README

Joseph Hadley

November 2025

1 Introduction

This is a modular, object-oriented program to run lattice simulations

2 Classes

- Simulation
- Lattice
- Action
- UpdateProposer
- Observer
- ReaderWriter

The classes Lattice, Action, and UpdateProposer are designed to contain multiple different modular pieces which are governed by Simulation, and can be slotted in and out. Observer has an internal list of observables that can be calculated. ReaderWriter may as well be part of Simulation really.

The file main.py is where I work, and then anything I want to save I put in the folder "runs".

3 UpdateProposer

The idea here is that we have multiple ways to suggest and carry out updates. All UpdateProposers have an update() abstract method and updateCycle().

Many of these use lazy initialisation, where details from the parent Simulation and lattice are supplied on first use of the updateCycle() method.

The Observer class has a chance to calculate and record observables after each updateCycle.

Currently available Update Proposers are given below:

3.1 VAEProposer

The modification to allow the decoder to see the initial input is done using the parameter double_input. This modifies the decoder to include the input.

3.1.1 VAE Notes

There are multiple ways to define the loss function. Which usually has the form:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}} \tag{1}$$

Recon refers to the reconstruction loss, which is the negative log-likelihood of the input under the decoder distribution:

$$\mathcal{L}_{\text{recon}} = -E_{q\phi(z|x)}[\log p_\theta(x|z)] \tag{2}$$

The exact form of this expression depends on the assumed likelihood model $p_\theta(x|z)$

KL refers to the Kullback Liebler divergence. Here it's a measure of how closely the latent space matches a multidimensional unit Gaussian. It is the term responsible for arranging the latent space like this.

$$D_{\text{KL}} = (P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx \tag{3}$$

With a Gaussian prior we end up with the KL Divergence, denoted D_{KL}:

$$D_{\text{KL}} = -\frac{1}{2} \sum_{d=1}^{D} \left(1 + \log\left(\sigma^2\right) - \mu^2 - \sigma^2\right) \tag{4}$$

We calculate this for each x_i in a batch $\{x_1, \ldots, x_B\}$, then aggregate across the batch by summing or taking the mean. This is called a reduction.

3.1.2 VAE Acceptance Probability

It's nontrivial to get the acceptance probability of a move from one state to another in a simulation where a VAE is providing the proposed update.

In a periodic ring of N lattice sites we choose a window of M sites to be updated. We place the window at a random choice of the N possible positions. Sites outside the window will not be updated.

The transition probability from state ϕ to state ϕ' is therefore:

$$\begin{aligned} p\left(\phi'_1 \ldots \phi'_N | \phi_1 \ldots \phi_N\right) = & \frac{1}{N} p\left(\phi'_1 \ldots \phi'_M | \phi_1 \ldots \phi_M\right) \left(\delta\left(\phi'_{M+1}\phi_{M+1}\right) \ldots \delta\left(\phi'_N \phi_N\right)\right) \\ & + \frac{1}{N} p\left(\phi'_2 \ldots \phi'_{M+1} | \phi_2 \ldots \phi_{M+1}\right) \left(\delta\left(\phi'_{M+2}\phi_{M+2}\right) \ldots \delta\left(\phi'_N \phi_N\right)\right) \left(\delta\left(\phi'_1 \phi_1\right)\right) \\ & + \ldots \end{aligned} \tag{5}$$

Assume we choose to update $\phi_i \ldots \phi_{i+M}$

In this ratio, required for detailed balance:

$$\frac{p\left(\phi'_1 \dots \phi'_N \mid \phi_1 \dots \phi_N\right)}{p\left(\phi_1 \dots \phi_N \mid \phi'_1 \dots \phi'_N\right)}$$

Only the terms with $p\left(\phi_i \dots \phi_{i+M} \mid \phi'_i \dots \phi'_{i+M}\right)$ contribute, since they are within the updated window:

$$\frac{p\left(\phi'_1 \dots \phi'_N \mid \phi_1 \dots \phi_N\right)}{p\left(\phi_1 \dots \phi_N \mid \phi'_1 \dots \phi'_N\right)} = \frac{p\left(\phi'_i \dots \phi'_{i+M} \mid \phi_i \dots \phi_{i+M}\right)}{p\left(\phi_i \dots \phi_{i+M} \mid \phi'_i \dots \phi'_{i+M}\right)}$$

Concentrate on those sites which are inside the window, for this case: $\phi_1 \dots \phi_M$. These are the sites that will be updated by the variational autoencoder.

3.2 ToyMVAEProposer

Reproduces the heatbath update, as if the Modified VAE had been trained exactly right for that situation. runs/TOYMVAE_test_vs_Heatbath.py confirms this.

3.3 HeatbathProposer

update() updates a site by choosing its value from a Gaussian distribution:

$$\phi'_i \sim \mathcal{N}\left(\mu_{\mathrm{HB}}, \sigma^2_{\mathrm{HB}}\right) \tag{6}$$

$$\mu_{\mathrm{HB}} = \frac{\sum_{\mathrm{NN}(i)} \phi_j}{2d + m^2} \tag{7}$$

$$\sigma^2_{\mathrm{HB}} = \frac{1}{2d + m^2} \tag{8}$$

d is the dimension, m a mass parameter, and the sum refers to nearest neighbours of the site ϕ_i to be updated.

In a heatbath update, the ratio of proposal probabilities is the reciprocal of the ratio of state densities, so the whole thing cancels to 1 and we always accept the update.

3.4 MetropolisProposer

update() picks a change to a site's value from a Gaussian:

$$\phi_i + d \tag{9}$$

$$d \sim \mathcal{N}\left(0, d^2_{\mathrm{Max}}\right) \tag{10}$$

where d_{Max} is a setup parameter for the proposer. This update is selected if based on whether a random number r is greater than a function of the change in the action:

$$\phi_i' = \begin{cases} \phi_i, & r < \exp\left(-\beta d S\right) \\ \phi_i + d, & \text{Otherwise} \end{cases} \tag{11}$$

updateCycle()

3.5 DummyProposer

A default, has stubs for update methods.

4 Derivation

$$S = \tfrac{1}{2}\phi^T A \phi \tag{12}$$