Lessons from Energy-Based Machine Learning for Biology

HHMI Janelia Junior Scientist Workshop in Theoretical Biophysics

Kristina Trifonova

October 2025

1 Energy-based Models

1.1 What is the learning problem?

Most learning problems are of the following type: given a dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is a matrix of independent variables and $\mathbf{y}$ is a vector of dependent variables, one wants to fit a model $f(\mathbf{x}; \boldsymbol{\theta})$ which is a function $f : \mathbf{x} \to y$ parameterized by $\boldsymbol{\theta}$ to the data $\mathcal{D}$.

The goal is to learn the right parameters $\boldsymbol{\theta}$ of the model f to accurately predict outputs from the inputs in our dataset. Roughly, we say that we have successfully learned f if we have chosen the right $\hat{\boldsymbol{\theta}}$ such that $f(\mathbf{x}; \hat{\boldsymbol{\theta}}) = \hat{y}$ for $(\mathbf{x}, \hat{y}) \sim \mathbf{P}(\mathbf{x},\ \mathbf{y})$, where $\mathcal{D} \sim \mathbf{P}$, suggesting that the learned parameters contain some information about the underlying distribution from which our data was sampled.

1.2 Energy-based models

1.2.1 What is an energy-based model?

A typical choice of model is a neural network $f(\mathbf{x}; \boldsymbol{\theta})$, that takes inputs $\mathbf{x}$ and passes them through hidden layers to directly compute a predicted output $\hat{y}$. To train the model, one compares the predicted $\hat{y}$ to the target $y^* \in \mathcal{D}$ with a loss function $\mathcal{L}(y; \boldsymbol{\theta})$ and uses backpropagation to iteratively update $\boldsymbol{\theta}$.

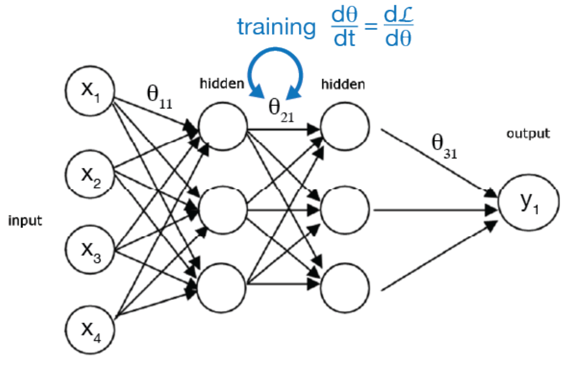

Figure 1: Training a neural network

In contrast, energy-based models use a model defined by $E(\mathbf{X}, \mathbf{y}; \boldsymbol{\theta})$ which is a function $E : (\mathbf{x}, \mathbf{y}) \to s \in \mathbb{R}$ that maps all variables of the system to a scalar energy. The model's prediction $\hat{y}$ corresponds to a minimum of the energy landscape for some value of the inputs $\mathbf{x}^* \in \mathcal{D}$:

$$\hat{y} = \text{argmin } E(\mathbf{x}^*, \mathbf{y})$$

It is not directly computed, as in the case above, but rather obtained through optimization on the landscape. Training reshapes the model's energy landscape to make data points $(\mathbf{x}^*, y^*) \in \mathcal{D}$ lower energy.

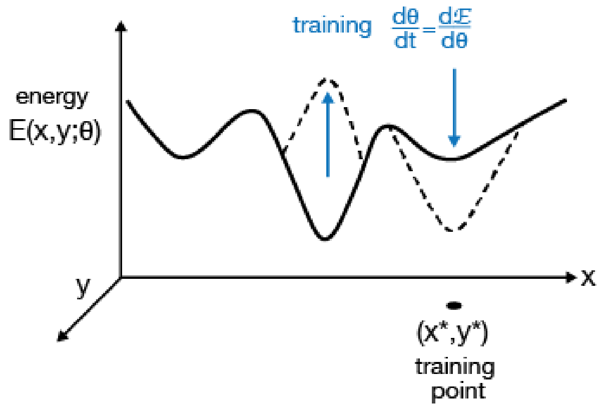

Figure 2: Training an energy-based model.

1.2.2 Why use an energy-based model?

Training a neural network allows you to learn a high-dimensional mathematical function. However, optimizing an energy-based model lets you learn a proper, normalized probability distribution, since energies can be used to define a Boltzmann distribution over states of the system (where a state is given by $\mathbf{x}, \mathbf{y}$ for a particular set of parameters $\boldsymbol{\theta}$).

$$\mathbf{P}(\mathbf{X}; \boldsymbol{\theta}) = \frac{e^{-\beta E(\mathbf{x},\mathbf{y};\boldsymbol{\theta})}}{\sum_{\mathbf{x},\mathbf{y}} e^{-\beta E(\mathbf{x},\mathbf{y};\boldsymbol{\theta})}} \tag{1}$$

This is particularly useful for generative tasks, in which one aims to generate new data points sampled from the same distribution as training data. Additionally, training these models does not required labeled data (since there is no notion of input or output), making them ideal for many unsupervised learning tasks. Modern examples of energy-based models include score-based diffusion models and even generative adversarial networks (GANs).

Energy-based models are not usually used for discriminative tasks, which involve predicting a label for inputs to sort them into different classes. In practice, they could solve these tasks by modeling a distribution that is conditional on the inputs $\mathbf{P}(y \mid \mathbf{x}; \theta)$. However, they are often overkill for this kind of structured data with labels. This is because:

1. **Normalization is hard.** Computing the partition function Z is often costly and can be intractable for large problems.

2. **Training requires negative examples.** To reshape the energy landscape, one needs to both create new energy minima at training points and destroy spurious minima of the existing landscape. In practice, this requires you to train on both positive and negative examples.

3. **Sampling is expensive.** Obtaining $\hat{y}$ requires sampling your existing model at every iteration of training. This can be costly, slow, and inefficient.

1.3 Physical and biological systems as energy-based models

Many physical (and biological) systems naturally have characteristics that are reminiscent of energy-based models. Most importantly:

1. **Dynamics follow energy gradient** Biology and physics offer a range of systems whose dynamics follow or approximate gradient descent on an energy landscape. This means that these systems do the equivalent of inference on a trained energy landscape (i.e. finding $\hat{y}$) for free. The system's dynamics naturally take it to states corresponding to energy minima – no need for computationally costly sampling of $\mathbf{P}(\mathbf{x};\boldsymbol{\theta})$ or computing of Z.

2. **Have trainable parameters** Many-body systems (e.g. molecular networks, liquid condensates, etc.) are parameterized by numerous coupled interactions and variables, any of which could play the role of trainable parameters $\boldsymbol{\theta}$ that reshape their energy landscapes.

3. **Fits into existing frameworks of biology** In many scenarios, we already think about the function of biological systems as an input-output computation. Thinking of biological systems as energy-based models extends this framework, suggesting that the computation they perform is trainable and molded by experience. This framing is similar to adaptation, and it is not contrived to imagine that such behavior could provide a fitness advantage to biological system.

However, things get trickier when it comes to training. Biology obeys physical constraints and conservation laws that computer models don't, meaning that most algorithms used for training on a computer don't immediately map to biological processes. The main discrepancies are:

1. **Biological interaction parameters can be hard to change** In machine learning, trainable parameters are usually weights/edges connecting network nodes. However, biological interaction parameters are thought to be hardwired by evolution, through mutation and selection on a population level (e.g. binding constants, rates, interactions, etc.). As such, it is hard to imagine a biologically plausible "learning algorithm" that trains these interactions without evolution.

2. **Backpropagation is based on global information** In machine learning, most energy-based models are trained with backpropagation. Biological backpropagation would require a system to change individual training parameters based on how their change affects a global function (e.g. error, etc.) that also depends on many other system parameters. While possible to engineer, (and potentially a description of how evolution changes network functions) it

would require complex network architectures and feedback, especially in molecular systems. [1]

2 Learning in Biology: Hebb's Rule and Local Learning

2.1 Local learning rules

We know that some cells – particularly those in the brain – solve learning tasks. To understand how biological learning might fit into the energy-based framework, we look to neuroscience.

Hebb's rule is an early, simplified model of how neurons in the brain learn associations without backpropagation.[2] Simply stated, Hebb's rule says "neurons that fire together, wire together," in other words, that the synaptic coupling between two neurons strengthens if both are active simultaneously.

This is an example of a local learning rule because synaptic couplings w_{ij} are updated based on the activities of neighboring neurons i and j, independent from the activity of any other neurons or information about the global function of the network. Eventually, the strength of synaptic couplings encodes associations between neurons, learned from repeated patterns of activation.

$$\frac{\partial w_{ij}}{\partial t} \propto \eta_i \eta_j \tag{2}$$

In this example and many others, the notion of locality is spatial. However, the only requirement for a learning rule to be local is that parameter updates are not based on any global function (error, fitness, etc.) of the network. A heuristic for locality is that the index of the training parameters matches that of the training signal informing updates.

2.2 Hopfield Models

The Hebbian rule can be used to train energy-based models. However, it cannot train models to learn true probability distributions. Instead, it leads to memorization. We illustrate this principle in Hopfield networks.

One of the simplest energy-based models that is often trained using Hebb's rule is the Hopfield network. This model consists of a network of N binary spins $\{x_i, ...x_N\}$ with symmetric, all-to-all connections $\{w_{ij}\}$. Its dynamics flip spins to flow towards minima of its energy function:

$$E = -\frac{1}{2}\sum_{ij} w_{ij} x_i x_j \tag{3}$$

[1] For an example of a chemical reaction network that implements backpropagation see Lakin, 29 (2023)

[2] We now know that things like spike-timing-dependent plasticity and long-term potentiation/depression are the more complex versions of Hebb's rule that actually happen in the brain.

The Hopfield network is trained to memorize patterns of spins $\mu^i = \{x_1^i, ...x_N^i\}$ by imposing the pattern μ onto its spins and reinforcing couplings between like spins using a Hebbian rule:

$$w_{ij} = \sum_\mu x_i^\mu x_j^\mu \tag{4}$$

The updated $\{w_{ij}\}$ define new attractors on the energy landscape, each one corresponding to a memory of the patterns observed during training. If one initializes the network and lets its dynamics play out, it will fall into the closest attractor, recovering the pattern of spins most similar to its initial configuration.

2.3 Going beyond Hebb's rule and memorization

The Hopfield model example shows that Hebb's rule alone is not sufficient to train physical systems to learn. As described above, the learned parameters lead the model to reproduce the exact patterns it was trained on but do not allow it to generate new patterns. The model does not learn underlying trends in data or generalize beyond what it's seen.

This is exemplified in the trained Hopfield model's dynamics. If we give the model a new input, e.g. initialize it near an attractor, it will always flow towards an existing minimum and retrieve a pattern it saw during training, never producing any new outputs.

Evidently, in order to learn a distribution, we must control both the location and relative depth of attractors in the energy landscape. To do this, we will need a framework that goes beyond Hebb's rule.

3 The Boltzmann Machine

3.1 An energy-based model for learning probability distributions

Hinton's innovation in the Boltzmann machine was coming up with a learning rule that trains energy-based models to learn generative probability distributions using local information.

The model itself is similar to a Hopfield model — it is a network of N binary spins $\{x_1, ...x_N\}$, $x_i \in \{0, 1\}$ with symmetric all-to-all connections θ_{ij}. The network's dynamics minimize its energy

$$E(v, h; \theta) = -\sum_{ij} \theta_{ij} x_i x_j \tag{5}$$

A subset of the spins are designated as visible units.

$$\{x_1, ...x_k\} := \{v_1 ... v_k\} \in \mathcal{V} \tag{6}$$

The rest are deemed hidden units (e.g. latent variables).

$$\{x_{k+1} ... x_N\} := \{h_1 ... h_m\} \in \mathcal{H} \tag{7}$$

The goal of training is to adjust spin couplings θ_{ij} to model a probability distribution over the visible spins, defined via a Boltzmann distribution using the system's energy:

$$\mathbf{P}(v,h) = \frac{e^{-\beta E(v,h)}}{Z} \tag{8}$$

where Z is the partition function.

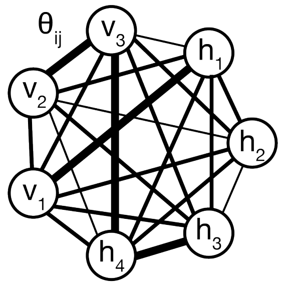

Figure 3: Boltzmann Machine with three visible units, four hidden units, and connections θ_{ij}

3.2 Contrastive learning: a local rule for generative modeling

Formally, the goal of Boltzmann machine training is to minimize the Kullback-Leibler divergence between the model's distribution over visible spins, $P_{\text{model}}(v;\theta)$, and a target training data distribution, $Q_{\text{data}}(v)$, by adjusting training parameters θ_{ij}. Using this formulation, we derive Hinton's learning rule (an approach called contrastive divergence).

Our training objective is to find

$$\hat{\boldsymbol{\theta}} = \text{argmin}_{\theta}\ D_{KL}(Q_{\text{data}}(\mathbf{v})||P_{\text{model}}(\mathbf{v};\boldsymbol{\theta}) \tag{9}$$

The KL-divergence to minimize is:

$$D_{KL}(Q_{\text{data}}||P_{\text{model}}) = \sum_{v\in\mathcal{V}} Q(v)\log\frac{Q(v)}{P(v;\theta)} \tag{10}$$

We expand the KL-divergence

$$D_{KL}(Q||P) = \sum_{v} Q(v)\log Q(v) - Q(v)\log P(v;\theta) \tag{11}$$

Hence, we see that the first term does not depend on θ_{ij}.

Therefore, minimizing D_{KL} is equivalent to maximizing the second term. In other words, this optimization is equivalent to maximizing the expected log-likelihood of observing the model distribution $P(v;\theta)$ under the data distribution $Q(v)$.

$$\hat{\boldsymbol{\theta}} = \text{argmax}_{\boldsymbol{\theta}}\ \langle\log P(\mathbf{v};\boldsymbol{\theta})\rangle_{\text{data}} \tag{12}$$

Assuming a Boltzmann distribution and expanding, we get that the second term is equivalent to

$$= \sum_v Q(v) \log \sum_h e^{-\beta E(v,h;\theta)} - \sum_v Q(v) \log \sum_{v,h} e^{-\beta E(v,h;\theta)} \tag{13}$$

The first term simplifies to the average energy of the model under the target distribution Q (h units are allowed to sample all possible values of h for a fixed distribution over v given by Q and the model's current θ values). The second term is the average energy of the model for the current values of θ with no constraints on v.

Now, we take a derivative of D_{KL} with respect to θ_{ij}. We start with the first term:

$$= \sum_v Q(v) \frac{\sum_h \frac{\partial(-\beta E)}{\partial \theta_{ij}} e^{-\beta E(v,h)}}{\sum_h e^{-\beta E(v,h)}} \tag{14}$$

$$= -\beta \sum_v Q(v) \sum_h P(h|v) \frac{\partial E}{\partial \theta_{ij}} \tag{15}$$

$$= -\beta \Big\langle \frac{\partial E}{\partial \theta_{ij}} \Big\rangle_{\mathrm{Q}} \tag{16}$$

$$\tag{17}$$

This is the data expectation (positive/wake phase).

Now, we take the gradient of the second term:

$$= -\frac{\partial}{\partial \theta_{ij}} \log Z \sum_v Q(v) \tag{18}$$

$$= -\frac{1}{Z} \frac{\partial Z}{\partial \theta_{ij}} \tag{19}$$

$$= -\sum_{v,h} P(v,h) \frac{\partial(-\beta E)}{\partial \theta_{ij}} \tag{20}$$

$$= \beta \Big\langle \frac{\partial E}{\partial \theta_{ij}} \Big\rangle_{\mathrm{P}} \tag{21}$$

$$\tag{22}$$

This is the model expectation (negative/sleep phase).

Putting them together, we see that the gradient of the KL-divergence is:

$$\frac{\partial D_{KL}}{d\theta_{ij}} \propto \Big\langle \frac{\partial E}{\partial \theta_{ij}} \Big\rangle_Q - \Big\langle \frac{\partial E}{\partial \theta_{ij}} \Big\rangle_P \tag{23}$$

We want to change θ_{ij} to follow the negative gradient of D_{KL}. Using the energy equation 5 to simplify, we get the following learning rule:

$$\frac{d\theta_{ij}}{dt} \propto -\frac{\partial D_{KL}}{d\theta_{ij}} = \langle x_i x_j \rangle_Q - \langle x_i x_j \rangle_P \tag{24}$$

Implementing this learning rule in practice requires alternating between two phases:

1. **Positive/wake phase:** the visible units sample positive examples of training data from Q and θ_{ij} are updated in a Hebbian manner to reinforce correlations, lowering the energy and raising the probability of these data points
2. **Negative/sleep phase:** the visible units sample negative examples from P generated by the model and θ_{ij} are updated in an anti-Hebbian manner to penalize these correlations, raising the energy and lowering the probability of these data points

To summarize, this learning rule prescribes a way to change training parameters θ_{ij} to match the model's distribution to a target data distribution $Q(v)$, using the difference in two-body correlations as a training signal. This is a local learning rule, because it updates weights θ_{ij} based only on the activities of x_i and x_j.

Boltzmann machine training will serve as the inspiration for how we think about training physical systems.

4 Translating to Biological Systems

4.1 General framework

Now, we're going to take Boltzmann machine training and show how it might map to natural processes biological systems do, potentially allowing them to learn a probability distribution from their environment. To do this, we're going to take Hinton's framework for the Boltzmann machine and apply it to an example statistical system.

4.2 A biological Boltzmann machine

Suppose there is a 2D lattice with N sites, where each site is occupied by one of M different types of molecules (a random bond Potts model). A microstate/configuration σ of this model is determined by the molecule type at each site on the lattice.

$$\sigma = \{x_1, x_2, ...x_N\},\ x_i \in \{A, B, ...\} \tag{25}$$

Suppose the system is in a grand canonical ensemble Ω, where it is in contact with an infinite reservoir of each molecule type. Thus, allowed configurations of the lattice include all $\sigma \in \Omega$, where each site on the lattice can be occupied by a molecule of any type, with no restrictions on macroscopic counts.

Molecules that are next to each other interact with some energy J_{ij}. Additionally, each molecule is accompanied by an intrinsic chemical potential μ_i. The energy of a particular configuration is thus a sum over chemical potentials of each species and interaction energies between each pair of adjacent molecules, both weighted by their counts.

$$\mathcal{G}(\sigma) = \sum_{i \le j \in M} n_{i,j} J_{i,j} + \sum_{i \in M} n_i \mu_i \tag{26}$$

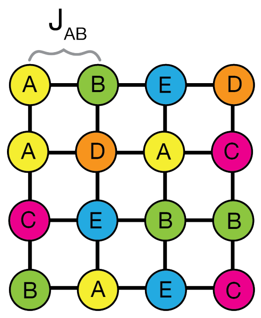

Figure 4: Potts model we want to train

In the Boltzmann machine, we aimed to learn a distribution over visible units. In this framework, we will aim to learn a distribution over observables of the system.

Suppose the observable we are interested in is $n_A(\sigma)$, the total count over all lattice sites of molecules of type A.

$$n_A = \sum_{i,j} \delta^A(x_{ij}), \ \delta^A = \begin{cases} 1, & \text{if } x_{ij} = A \\ 0 & \text{if } x_{ij} \neq A \end{cases} \tag{27}$$

Suppose you want to learn a distribution $Q_{n_A}(\alpha)$ over all possible $n_A(\sigma)$ macrostates, i.e. a probability distribution over every possible count of molecule A.

In particular, we will assume that we want to have the system have a lot of molecule A most of the time, as shown in the sketch of $Q_{n_A}(\alpha)$ below. Suppose that the system currently has a different distribution $P_{n_A}(\alpha)$ over molecule A counts, where most of the time very few molecules of A show up on the lattice. Our goal will be to tune a training parameter to get P_{n_A} to look like Q_{n_A}, so that molecules of A show up more frequently.

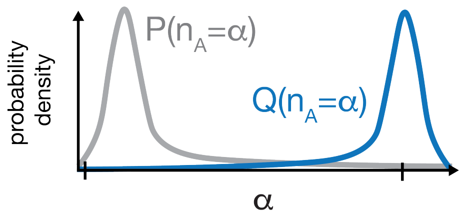

Figure 5: Current model distribution (P) vs. target distribution (Q) over possible counts of A

Suppose our training parameters are J_{ij}, the interaction energies between species i and j when they're next to each other on the lattice. We want to change J_{ij} to minimize $D_{KL}(Q_{n_A}||P_{n_A})$. Following the same derivation in section 3 gives us a Boltzmann machine-like rule.

Using the system's energy in 26, we see that

$$\frac{\partial \mathcal{G}}{\partial J_{ij}} = n_{i,j} \tag{28}$$

Following the derivation in section 3 we see that, for this model, the Boltzmann learning rule is:

$$\frac{dJ_{i,j}}{dt} \propto -\frac{\partial D_{KL}}{\partial J_{i,j}} = \langle n_{i,j} \rangle_P - \langle n_{i,j} \rangle_Q \tag{29}$$

To train this lattice to "learn" a distribution $Q_{n_A}(\alpha)$, the learning rule would prescribe the following:

1. **Wake phase:** clamp the lattice to $Q_{n_A}(\alpha)$, forcing each macrostate of n_A to show up with probabilities corresponding to those in the target distribution. Sample to find $\langle n_{i,j} \rangle_Q$.
2. **Sleep phase:** remove constraints to return to the grand canonical ensemble Ω, letting the system freely sample $P_{n_A}(\alpha \; ; \; J_{ij})$ to find $\langle n_{i,j} \rangle_P$.
3. **Weight update:** update J_{ij} based on the difference in $\langle n_{i,j} \rangle$ in the sleep and wake phases.

The process above would be repeated until, eventually, the sleep and wake phases look identical, meaning that $P_{n_A}(\alpha) = Q_{n_A}(\alpha)$ (or else until the distributions are as close as possible).

4.3 Intuition, Practical Implementation & Notes

Intuitively this rule might lead to something like the following. Let's return to our example set up, in which we want to learn $Q_{n_A}(\alpha)$ where the system mostly samples configurations with high counts of molecule A even though our system currently has a distribution P_{n_A} in which molecules of A don't show up often.

Suppose that when clamping the distribution to Q in the wake phase, lots of molecules of B show up alongside A. The learning rule would suggest strengthening the coupling between A and B, J_{AB}. Now, even in the unclamped distribution, A molecules have a higher chance of being pulled in thanks to strengthened interactions with B molecules.

Conversely, suppose that letting the system sample P (the wrong distribution) in the sleep phase leads many molecules of C to show up. Maybe this is because A and C have repulsive interactions. The learning rule would thus say to weaken the coupling between A and C, J_{AC}. Now, we've made it easier for molecules of A to show up, moving us closer to our desired distribution Q.

Some miscellaneous notes.

1. In practice, "clamping" the system in the wake phase might look like switching the statistical ensemble of the system from grand canonical to hybrid canonical, in which there is a fixed chemical potential for molecules of type A, enforcing the particular distribution Q.
2. You cannot learn a distribution over macrostates which is not possible to clamp to — this learning rule makes no guarantees that clamping is possible. It just says that *if* you can clamp to it, you can learn it.
3. The choice of observable to learn is independent of the learning rule; it informs how you must clamp. We could have chosen a different observable and used the same learning rule to match Q.

The big idea. We've shifted from the Boltzmann machine framework of learning distributions over visible units by switching between sleep and wake phases to a thermodynamic framework where we learn distributions over macrostates of a statistical system by switching between ensembles.

For further exploration of these ideas in particular systems and settings (specifically in cellular many-to-many molecular networks), stay tuned for my publication coming out soon!