the most common tasks are:

- Classification: Outputs are divided into two or more classes. The goal is to produce a model that assigns inputs into one of these classes. An example is to identify digits based on pictures of hand-written ones. Classification is typically supervised learning.
- Regression: Finding a functional relationship between an input data set and a reference data set. The goal is to construct a function that maps input data to continuous output values.
- Clustering: Data are divided into groups with certain common traits, without knowing the different groups beforehand. It is thus a form of unsupervised learning.

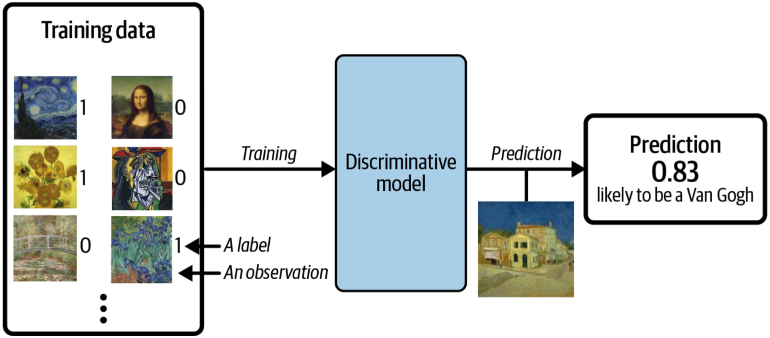
The plethora of machine learning algorithms/methods

1. Deep learning: Neural Networks (NN), Convolutional NN, Recurrent NN, Boltzmann machines, autoencoders and variational autoencoders and generative adversarial networks, stable diffusion and many more generative models
2. Bayesian statistics and Bayesian Machine Learning, Bayesian experimental design, Bayesian Regression models, Bayesian neural networks, Gaussian processes and much more
3. Dimensionality reduction (Principal component analysis), Clustering Methods and more
4. Ensemble Methods, Random forests, bagging and voting methods, gradient boosting approaches
5. Linear and logistic regression, Kernel methods, support vector machines and more
6. Reinforcement Learning; Transfer Learning and more

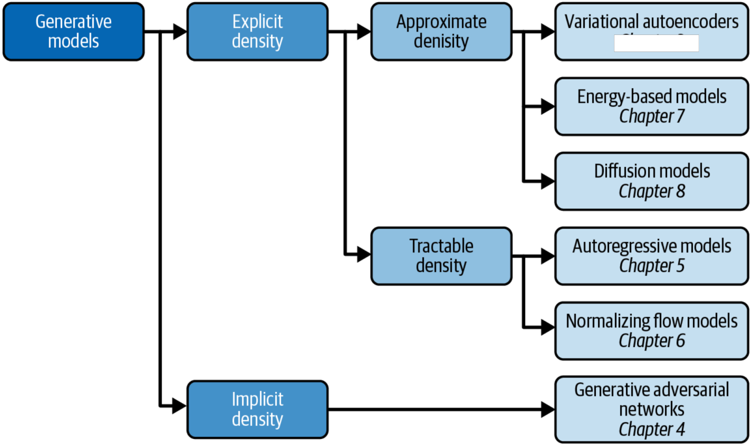
Example of generative modeling, taken from Generative Deep Learning by David Foster



Example of discriminative modeling, taken from Generative Deeep Learning by David Foster



Taxonomy of generative deep learning, taken from Generative Deep Learning by David Foster



Good books with hands-on material and codes

- Sebastian Rashcka et al, Machine learning with Sickit-Learn and PyTorch
- David Foster, Generative Deep Learning with TensorFlow
- Bali and Gavras, Generative AI with Python and TensorFlow 2

All three books have GitHub addresses from where one can download all codes. We will borrow most of the material from these three texts as well as from Goodfellow, Bengio and Courville's text Deep Learning

What are the basic Machine Learning ingredients?

Almost every problem in ML and data science starts with the same ingredients:

- The dataset $\boldsymbol{x}$ (could be some observable quantity of the system we are studying)
- A model which is a function of a set of parameters $\boldsymbol{\alpha}$ that relates to the dataset, say a likelihood function $p(\boldsymbol{x}|\boldsymbol{\alpha})$ or just a simple model $f(\boldsymbol{\alpha})$
- A so-called **loss/cost/risk** function $\mathcal{C}(\boldsymbol{x}, f(\boldsymbol{\alpha}))$ which allows us to decide how well our model represents the dataset.

We seek to minimize the function $\mathcal{C}(\boldsymbol{x}, f(\boldsymbol{\alpha}))$ by finding the parameter values which minimize $\mathcal{C}$. This leads to various minimization algorithms. It may surprise many, but at the heart of all machine learning algortihms there is an optimization problem.

Low-level machine learning, the family of ordinary least squares methods

Our data which we want to apply a machine learning method on, consist of a set of inputs $\boldsymbol{x}^T = [x_0, x_1, x_2, \dots, x_{n-1}]$ and the outputs we want to model $\boldsymbol{y}^T = [y_0, y_1, y_2, \dots, y_{n-1}]$. We assume that the output data can be represented (for a regression case) by a continuous function f through

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon.$$

Setting up the equations

In linear regression we approximate the unknown function with another continuous function $\boldsymbol{\tilde{y}}(\boldsymbol{x})$ which depends linearly on some unknown parameters $\boldsymbol{\theta}^T = [\theta_0, \theta_1, \theta_2, \dots, \theta_{p-1}]$.
The input data can be organized in terms of a so-called design matrix with an approximating function $\boldsymbol{\tilde{y}}$

$$\boldsymbol{\tilde{y}} = \boldsymbol{X}\boldsymbol{\theta},$$

The objective/cost/loss function

The simplest approach is the mean squared error

$$C(\boldsymbol{\Theta})=\frac{1}{n}\sum_{i=0}^{n-1}\left(y_i-\tilde{y}_i\right)^2=\frac{1}{n}\left\{\left(\boldsymbol{y}-\tilde{\boldsymbol{y}}\right)^T\left(\boldsymbol{y}-\tilde{\boldsymbol{y}}\right)\right\},$$

or using the matrix $\boldsymbol{X}$ and in a more compact matrix-vector notation as

$$C(\boldsymbol{\Theta})=\frac{1}{n}\left\{\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\right)^T\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\theta}\right)\right\}.$$

This function represents one of many possible ways to define the so-called cost function.

Training solution

Optimizing with respect to the unknown parameters θ_j we get

$$\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta},$$

and if the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is invertible we have the optimal values

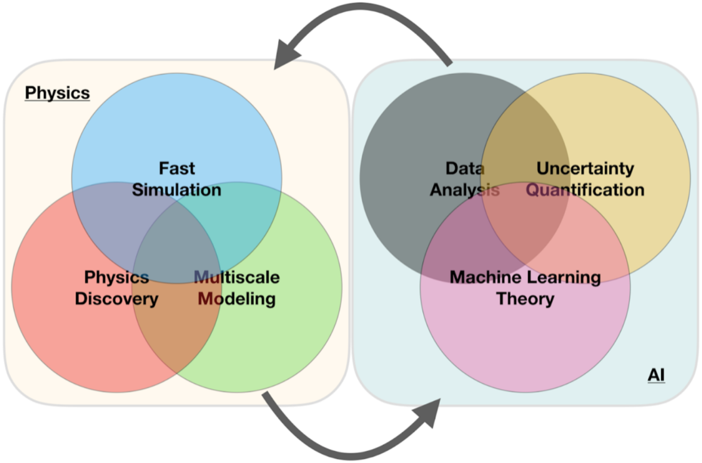
$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

We say we 'learn' the unknown parameters $\boldsymbol{\theta}$ from the last equation.

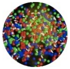
Selected references

- Mehta et al. and Physics Reports (2019).
- Machine Learning and the Physical Sciences by Carleo et al
- Artificial Intelligence and Machine Learning in Nuclear Physics, Amber Boehnlein et al., Reviews Modern of Physics 94, 031003 (2022)
- Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research 5, 033062 (2023)
- Neural-network quantum states for ultra-cold Fermi gases, Jane Kim et al, Nature Physics Communcication, submitted
- Message-Passing Neural Quantum States for the Homogeneous Electron Gas, Gabriel Pescia, Jane Kim et al. arXiv.2305.07240,
- "Efficient solutions of fermionic systems using artificial neural networks, Nordhagen et al, Frontiers in Physics

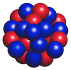
Machine learning. A simple perspective on the interface between ML and Physics

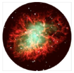


ML in Nuclear Physics (or any field in physics)

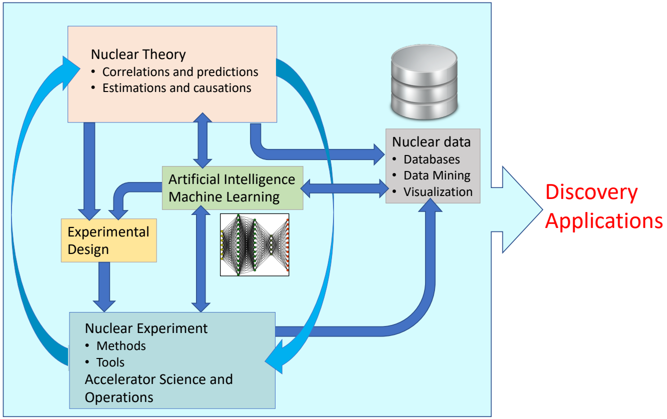

Hot and Dense Nuclear Matter

Hadrons


Atomic Nucleus


Nuclei in the Cosmos

Fundamental interactions



Phase Transitions and Critical Phenomena

1. Definition: A phase transition is characterized by an abrupt, non-analytic change in a macroscopic property of a system as some external parameter (e.g. temperature) is varied . In simpler terms, the system's state or phase changes dramatically at a critical point.
2. Order Parameter: Associated with each phase transition is an order parameter – a quantity that is zero in one phase and non-zero in the other. For example, magnetization plays the role of an order parameter in magnetic systems, distinguishing ordered (magnetized) from disordered (unmagnetized) phases.
3. Critical Point: At the critical temperature (or pressure, etc.), the order parameter changes (continuous or discontinuous) and the system exhibits critical phenomena: large fluctuations, divergence of correlation length, and the onset of scale invariance. Critical points of second-order transitions feature continuous change of the order parameter with characteristic critical exponents and universal behavior across different systems.

Phase Transitions: Definitions

1. A *phase transition* is a qualitative change in the state of a system when a control parameter (e.g. temperature) passes a critical point.
2. **Order parameter**: quantity that distinguishes phases (e.g. magnetization M for magnetic systems).
3. **Order vs disorder**: e.g. below T_c a ferromagnet has $|M| > 0$ (ordered), above T_c $M = 0$ (disordered).
4. Phases can break symmetries; transitions can be *continuous* (second-order) or *first-order*.

Order Parameter and Symmetry Breaking

1. Phase transitions often involve spontaneous symmetry breaking (e.g. Ising model Z_2 symmetry).
2. The order parameter (e.g. magnetization $M = \frac{1}{N} \sum_i s_i$) changes behavior at T_c.
3. In ferromagnets: $M = 0$ for $T > T_c$ (symmetric paramagnet), $M \neq 0$ for $T < T_c$ (broken symmetry).
4. Example: in 2D Ising model, two symmetric ordered states (up/down) below T_c.

Critical Phenomena and Scaling

1. Near a continuous transition, observables follow power laws: $M \sim |T - T_c|^{\beta}$, correlation length $\xi \sim |T - T_c|^{-\nu}$, etc.
2. **Critical exponents** $(\alpha, \beta, \gamma, \nu, \dots)$ characterize singular behavior.
3. Universality: systems with the same symmetry and dimension share exponents.
4. The classical example is the two-dimensional Ising exponents known analytically (Onsager).
5. At $T \to T_c$, correlation length $\xi \to \infty$, large-scale fluctuations appear.

2D Ising Model: Definition

1. Spins $s_i = \pm 1$ on a 2D square lattice, nearest-neighbor ferromagnetic coupling.
2. Hamiltonian: $H = -J \sum_{\langle i,j \rangle} s_i s_j$, with $J > 0$ favoring alignment.
3. Exhibits a second-order phase transition at critical temperature T_c.
4. Order parameter: magnetization $M = \frac{1}{N} \sum_i s_i$.
5. Below T_c, $M \neq 0$ (ferromagnetic order); above T_c, $M = 0$ (paramagnet).

2D Ising Model: Critical Temperature

1. Exact result (Onsager): critical point T_c satisfies $T_c \approx \frac{2J}{\ln(1+\sqrt{2})} \approx 2.269J$.
2. At $T > T_c$: spins are mostly disordered, no net magnetization.
3. At $T < T_c$: long-range order develops (nonzero M).
4. Correlation length ξ diverges at T_c
5. Example: at $T = T_c$ large clusters of aligned spins appear.

q-State Potts Model: Definition

- Generalization of Ising: each spin $s_i \in \{1, 2, \ldots, q\}$.
- Ferromagnetic Potts Hamiltonian:

$$H = -J \sum_{\langle i,j \rangle} \delta_{s_i,s_j},$$

where $\delta_{a,b} = 1$ if $a = b$, else 0.

- If $q = 2$, reduces to the Ising model. Higher q allows richer symmetry breaking ($\mathbb{Z}_q$).
- Widely used to study phase transitions with multiple equivalent ordered states.

2D Potts Model: Phase Behavior

- In 2D, the ferromagnetic Potts model has a phase transition for all $q \geq 1$
- Exact critical point:

$$\frac{J}{k_B T_c} = \ln\bigl(1 + \sqrt{q}\bigr).$$

- The nature of the transition depends on q
  - $1 \leq q \leq 4$: continuous (second-order) transition.
  - $q > 4$: discontinuous (first-order) transition (latent heat appears).
- Example: $q = 3, 4$ have continuous transitions; $q = 5$ and higher show first-order behavior.

Monte Carlo Sampling of Spin Models

1. Use Monte Carlo (MC) to generate spin configurations at given T: sample from Boltzmann distribution $P \propto e^{-H/T}$.
2. Metropolis algorithm: attempt random single-spin flips to equilibrate the system.
3. Provides training data: spin configurations $\{s_i\}$ labeled by temperature or phase.
4. Ensures statistical equilibrium and detailed balance
5. Efficient sampling (especially near T_c cluster algorithms help, e.g. Wolff or Swendsen-Wang).

Metropolis Algorithm

- Initialize spins randomly or in a fixed state.
- Repeat for many steps:
  1. Pick a random lattice site i.
  2. Propose flipping $s_i \to -s_i$ (Ising) or change state (Potts).
  3. Compute energy change ΔE.
  4. If $\Delta E \leq 0$, accept the flip (lower energy).
  5. Else accept with probability $\exp(-\Delta E/T)$ (Boltzmann factor)
  6. Otherwise, reject and keep the old state.
- After equilibration, record configurations as samples.

Metropolis Algorithm (Pseudo-code)

```
for T in temperature_list:
   # Initialize lattice (e.g., random spins)
   config = random_configuration(Lx, Ly)
   for step in range(num_steps):
       i,j = random_site()
       dE = compute_deltaE(config, i, j)  # energy change if spin flip
       if dE <= 0 or rand() < exp(-dE/T):
           flip_spin(config, i, j)
   record_configuration(config, T)
```

Monte Carlo Data for ML

1. Generate many spin configurations across a range of temperatures T.
2. Label each configuration by its temperature or by phase (ordered/disordered).
3. This labeled dataset is used for *supervised* methods (e.g. CNN).
4. For *unsupervised* methods (PCA, VAE), labels are not used in training.
5. Data augmentation: one can use symmetries (e.g. spin flip) to enlarge dataset.

Principal Component Analysis (PCA) Basics

1. PCA is an unsupervised method for dimensionality reduction.
2. Finds orthogonal directions (principal components) of maximum variance in data.
3. Project data onto the first few PCs to visualize structure.
4. Advantages: linear, fast, and interpretable (PCs are linear combinations of features).
5. Disadvantage: only captures linear correlations (may miss complex features).

PCA for Phase Identification

1. Apply PCA to the ensemble of spin configurations (flattened to vectors).
2. The first principal component (PC1) often correlates with the order parameter (e.g. magnetization).
3. Hu et al. (2017) found PCA distinguishes different phases and can locate critical points
4. By plotting data in the subspace of PCs, one sees separation of low-T (ordered) vs high-T (disordered) points.
5. No labels needed: phase transitions are revealed by clustering in PC space

PCA Workflow for Spin Data

1. Collect data matrix X of shape (num_samples) $\times$ (num_features), e.g. $N \times (L \times L)$.
2. Subtract the mean from each column (feature) of X.
3. Compute covariance matrix $C = X^T X$ (or use SVD on X directly).
4. Obtain eigenvalues/vectors of C: $C = U \Lambda U^T$. Columns of U are principal directions.
5. Sort by eigenvalues (variance). Project X onto top k PCs: $X_{\mathrm{red}} = X\, U[:, 1:k]$.
6. Analyze X_{red}: e.g. scatter plot PC1 vs PC2.

PCA Example: Ising Model

1. In the 2D Ising model, PC1 is essentially proportional to the overall magnetization.
2. At $T < T_c$, configurations cluster with large positive or negative PC1 (ordered states).
3. At $T > T_c$, configurations cluster near $PC1 \approx 0$ (disordered).
4. The variance captured by PC1 drops sharply at T_c, signaling the transition.
5. PCA automatically finds these features, without knowing the physics a priori.

PCA Limitations

1. PCA is linear: complex nonlinear features (e.g. vortex order) may not be captured.
2. Example: In a frustrated 2D spin model, PCA failed to detect certain correlations (vorticity)
3. PCA does not directly classify; it provides features for clustering or visualization.
4. Sensitive to scaling: data should be normalized appropriately.
5. Still useful as a first-pass: identifies the most significant variations

PCA with PyTorch (Example Code)

```
import torch

# X: tensor of shape (N, L*L) containing spin configurations as floats
# Center the data
X = X - X.mean(dim=0, keepdim=True)

# Compute covariance (or use torch.pca_lowrank)
cov = torch.mm(X.t(), X) / (X.size(0)-1)

# Eigen-decomposition (SVD) of covariance
U, S, V = torch.svd(cov)

# Select first k principal components
k = 2
PCs = U[:, :k]  # shape (L*L, k)

# Project data onto principal components
X_reduced = torch.mm(X, PCs)  # shape (N, k)
```

Convolutional Neural Networks (CNNs)

1. CNNs are deep neural networks designed for spatial data (e.g. images).
2. Architecture: convolutional layers (feature detectors) + pooling, followed by fully connected layers.
3. In physics: treat spin lattice as an image with multiple channels (e.g. one channel of spins).
4. CNNs can learn complex nonlinear features automatically from data.
5. They require labeled examples for training (supervised learning).

CNN for Phase Classification

1. Prepare training data: spin configurations labeled by phase or temperature.
2. CNN learns to map configuration $\rightarrow$ phase label (ordered/disordered) or predict T.
3. As shown by Carrasquilla and Melko (2017), CNNs can identify phases from raw states
4. Achieves high accuracy on Ising and other models when training labels are available.
5. CNNs exploit locality: can detect clusters or domains of aligned spins via convolution filters.

Example CNN Architecture

1. **Input**: single-channel $L \times L$ lattice (values -1 or $+1$).
2. **Conv layer 1**: e.g. 8 filters of size 3×3, ReLU activation, stride=1, padding=1.
3. **Conv layer 2**: 16 filters of size 3×3, ReLU, followed by a 2×2 max-pooling.
4. **Fully Connected**: flatten feature maps to vector; FC layer to 64 units (ReLU); final FC to 2 outputs (softmax for binary phase).
5. **Training**: minimize cross-entropy loss between predicted and true labels.
6. **Note**: architecture and hyperparameters can be tuned for best performance.

CNN: Training and Results

1. Train on many labeled samples (e.g. temperatures T and whether $T < T_c$ or $T > T_c$).
2. The network learns features such as magnetization domains, energy patterns, etc.
3. CNN accuracy can be very high (often $\sim$100% on clean data) for distinguishing phases.
4. Fukushima and Sakai (2021): a CNN trained on 2D Ising can detect transition in q-state Potts
5. CNN behavior: at high T it effectively uses average energy; at low T it correlates with magnetization

CNN Interpretability

1. CNNs are often seen as **black boxes**, but their learned filters can sometimes be interpreted.
2. Outputs correlate with known physics:
    2.1 At low T: classification heavily influenced by magnetization (order).
    2.2 At high T: classification influenced by internal energy (disorder)
3. CNNs can generalize: e.g. Ising-trained CNN finds Potts T_c
4. Visualization methods (e.g. saliency maps) can highlight what CNN focuses on.

CNN (PyTorch) Code Example

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class PhaseCNN(nn.Module):
   def __init__(self, L):
       super(PhaseCNN, self).__init__()
       self.conv1 = nn.Conv2d(1, 8, kernel_size=3, padding=1)  # 1 cha
       self.conv2 = nn.Conv2d(8, 16, kernel_size=3, padding=1) # 8 ->
       self.pool = nn.MaxPool2d(2)  # downsample by 2
       self.fc1 = nn.Linear(16 * (L//2) * (L//2), 64)
       self.fc2 = nn.Linear(64, 2)  # 2 output classes

   def forward(self, x):
       x = F.relu(self.conv1(x))      # (B,8,L,L)
       x = self.pool(F.relu(self.conv2(x)))  # (B,16,L/2,L/2)
       x = x.view(x.size(0), -1)      # flatten
       x = F.relu(self.fc1(x))
       x = self.fc2(x)                # logits for 2 classes
       return x

# Example usage:
model = PhaseCNN(L=32)            # for a 32x32 lattice
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
```

Variational Autoencoders (VAE) Overview

1. A VAE is an *unsupervised* generative model that learns a latent representation of data.
2. Components:
    2.1 **Encoder**: maps input X to parameters $(\mu, \log \sigma^2)$ of a latent Gaussian.
    2.2 **Latent** z: sampled via $z = \mu + \sigma\epsilon$ ($\epsilon \sim N(0, I)$).
    2.3 **Decoder**: reconstructs input $\hat{X}$ from z.
3. Loss: reconstruction error + KL divergence to enforce latent prior $\mathcal{N}(0, I)$.
4. VAEs can both encode data compactly and generate new samples by sampling z.

VAE for Spin Configurations

1. Train VAE on spin configurations (no labels).
2. Latent space (usually low-dimensional) captures key features (like order parameter).
3. Walker et al. (2020): latent variables provide metrics to track order vs disorder in Ising
4. They found the latent representation closely corresponds to physical order (magnetization)
5. After training, one can:
   - 5.1 Inspect latent space (e.g. scatter plot of (μ_1, μ_2)) to distinguish phases.
   - 5.2 Sample $z \sim N(0, 1)$ and decode to generate synthetic configurations.

VAE Architecture Details

- Typically use convolutional encoder/decoder for 2D structure.
- Example:
  1. Encoder: conv layers downsampling to a flat vector $\rightarrow$ linear layers $\rightarrow (\mu, \log\sigma^2)$ (size of latent space, e.g. 2–10 dims).
  2. Decoder: linear layer from z to feature map size, followed by transposed-conv layers to reconstruct $L \times L$ lattice.
- Activation: ReLU (or LeakyReLU); final output often sigmoid to model spin distribution.
- Training with minibatch gradient descent optimizing

$$\mathcal{L} = \mathbb{E}[\|X - \hat{X}\|^2] + \mathrm{KL}(\mathcal{N}(\mu, \sigma) \,\|\, \mathcal{N}(0, 1)).$$

VAE Results on Ising Model

1. The first latent dimension (ν_0) learned by the VAE correlated strongly with magnetization
2. Plotting ν_0 vs temperature shows clear change around T_c (order–disorder).
3. This means VAE "discovered" the order parameter without supervision.
4. The VAE predicted the critical region and crossover consistently with theory
5. Latent space clustering: ordered-phase points separate from disordered.

VAE: Generation and Interpretation

1. After training, sample random z from Gaussian prior and decode to generate configurations.
2. The VAE latent space is continuous: can interpolate between phases.
3. The learned representation is smooth and disentangled: one latent coordinate tracks magnetization, others track disorder.
4. VAEs can also be used for anomaly detection: points with unusual z indicate atypical states.
5. Overall, VAEs provide both a dimensionally-reduced view of phase structure and a generative model.

VAE (PyTorch) Code Example

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class VAE(nn.Module):
   def __init__(self, L, latent_dim=2):
       super(VAE, self).__init__()
       # Encoder: conv -> conv -> flatten -> fc_mu/fc_logvar
       self.encoder = nn.Sequential(
           nn.Conv2d(1, 8, 3, stride=2, padding=1),   # -> (8, L/2, L/
           nn.ReLU(),
           nn.Conv2d(8, 16, 3, stride=2, padding=1),  # -> (16, L/4, L
           nn.ReLU(),
           nn.Flatten()
       )
       self.fc_mu = nn.Linear(16*(L//4)*(L//4), latent_dim)
       self.fc_logvar = nn.Linear(16*(L//4)*(L//4), latent_dim)

       # Decoder: linear -> unflatten -> convTranspose -> convTranspos
       self.decoder_fc = nn.Linear(latent_dim, 16*(L//4)*(L//4))
       self.decoder = nn.Sequential(
           nn.Unflatten(1, (16, L//4, L//4)),
           nn.ConvTranspose2d(16, 8, 3, stride=2, padding=1, output_pa
           nn.ReLU(),
           nn.ConvTranspose2d(8, 1, 3, stride=2, padding=1, output_pad
           nn.Sigmoid()
       )
```

Supervised vs Unsupervised Methods

1. **Supervised (CNN)**: Requires labeled data (phase labels or temperatures). Learns a direct mapping $\{\text{config}\} \to \{\text{phase}\}$.
2. **Unsupervised (PCA, VAE)**: Uses only the raw configurations without labels. Learns features or representations of the data.
3. PCA reduces dimensionality; requires no training labels
4. VAE learns a latent generative model; also label-free
5. CNN typically achieves higher accuracy in classifying known phases, but needs supervised labels.

Method Interpretability and Features

1. **PCA**: Principal components often have clear physical meaning (e.g. PC1 and magnetization)
2. **CNN**: Filters are less directly interpretable; features are learned. However, some correlation with physics (energy, M) emerges
3. **VAE**: Latent variables can often be interpreted as order/disorder features (e.g. one latent is approximately equal to the magnetization)
4. CNN is a "black box" classifier; PCA/VAE provide insight into data structure.
5. In terms of visualization: PCA and VAE produce low-dim plots of data (semi-transparent), whereas CNN only outputs a decision boundary.

Performance and Use Cases

1. **PCA**: Fast to compute; good for preliminary analysis of large datasets. Best for linearizable transitions.
2. **CNN**: High classification accuracy; powerful for large and complex datasets. Can predict critical T or classify multiple phases
3. **VAE**: Useful when no labels are available; provides a generative model. Effective in detecting transitions by latent statistics
4. Computational cost: PCA very cheap, CNN and VAE require training time (GPU recommended for large data).
5. Choosing a method: depends on data availability and goal (classification vs insight vs generation).

Summary of Methods

1. **PCA**: Unsupervised, linear, interpretable. Good for dimensionality reduction and initial exploration
2. **CNN**: Supervised, non-linear, high accuracy. Requires labels, but learns complex features (works across models
3. **VAE**: Unsupervised, generative. Learns latent representation reflecting order/disorder
4. Each method has trade-offs in accuracy, interpretability, and data requirements.
5. Combining methods (e.g. using PCA or VAE features as input to another classifier) can also be fruitful.

Conclusions

1. Machine learning provides powerful tools for studying phase transitions in statistical models.
2. *Unsupervised* methods (PCA, VAE) can discover phase structure without labels
3. *Supervised* methods (CNNs) achieve high classification performance given labeled data
4. Interpretability: PCA/VAE offer more insight into physics (latent/PC represent order parameters), while CNNs focus on prediction accuracy.
5. Choice of method depends on the problem: data availability, need for generative modeling, and interpretability.
6. Future directions: deeper architectures (e.g. ResNets), unsupervised generative flows, transfer learning across models, real experimental data.

References

1. Carrasquilla, J. & Melko, R. G. (2017). Machine learning phases of matter. *Nature Physics*, 13, 431–434
2. Hu, W. *et al.* (2017). Discovering phases, phase transitions through unsupervised ML. *Phys. Rev. E* 95, 062122
3. Fukushima, K. & Sakai, K. (2021). Can a CNN trained on Ising detect Potts? *Prog. Theor. Exp. Phys.* 2021, 061A01
4. Walker, N. *et al.* (2020). 2D Ising model crossover via VAE. *Sci. Rep.* 10, 13047
5. Add refs