PHYS 414 Problem Set 3:
Machine learning redux

Problem 1: Machine learning and Markov chain Monte Carlo

In Problem 2 of the first homework, you developed a simple machine learning algorithm that learned to classify points on a two-dimensional plane. The network was described by a set of parameters $\boldsymbol{w}$, and the goal of the problem was to find the parameters $\boldsymbol{w}^*$ that maximized the posterior distribution $\mathcal{P}(\boldsymbol{w}|\mathcal{D})$. This would give the single network that is most likely to describe the data $\mathcal{D}$. However in many real world scenarios you may interested in not just a single optimal network that solves your problem, but also the degree of certainty in the network's answer. For example, imagine you trained a network to classify radiology images to detect cancers, and you applied the network to data from a new patient. If the network detected a tumor in the image, you would want to know whether this is a clear-cut case or something borderline. Designing "smarter" networks that indicate their own certainty or confusion is the goal of *Bayesian machine learning*. In this problem, you will revisit your old code to make it Bayesian.

One way to measure uncertainty is to look at a whole ensemble of networks, each with a different set of parameters $\boldsymbol{w}$. Imagine these parameters were distributed according to the probability distribution $\mathcal{P}(\boldsymbol{w}|\mathcal{D})$. The network parameters that better described the data would appear more often in the ensemble, because their values of $\mathcal{P}$ would be higher, while those that were quite terrible (with low $\mathcal{P}$) would appear very infrequently. The optimal network $\boldsymbol{w}^*$ would appear most frequently. If we had such an ensemble, let us say K different parameter sets, we could label them $\{\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(K)}\}$. Each set here is a three-vector, $\boldsymbol{w}^{(\alpha)} = (w_1^{(\alpha)}, w_2^{(\alpha)}, w_3^{(\alpha)})$ for $\alpha = 1, \ldots, K$. In part c) of the earlier problem we had particular data points we were interested in classifying, for example $\boldsymbol{x}^B = (-1, -1)$. There we just looked at the output the optimal network, $p_1(\boldsymbol{x}^B; \boldsymbol{w}^*)$. But if we had an ensemble, we could calculate the outputs of all the K different networks in the ensemble:

$$p_1(\boldsymbol{x}^B; \boldsymbol{w}^{(\alpha)}), \quad \alpha = 1, \ldots, K. \tag{1}$$

The spread in the results tells us how confident we can be in the prediction for $\boldsymbol{x}^B$: if the vast majority of the networks give a similar answer, we can be highly certain in the classification. If there is a huge spread in the answers, we cannot trust the classification of the data point. The ensemble encapsulates the "wisdom of crowds": for confident classification we seek general agreement among the many network predictions, despite their differences in parameters.

Interestingly, the problem of generating this ensemble is closely related to one of the earliest and most famous algorithms in computational physics, *Markov chain Monte Carlo* (MCMC). The algorithm was developed as a byproduct of research on thermonuclear weapons at Los Alamos after World World II, but it aims to solve a very general problem in statistical physics. Imagine you have a physical system whose states are labeled by some vector $\boldsymbol{s}$, which can be very high dimensional (for example labeling configurations of all possible spins on a lattice). In class we typically label states by a scalar n, and this is just a generalization of that idea. Associated with each system state $\boldsymbol{s}$ is some energy $E(\boldsymbol{s})$, which we can calculate. What if we were interested

in equilibrium properties of the system, for example the average $\langle A \rangle$ of some function $A(\boldsymbol{s})$ that corresponds to a physical observable. We know that in equilibrium the states are distributed according to the Boltzmann distribution, $p^{\rm eq}(\boldsymbol{s}) = \exp(-\beta E(\boldsymbol{s}))/Z$, and hence the average can be written as:

$$\langle A \rangle = \sum_{\boldsymbol{s}} A(\boldsymbol{s}) p^{\rm eq}(\boldsymbol{s}). \tag{2}$$

This sum may be difficult to carry out exactly for systems with many states, even on modern supercomputers (much less the MANIAC computer available at Los Alamos in 1952). For example a 100×100 lattice of Ising spins, where each spin can be ± 1, has $\approx 2 \times 10^{3010}$ distinct states, and hence that many terms in the sum. Clearly we have to use some kind of approximation. The above sum is a weighted average, where each contribution $A(\boldsymbol{s})$ is weighted by the probability $p^{\rm eq}(\boldsymbol{s})$. If we had a large set of K random samples $\{\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \ldots, \boldsymbol{s}^{(K)}\}$ drawn from the distribution $p^{\rm eq}(\boldsymbol{s})$, we could just approximate $\langle A \rangle$ as the mean of $A(\boldsymbol{s})$ over this set,

$$\langle A \rangle \approx \frac{1}{K} \sum_{\alpha=1}^{K} A(\boldsymbol{s}^{(\alpha)}). \tag{3}$$

So long as the samples $\boldsymbol{s}^{(\alpha)}$ are distributed according to probability $p^{\rm eq}(\boldsymbol{s})$, we know that as K gets larger the approximation in Eq. (3) gets closer and closer to Eq. (2). And we will typically get convergence of the result at much smaller K than the full size of the state space. This is because for many states $\boldsymbol{s}$ the associated equilibrium probability $p^{\rm eq}(\boldsymbol{s})$ will be small (the energies $E(\boldsymbol{s})$ relatively high), and the contribution to the sum in Eq. (2) will be negligible. But these high-energy states are the same ones we are unlikely to pick if we are randomly sampling from $p^{\rm eq}(\boldsymbol{s})$ to generate the ensemble for Eq. (3). Thus we can get a good approximate result even when leaving them out.

The crux of the issue then boils down to one question: if we have some complicated probability distribution $p^{\rm eq}(\boldsymbol{s})$, how do we create an ensemble of K samples $\{\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \ldots, \boldsymbol{s}^{(K)}\}$ drawn from this distribution? The remarkably simple solution was described in the seminal 1953 paper "Equation of State Calculations by Fast Computing Machines", by Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller. Published in the *Journal of Chemical Physics*, it is one of the top cited physics papers of all time, amassing some 45,000 citations to date. The reason it is so useful is that the problem of generating samples from a given distribution is ubiquitous in many areas of science. Because the algorithm works for any function $E(\boldsymbol{s})$—it does not have to be an actual physical energy—we can make $p^{\rm eq}(\boldsymbol{s})$ correspond to any distribution we are interested in, even if it has nothing to do with thermodynamic equilibrium. We will use precisely this trick to generate the ensemble of network parameters for Bayesian machine learning. Today there are many variations of the original algorithm, falling in the broad category of "MCMC techniques" (a research area in its own right).

To understand the trick behind the original MCMC algorithm, imagine we had a system with discrete states $n = 1, \ldots, N$ and an $N \times N$ Markovian transition probability matrix W. The component W_{nm} of the matrix describes the probability that you make a transition to state n in time δt, given that you started in state m. If these transitions satisfy detailed local detailed balance,

$$\frac{W_{nm}}{W_{mn}} = e^{-\beta(E_n - E_m)} = \frac{p_n^{\rm eq}}{p_m^{\rm eq}}, \tag{4}$$

then we know that if we wait long enough, the sequence of states generated by the transitions should eventually be distributed according to the Boltzmann equilibrium probability $p_n^{\text{eq}} = \exp(-\beta E_n)/Z$. This gave Metropolis and his collaborators an idea: what if you were to artificially construct a Markovian transition process that took you from one state $\boldsymbol{s}$ to another $\boldsymbol{s}'$, which was designed to satisfy a certain detailed balance relation? Here the state $\boldsymbol{s}$ does not even have to be discrete, but we need to specify the probability $W(\boldsymbol{s} \to \boldsymbol{s}')$ that you go from one $\boldsymbol{s}$ to another $\boldsymbol{s}'$ (this plays the role of the transition matrix element). We will demand that these probabilities satisfy

$$\frac{W(\boldsymbol{s} \to \boldsymbol{s}')}{W(\boldsymbol{s}' \to \boldsymbol{s})} = e^{-\beta(E(\boldsymbol{s}')-E(\boldsymbol{s}))} = \frac{p^{\text{eq}}(\boldsymbol{s}')}{p^{\text{eq}}(\boldsymbol{s})}. \tag{5}$$

If we start at a random state $\boldsymbol{s}^{(1)}$ and then use our transition procedure to go from $\boldsymbol{s}^{(1)} \to \boldsymbol{s}^{(2)}$, and then from $\boldsymbol{s}^{(2)} \to \boldsymbol{s}^{(3)}$ and so on, we will eventually have created a set of K random samples $\{\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \ldots, \boldsymbol{s}^{(K)}\}$ that are guaranteed to be distributed according to $p^{\text{eq}}(\boldsymbol{s})$, assuming K is large. The only question is how to construct a transition procedure that satisfies Eq. (5). We will return to this issue below.

To see how this is relevant to the machine learning case, let us make a few analogies: the parameter vector of the network $\boldsymbol{w}$ is analogous to the state $\boldsymbol{s}$. The distribution $\mathcal{P}(\boldsymbol{w}|\mathcal{D})$ that we want to sample from is analogous to the equilibrium distribution $p^{\text{eq}}(\boldsymbol{s})$. The energy function in the thermodynamic case was related to the distribution through $E(\boldsymbol{s}) = -k_B T \log p^{\text{eq}}(\boldsymbol{s}) - k_B T \log Z$, by inversion of the Boltzmann distribution equation. Similarly we can define an "energy" in the machine learning case as $E(\boldsymbol{w}) = -\log \mathcal{P}(\boldsymbol{w}|\mathcal{D})$. Note that we have set $k_B T = 1$ here and ignored the offset $-k_B T \log Z$, since adding a constant to the energy does not affect the physics. We have encountered this "energy" before: it is precisely the loss function for machine learning, $\mathcal{L}(\boldsymbol{w}|\mathcal{D})$, for which you derived an expression in part a) of the earlier problem. Thus we will use $E(\boldsymbol{w}) \equiv \mathcal{L}(\boldsymbol{w}|\mathcal{D})$. All that is left to do is to specify a transition process that will update the network parameters at each step, such that the transition probabilities satisfy

$$\frac{W(\boldsymbol{w} \to \boldsymbol{w}')}{W(\boldsymbol{w}' \to \boldsymbol{w})} = e^{-(E(\boldsymbol{w}')-E(\boldsymbol{w}))}. \tag{6}$$

How to set up such an update process is the goal of the next part.

a) The heart of the MCMC algorithm is the following update procedure. Let the current network parameter set be $\boldsymbol{w}$. To find $\boldsymbol{w}'$ at the next time step, we choose a random perturbation $\delta\boldsymbol{w}$. The details are up to us, but a common choice is making each component of $\delta\boldsymbol{w}$ a small random number between $-\epsilon$ and ϵ, for $\epsilon \ll 1$. The update rule at each time step is as follows:

1. Calculate $\rho = e^{-(E(\boldsymbol{w}+\delta\boldsymbol{w})-E(\boldsymbol{w}))}$. If $\rho \geq 1$, then set $\boldsymbol{w}' = \boldsymbol{w} + \delta\boldsymbol{w}$ (we accept the perturbation).

2. If $\rho < 1$, then choose a random number r drawn uniformly from the range 0 to 1. If $r < \rho$, then then set $\boldsymbol{w}' = \boldsymbol{w} + \delta\boldsymbol{w}$ (we accept the perturbation). If $r > \rho$, then set $\boldsymbol{w}' = \boldsymbol{w}$ (we reject the perturbation, and keep the same parameter set for the next iteration step).

We can think of this procedure as specifying a random Markovian transition process that takes some input state $\boldsymbol{w}$ and outputs a state $\boldsymbol{w}'$. Clearly the process is reversible, because once we get to the new state $\boldsymbol{w}'$, there is a chance of accepting the exact reverse perturbation that will

take us back to the state $\boldsymbol{w}$. Argue that the procedure above satisfies the local detailed balance condition of Eq. (6). *Hint:* Think about every scenario of acceptance / rejection that can happen for a generic transition from $\boldsymbol{w} \to \boldsymbol{w}'$, and what those probabilities are. Then consider how the algorithm would behave with the corresponding reverse transition from $\boldsymbol{w}'$ to $\boldsymbol{w}$. For example, if you have a certain value of ρ going from $\boldsymbol{w} \to \boldsymbol{w}'$, you know that going from $\boldsymbol{w}'$ to $\boldsymbol{w}$ you have a different ρ that is exactly one divided by the original ρ. Using insights like this, you can work out a simple argument. You do not need to do any elaborate math or explicitly figure out the whole matrix W (which is infinitely large).

b) Now that we have chosen an update procedure, let us test it out in code. You already have the code that calculates the loss function $E(\boldsymbol{w}) \equiv \mathcal{L}(\boldsymbol{w}|\mathcal{D})$ from before. Set the regularization parameter $s = 1$. Choose an arbitrary starting guess for the first set of network parameters, $\boldsymbol{w}^{(1)} = (1, 1, 1)$. Now generate $\boldsymbol{w}^{(2)}$, $\boldsymbol{w}^{(3)}$, and so on, using the MCMC procedure. To choose a perturbation $\delta\boldsymbol{w}$, at each time step we can choose three random numbers (q_1, q_2, q_3), each q_i between 0 and 1. Then set $\delta\boldsymbol{w} = 0.05(q_1 - 0.5, q_2 - 0.5, q_3 - 0.5)$. This guarantees that $\delta\boldsymbol{w}$ is small, and we explore both positive and negative perturbations to the network parameters. After each MCMC step, save the set of network parameters $\boldsymbol{w}^{(\alpha)}$ you generate in this process (as a row in an array), and repeat until you collect a large number K of samples. K should be fairly big, for example $K > 10^4$.

c) Calculate the mean $\boldsymbol{w}$ of your set of samples $\{\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \ldots, \boldsymbol{w}^{(K)}\}$. Compare it to the optimal parameter set $\boldsymbol{w}^*$ you found in the earlier problem set. You should find that they are quite similar: the MCMC procedure generates a "cloud" of points in the space of network parameters that is centered around $\boldsymbol{w}^*$. This is because $\boldsymbol{w}^*$ is where the probability distribution of the cloud, $\mathcal{P}(\boldsymbol{w}|\mathcal{D})$, has its peak.

d) Let us consider the test point $\boldsymbol{x}^B = (-1, -1)$. Plot a histogram of the predictions

$$p_1(\boldsymbol{x}^B; \boldsymbol{w}^{(\alpha)}), \quad \alpha = 1, \ldots, K. \tag{7}$$

for all K parameter sets in your ensemble. You should find this histogram to be quite broad: $\boldsymbol{x}^B$ is a borderline point where there is a lot of confusion amoung the different networks, and hence we would never put a huge amount of trust in the machine learning classification. (If this was a cancer diagnosis, we would definitely send the patient to get additional testing.) Now plot similar histograms for the test points $\boldsymbol{x}^A = (-1, -4)$ and $\boldsymbol{x}^C = (-1, 3)$. In contrast, here you should find very narrow distributions: there is a near consensus among the networks in the ensemble, and hence we can trust the machine learning predictions.