CS 189/289A Introduction to Machine Learning
Fall 2024 Jennifer Listgarten, Saeed Saremi

DIS11

1 Score-matching

We focus on the task of training a generative model approximating a data distribution $p_D(x)$. More particularly, we consider score-based generative models parameterized as $p_\theta(x) = \frac{1}{Z_\theta}\exp(-E_\theta(x))$. Here, E_θ typically corresponds to a neural network, where θ denotes the set of parameters learned during training; Z_θ is a normalizing constant that ensures that $p_\theta(x)$ is a density.

Last week, we covered a first approach to train the model via maximum likelihood:

$$\max_\theta \mathbb{E}_{p_D(x)}\left[\log p_\theta(x)\right]. \tag{1}$$

An alternative approach, named score matching, consists in minimizing the following loss:

$$\min_\theta \mathbb{E}_{p_D(x)}\left[\frac{1}{2}\left\|s_\theta(x) - \nabla_x \log p_D(x)\right\|^2\right], \quad \text{where} \quad s_\theta(x) = \nabla_x \log p_\theta(x).$$

(a) Qualitatively, explain why the new optimization problem is appropriate to train a generative model.

(b) To circumvent this issue, one solution is to rely on noise perturbations of the data. Let $\tilde{x}$ be a perturbed version of x, sampled from $q_\sigma(\tilde{x} \mid x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I)$, that is, $\tilde{x}$ follows a normal distribution, centered at x with covariance $\sigma^2 I$. We consider the modified problem:

$$\min_\theta \mathbb{E}_{p_D(x)} \mathbb{E}_{q_\sigma(\tilde{x}|x)} \left[\frac{1}{2} \left\| s_\theta(\tilde{x}) - \nabla_{\tilde{x}} \log q_\sigma(\tilde{x} \mid x) \right\|^2 \right]. \tag{2}$$

Simplify the new problem.

(c) Suggest an algorithm to learn θ.

(d) One limitation of the maximum likelihood based approach from Equation (1) is that it may learn densities that poorly approximate the true data distribution in low density regions. Justify why this is the case. Is the same behavior expected with generative models trained with Equation (2)?

2 Derivation of the Bellman Equations

This problem attempts to provide some intuitions for Markov Decision Process (MDP). The seemingly simple equations related to MDP sometimes hide subtleties, which we hope to illustrate through this exercise.

In an MDP, at each time step t, an agent receives a representation of the environment's state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$, which leads the environment to produce a reward $R_{t+1} \in \mathcal{R}$ and a new state $S_{t+1} \in \mathcal{S}$. To set up an MDP, we need a few definitions. First, we define its dynamics, because we make the Markov assumption, it suffices to define the *one-step dynamics*:

$$p(s', r|s, a) = \mathbf{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

Then, we define the *return* following time t as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

where $0 \leq \gamma \leq 1$ is the *discount rate*. We also define a *policy* that maps from states to the probabilities of actions:

$$\pi(a|s) = \mathbf{P}[A_t = a|S_t = s]$$

Finally, for all $s \in \mathcal{S}$, we define the *state-value function* of the state s under policy π as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

which represents the expected future return when starting at s and following π thereafter.

Similarly, for all $s \in \mathcal{S}, a \in \mathcal{A}$, the *action-value function* of taking the action a in state s under policy π is:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a]$$

A brief note on the notations: the notation $\mathbb{E}_\pi[X|Y = y]$ is used to indicate the expectation of random variable X conditioning on the random variable Y with value y, when following the policy π. Often, X relates to the reward. Implicitly, the expectation is taken over all variables that is random inside the expectation (except for the ones we condition on), which could include the state, the action and the reward (at the current and future time points), all of which could be influenced by the policy π. Derivations in MDP often involves manipulating this expectation.

(a) To familiarize ourselves with the notations, let's try to understand the relationships between these definitions by expressing the following:

- Express the expectation of R_{t+1} in terms of π and $p(s', r|s, a)$
- Express $v_\pi(s)$ in terms of q_π and π
- Express $q_\pi(s, a)$ in terms of v_π and $p(s', r|s, a)$

(b) Next, we derive the Bellman equation, which demonstrates that a fundamental property of the value function is that it satisfies a certain recursive relationship. There are actually two related notions: the *Bellman expectation equation* and the *Bellman optimality equation*. Let's first derive the Bellman expectation equation (defined for all $s \in \mathcal{S}$):

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

(c) In MDP, we are interested in finding the *optimal policy* which maximizes the value function. We define the *optimal state-value function* as the state-value function under the optimal policy π_*:

$$v_*(s) = \max_\pi v_\pi(s)$$

Similarity, the optimal action-value function as:

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

Let's derive the Bellman optimality equation (defined for all $s \in \mathcal{S}$):

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

Intuitively, this states that the value of a state under an optimal policy equals the expected return for the best action from that state.

3 Space Exploration

You are controlling a spacecraft on a mission to explore and gather data from various celestial bodies in a solar system. The spacecraft can be in one of three states based on its energy levels: 'FullEnergy', 'LowEnergy', and 'Depleted'. 'Depleted' is a terminal state, representing the spacecraft running out of energy and being unable to continue its mission. We denote the states as $\mathcal{S} = \{F, L, D\}$.

At each state (except "Depleted"), there are two possible actions: 'Conserve' and 'Explore'. 'Conserve' represents cautious exploration with energy conservation, while 'Explore' represents aggressive exploration consuming more energy. We denote the actions as $\mathcal{A} = \{C, E\}$.

The one-step dynamics of the system is defined as the following:

From 'FullEnergy':

1. Conserve (FullEnergy → FullEnergy): Probability = 1, Reward = 1 (safe exploration)
2. Explore (FullEnergy → LowEnergy): Probability = 0.5, Reward = 2 (risky but more rewarding exploration)
3. Explore (FullEnergy → FullEnergy): Probability = 0.5, Reward = 2 (successful aggressive exploration without losing much energy)

From 'LowEnergy':

1. Conserve (LowEnergy → FullEnergy): Probability = 0.5, Reward = 1 (successful energy conservation)
2. Conserve (LowEnergy → LowEnergy): Probability = 0.5, Reward = 1 (maintaining energy level)
3. Explore (LowEnergy → Depleted): Probability = 1, Reward = -10 (running out of energy)

In this problem, we will derive the policy iteration and value iteration updates for the MDP above. As a reminder, policy iteration consists of a *policy evaluation* step followed by a *policy improvement* step, defined as:

$$\text{Policy evaluation: } v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

$$\text{Policy improvement: } \pi'(s) = \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Value iteration effectively combines policy improvement and a truncated policy evaluation:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

Note how the policy iteration and value iteration correspond to turning the Bellman expectation equation and the Bellman optimality equation into the respective updates rules.

(a) Suppose we initialize with a policy that always conserves regardless of the states, i.e. $\pi_0(C|s) = 1, \pi_0(E|s) = 0$ for all s. Also, we initialize value functions $v_0(s) = 0$ for all s. Let the discount rate $\gamma = 0.5$. Run policy iteration for two iterations. Does policy iteration converges after two iterations?

(b) Run value iteration for two iterations. Does it converge after two iterations?

(c) You might notice how these algorithms all look very similar to each other. This is because, as we have seen in the previous exercise, they are essentially just different angles of viewing the same relationship (or different ways of rewriting v and q in relation to each other and themselves, if you like). In practice, however, policy iteration and value iteration might have different convergence time, depending on the MDP and the specific implementations. What are some factors that might affect the convergence time of these algorithms?