CS 229, Fall 2020
Problem Set #3

Due Wednesday, November 4 at 11:59 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/fall2020/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work. (4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted. (5) To account for late days, the due date is Wednesday, November 4 at 11:59 pm. If you submit after Wednesday, November 4 at 11:59 pm, you will begin consuming your late days. If you wish to submit on time, submit before Wednesday, November 4 at 11:59 pm. Please make sure to reserve sufficient time for potentially compiling, scanning, and uploading your homework questions.

All students must submit an electronic PDF version of the written questions. We highly recommend typesetting your solutions via LaTeX, and we will award one bonus point for submissions typeset in LaTeX. Please make sure to tag your solutions properly on Gradescope. The graders reserve the right to penalize incorrectly tagged solutions by 0.5 points per question. All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. You should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors. Your submission may be evaluated by the auto-grader using a private test set, or used for verifying the outputs reported in the writeup.

1. **[15 points] KL divergence and Maximum Likelihood**

The Kullback-Leibler (KL) divergence is a measure of how much one probability distribution is different from a second one. It is a concept that originated in Information Theory, but has made its way into several other fields, including Statistics, Machine Learning, Information Geometry, and many more. In Machine Learning, the KL divergence plays a crucial role, connecting various concepts that might otherwise seem unrelated.

In this problem, we will introduce KL divergence over discrete distributions, practice some simple manipulations, and see its connection to Maximum Likelihood Estimation.

The *KL divergence* between two discrete-valued distributions $P(X), Q(X)$ over the outcome space $\mathcal{X}$ is defined as follows[1]:

$$D_{KL}(P\|Q) = \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)}$$

For notational convenience, we assume $P(x) > 0, \forall x$. (One other standard thing to do is to adopt the convention that "$0\log 0 = 0$.") Sometimes, we also write the KL divergence more explicitly as $D_{KL}(P||Q) = D_{KL}(P(X)||Q(X))$.

Background on Information Theory

Before we dive deeper, we give a brief (optional) Information Theoretic background on KL divergence. While this introduction is not necessary to answer the assignment question, it may help you better understand and appreciate why we study KL divergence, and how Information Theory can be relevant to Machine Learning.

We start with the *entropy* $H(P)$ of a probability distribution $P(X)$, which is defined as

$$H(P) = -\sum_{x\in\mathcal{X}} P(x)\log P(x).$$

Intuitively, entropy measures how dispersed a probability distribution is. For example, a uniform distribution is considered to have very high entropy (i.e. a lot of uncertainty), whereas a distribution that assigns all its mass on a single point is considered to have zero entropy (i.e. no uncertainty). Notably, it can be shown that among continuous distributions over $\mathbb{R}$, the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ has the highest entropy (highest uncertainty) among all possible distributions that have the given mean μ and variance σ^2.

To further solidify our intuition, we present motivation from communication theory. Suppose we want to communicate from a source to a destination, and our messages are always (a sequence of) discrete symbols over space $\mathcal{X}$ (for example, $\mathcal{X}$ could be letters $\{\mathrm{a}, \mathrm{b}, \ldots, \mathrm{z}\}$). We want to construct an encoding scheme for our symbols in the form of sequences of binary bits that are transmitted over the channel. Further, suppose that in the long run the frequency of occurrence of symbols follow a probability distribution $P(X)$. This means, in the long run, the fraction of times the symbol x gets transmitted is $P(x)$.

A common desire is to construct an encoding scheme such that the average number of bits per symbol transmitted remains as small as possible. Intuitively, this means we want very frequent symbols to be assigned to a bit pattern having a small number of bits. Likewise, because we are

[1] If P and Q are densities for continuous-valued random variables, then the sum is replaced by an integral, and everything stated in this problem works fine as well. But for the sake of simplicity, in this problem we'll just work with this form of KL divergence for probability mass functions/discrete-valued distributions.

interested in reducing the average number of bits per symbol in the long term, it is tolerable for infrequent words to be assigned to bit patterns having a large number of bits, since their low frequency has little effect on the long term average. The encoding scheme can be as complex as we desire, for example, a single bit could possibly represent a long sequence of multiple symbols (if that specific pattern of symbols is very common). The entropy of a probability distribution $P(X)$ is its optimal bit rate, i.e., the lowest average bits per message that can possibly be achieved if the symbols $x \in \mathcal{X}$ occur according to $P(X)$. It does not specifically tell us *how* to construct that optimal encoding scheme. It only tells us that no encoding can possibly give us a lower long term bits per message than $H(P)$.

To see a concrete example, suppose our messages have a vocabulary of $K = 32$ symbols, and each symbol has an equal probability of transmission in the long term (i.e, uniform probability distribution). An encoding scheme that would work well for this scenario would be to have $\log_2 K$ bits per symbol, and assign each symbol some unique combination of the $\log_2 K$ bits. In fact, it turns out that this is the most efficient encoding one can come up with for the uniform distribution scenario.

It may have occurred to you by now that the long term average number of bits per message depends only on the frequency of occurrence of symbols. The encoding scheme of scenario A can in theory be reused in scenario B with a different set of symbols (assume equal vocabulary size for simplicity), with the same long term efficiency, as long as the symbols of scenario B follow the same probability distribution as the symbols of scenario A. It might also have occurred to you, that reusing the encoding scheme designed to be optimal for scenario A, for messages in scenario B having a *different probability* of symbols, will always be suboptimal for scenario B. To be clear, we do not need know *what* the specific optimal schemes are in either scenarios. As long as we know the distributions of their symbols, we can say that the optimal scheme designed for scenario A will be suboptimal for scenario B if the distributions are different.

Concretely, if we reuse the optimal scheme designed for a scenario having symbol distribution $Q(X)$, into a scenario that has symbol distribution $P(X)$, the long term average number of bits per symbol achieved is called the *cross entropy*, denoted by $H(P, Q)$:

$$H(P, Q) = -\sum_{x \in \mathcal{X}} P(x) \log Q(x).$$

To recap, the entropy $H(P)$ is the best possible long term average bits per message (optimal) that can be achieved under a symbol distribution $P(X)$ by using an encoding scheme (possibly unknown) specifically designed for $P(X)$. The cross entropy $H(P, Q)$ is the long term average bits per message (suboptimal) that results under a symbol distribution $P(X)$, by reusing an encoding scheme (possibly unknown) designed to be optimal for a scenario with symbol distribution $Q(X)$.

Now, KL divergence is the penalty we pay, as measured in average number of bits, for using the optimal scheme for $Q(X)$, under the scenario where symbols are actually distributed as $P(X)$. It is straightforward to see this

$$\begin{aligned} D_{KL}(P\|Q) &= \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \\ &= \sum_{x \in \mathcal{X}} P(x) \log P(x) - \sum_{x \in \mathcal{X}} P(x) \log Q(x) \\ &= H(P, Q) - H(P). \quad \text{(difference in average number of bits.)} \end{aligned}$$

If the cross entropy between P and Q is $H(P)$ (and hence $D_{KL}(P||Q) = 0$) then it necessarily means $P = Q$. In Machine Learning, it is a common task to find a distribution Q that is "close" to another distribution P. To achieve this, it is common to use $D_{KL}(Q||P)$ as the loss function to be optimized. As we will see in this question below, Maximum Likelihood Estimation, which is a commonly used optimization objective, turns out to be equivalent to minimizing the KL divergence between the training data (i.e. the empirical distribution over the data) and the model.

Now, we get back to showing some simple properties of KL divergence.

(a) [5 points] **Nonnegativity.**
Prove the following:

$$\forall P, Q. \quad D_{KL}(P\|Q) \geq 0$$

and

$$D_{KL}(P\|Q) = 0 \qquad \text{if and only if} \qquad P = Q.$$

[Hint: You may use the following result, called **Jensen's inequality**. If f is a convex function, and X is a random variable, then $E[f(X)] \geq f(E[X])$. Moreover, if f is strictly convex (f is convex if its Hessian satisfies $H \geq 0$; it is *strictly* convex if $H > 0$; for instance $f(x) = -\log x$ is strictly convex), then $E[f(X)] = f(E[X])$ implies that $X = E[X]$ with probability 1; i.e., X is actually a constant.]

(b) [5 points] **Chain rule for KL divergence.**
The KL divergence between 2 conditional distributions $P(X|Y), Q(X|Y)$ is defined as follows:

$$D_{KL}(P(X|Y)\|Q(X|Y)) = \sum_y P(y) \left(\sum_x P(x|y) \log \frac{P(x|y)}{Q(x|y)} \right)$$

This can be thought of as the expected KL divergence between the corresponding conditional distributions on x (that is, between $P(X|Y = y)$ and $Q(X|Y = y)$), where the expectation is taken over the random y.
Prove the following chain rule for KL divergence:

$$D_{KL}(P(X,Y)\|Q(X,Y)) = D_{KL}(P(X)\|Q(X)) + D_{KL}(P(Y|X)\|Q(Y|X)).$$

(c) [5 points] **KL and maximum likelihood.**
Consider a density estimation problem, and suppose we are given a training set $\{x^{(i)}; i = 1, \ldots, n\}$. Let the empirical distribution be $\hat{P}(x) = \frac{1}{n} \sum_{i=1}^{n} 1\{x^{(i)} = x\}$. ($\hat{P}$ is just the uniform distribution over the training set; i.e., sampling from the empirical distribution is the same as picking a random example from the training set.)
Suppose we have some family of distributions P_θ parameterized by θ. (If you like, think of $P_\theta(x)$ as an alternative notation for $P(x; \theta)$.) Prove that finding the maximum likelihood estimate for the parameter θ is equivalent to finding P_θ with minimal KL divergence from $\hat{P}$. I.e. prove:

$$\arg\min_\theta D_{KL}(\hat{P}\|P_\theta) = \arg\max_\theta \sum_{i=1}^{n} \log P_\theta(x^{(i)})$$

2. **[45 points] Semi-supervised EM**

Expectation Maximization (EM) is a classical algorithm for unsupervised learning (*i.e.,* learning with hidden or latent variables). In this problem we will explore one of the ways in which the EM algorithm can be adapted to the semi-supervised setting, where we have some labeled examples along with unlabeled examples.

In the standard unsupervised setting, we have $n \in \mathbb{N}$ unlabeled examples $\{x^{(1)}, \ldots, x^{(n)}\}$. We wish to learn the parameters of $p(x, z; \theta)$ from the data, but $z^{(i)}$'s are not observed. The classical EM algorithm is designed for this very purpose, where we maximize the intractable $p(x; \theta)$ indirectly by iteratively performing the E-step and M-step, each time maximizing a tractable lower bound of $p(x; \theta)$. Our objective can be concretely written as:

$$\begin{aligned}\ell_{\text{unsup}}(\theta) &= \sum_{i=1}^{n} \log p(x^{(i)}; \theta) \\ &= \sum_{i=1}^{n} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)\end{aligned}$$

Now, we will attempt to construct an extension of EM to the semi-supervised setting. Let us suppose we have an *additional* $\tilde{n} \in \mathbb{N}$ labeled examples $\{(\tilde{x}^{(1)}, \tilde{z}^{(1)}), \ldots, (\tilde{x}^{(\tilde{n})}, \tilde{z}^{(\tilde{n})})\}$ where both x and z are observed. We want to simultaneously maximize the marginal likelihood of the parameters using the unlabeled examples, and full likelihood of the parameters using the labeled examples, by optimizing their weighted sum (with some hyperparameter α). More concretely, our semi-supervised objective $\ell_{\text{semi-sup}}(\theta)$ can be written as:

$$\begin{aligned}\ell_{\text{sup}}(\theta) &= \sum_{i=1}^{\tilde{n}} \log p(\tilde{x}^{(i)}, \tilde{z}^{(i)}; \theta) \\ \ell_{\text{semi-sup}}(\theta) &= \ell_{\text{unsup}}(\theta) + \alpha \ell_{\text{sup}}(\theta)\end{aligned}$$

We can derive the EM steps for the semi-supervised setting using the same approach and steps as before. You are *strongly encouraged* to show to yourself (no need to include in the write-up) that we end up with:

E-step (semi-supervised)

For each $i \in \{1, \ldots, n\}$, set

$$Q_i^{(t)}(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta^{(t)})$$

M-step (semi-supervised)

$$\theta^{(t+1)} := \arg\max_{\theta} \left[\sum_{i=1}^{n} \left(\sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i^{(t)}(z^{(i)})} \right) + \alpha \left(\sum_{i=1}^{\tilde{n}} \log p(\tilde{x}^{(i)}, \tilde{z}^{(i)}; \theta) \right) \right]$$

(a) [5 points] **Convergence.** First we will show that this algorithm eventually converges. In order to prove this, it is sufficient to show that our semi-supervised objective $\ell_{\text{semi-sup}}(\theta)$ monotonically increases with each iteration of E and M step. Specifically, let $\theta^{(t)}$ be the parameters obtained at the end of t EM-steps. Show that $\ell_{\text{semi-sup}}(\theta^{(t+1)}) \geq \ell_{\text{semi-sup}}(\theta^{(t)})$.

Semi-supervised GMM

Now we will revisit the Gaussian Mixture Model (GMM), to apply our semi-supervised EM algorithm. Let us consider a scenario where data is generated from $k \in \mathbb{N}$ Gaussian distributions, with unknown means $\mu_j \in \mathbb{R}^d$ and covariances $\Sigma_j \in \mathbb{S}_+^d$ where $j \in \{1, \ldots, k\}$. We have n data points $x^{(i)} \in \mathbb{R}^d, i \in \{1, \ldots, n\}$, and each data point has a corresponding latent (hidden/unknown) variable $z^{(i)} \in \{1, \ldots, k\}$ indicating which distribution $x^{(i)}$ belongs to. Specifically, $z^{(i)} \sim \text{Multinomial}(\phi)$, such that $\sum_{j=1}^{k} \phi_j = 1$ and $\phi_j \geq 0$ for all j, and $x^{(i)}|z^{(i)} \sim \mathcal{N}\left(\mu_{z^{(i)}}, \Sigma_{z^{(i)}}\right)$ i.i.d. So, μ, Σ, and ϕ are the model parameters.

We also have additional $\tilde{n}$ data points $\tilde{x}^{(i)} \in \mathbb{R}^d, i \in \{1, \ldots, \tilde{n}\}$, and an associated *observed* variable $\tilde{z}^{(i)} \in \{1, \ldots, k\}$ indicating the distribution $\tilde{x}^{(i)}$ belongs to. Note that $\tilde{z}^{(i)}$ are known constants (in contrast to $z^{(i)}$ which are unknown *random* variables). As before, we assume $\tilde{x}^{(i)}|\tilde{z}^{(i)} \sim \mathcal{N}\left(\mu_{\tilde{z}^{(i)}}, \Sigma_{\tilde{z}^{(i)}}\right)$ i.i.d.

In summary we have $n + \tilde{n}$ examples, of which n are unlabeled data points x's with unobserved z's, and $\tilde{n}$ are labeled data points $\tilde{x}^{(i)}$ with corresponding observed labels $\tilde{z}^{(i)}$. The traditional EM algorithm is designed to take only the n unlabeled examples as input, and learn the model parameters μ, Σ, and ϕ.

Our task now will be to apply the semi-supervised EM algorithm to GMMs in order to also leverage the additional $\tilde{n}$ labeled examples, and come up with semi-supervised E-step and M-step update rules specific to GMMs. Again, note that we are learning a single set of parameters to model all of the examples, which consists of both the unlabeled and labeled examples. Whenever required, you can cite the lecture notes for derivations and steps.

(b) [4 points] **Semi-supervised E-Step.** Clearly state which are all the latent variables that need to be re-estimated in the E-step. Derive the E-step to re-estimate all the stated latent variables. Your final E-step expression must only involve x, z, μ, Σ, ϕ and universal constants.

(c) [16 points] **Semi-supervised M-Step.** Clearly state which are all the parameters that need to be re-estimated in the M-step. Derive the M-step to re-estimate all the stated parameters. Specifically, derive closed form expressions for the parameter update rules for $\mu^{(t+1)}$, $\Sigma^{(t+1)}$ and $\phi^{(t+1)}$ based on the semi-supervised objective.

(d) [7 points] **Classical (Unsupervised) EM Implementation.** For this sub-question, we are only going to consider the n unlabelled examples. Follow the instructions in `src/semi_supervised_em/gmm.py` to implement the traditional EM algorithm, and run it on the unlabelled data-set until convergence.

Run three trials and use the provided plotting function to construct a scatter plot of the resulting assignments to clusters (one plot for each trial). Your plot should indicate cluster assignments with colors they got assigned to (*i.e.,* the cluster which had the highest probability in the final E-step).

Submit the three plots obtained above in your write-up.

(e) [10 points] **Semi-supervised EM Implementation.** Now we will consider both the labelled and unlabelled examples (a total of $n+\tilde{n}$), with 5 labelled examples per cluster. We have provided starter code for splitting the dataset into matrices `x` and `x_tilde` of unlabelled and labelled examples respectively. Add to your code in `src/semi_supervised_em/gmm.py` to implement the modified EM algorithm, and run it on the dataset until convergence.

Create a plot for each trial, as done in the previous sub-question.

Submit the three plots obtained above in your write-up.

(f) [3 points] **Comparison of Unsupervised and Semi-supervised EM.** Briefly describe the differences you saw in unsupervised *vs.* semi-supervised EM for each of the following:

i. Number of iterations taken to converge.

ii. Stability (*i.e.,* how much did assignments change with different random initializations?)

iii. Overall quality of assignments.

Note: The dataset was sampled from a mixture of three low-variance Gaussian distributions, and a fourth, high-variance Gaussian distribution. This should be useful in determining the overall quality of the assignments that were found by the two algorithms.

3. **[45 points] Variational Inference in a Linear Gaussian Model**

In this problem, we will introduce an algorithm for probabilistic inference in latent variable models. We consider a latent variable model where $p(x, z) = p(z) \cdot p(x \mid z)$ where z is the latent variable and x is the observed variable. We are interested in the following probabilistic inference tasks: *given* a latent variable model and an example x, we wish to determine the marginal distribution $p(x)$ and the posterior distribution $p(z \mid x)$.

(Broader context: latent variable models have many applications. For example, to model the language, the observed variable x can be a document, and the latent variable z can mean the topic of the document. Computing the posterior distribution in this case corresponds to inferring the topic of the document. Moreover, though in this question we always operate with a given latent variable model, computing the posterior in some latent variable model can also be used as a sub-procedure for *learning* the latent variable model (as in the EM algorithm). See the remark at the end of the question as well.)

Specifically, we will introduce and study a particular *approximate* inference algorithm: Stochastic Gradient Variational Bayes (SGVB), which is closely related to the EM algorithm introduced in the lectures.[2] Concretely, consider a latent variable model with latent variables $z \in \mathbb{R}^m$ and observed variables $x \in \mathbb{R}^d$, drawn according to

$$z \sim \mathcal{N}(0, I_m) \tag{1}$$

$$x \mid z \sim \mathcal{N}(f(z), \gamma^2 I_d), \tag{2}$$

where $\gamma > 0$, I_m and I_d are identity matrices of sizes $m \times m$ and $d \times d$ respectively, and $f : \mathbb{R}^m \to \mathbb{R}^d$ maps z to the mean of the conditional Gaussian distribution of x given z.[3] In the subsequent text, we shall refer to the distributions in Eqs. (1) and (2) simply as $p(z)$ and $p(x \mid z)$ respectively. When f is chosen to be a deep neural network, the resulting "deep" latent variable model is capable of modeling highly complex distributions over x.[4]

In this problem, however, we shall consider the simplified case of "Linear Gaussian Model", where f is parameterized by a linear function:

$$f(z) = Wz + b \tag{3}$$

for some given choice of $W \in \mathbb{R}^{d \times m}, b \in \mathbb{R}^d$.

Unlike the case where f is a neural network for which SGVB is the default algorithm nowadays, the Linear Gaussian Model admits a much simpler solution—the marginal distribution $p(x)$ can be exactly and analytically derived, and so can the posterior distribution $p(z \mid x)$. We will use the analytical solution as a gold standard to help us understand the mechanism and behavior of SGVB on this simplified case. (In contrast, it is much more challenging to understand or debug SGVB for nonlinear f due to the complexity of the algorithm and the lack of gold standard.)

Concretely, you will consider both the analytical approach (in parts (a) and (e)) and the SGVB algorithm (in parts (b-d)), and analyze the SGVB algorithm by comparing it with the analytical solution (in part (e)).

[2] You may see that the algorithm is very similar to the variational auto-encoder algorithm introduced in Section 4 of the lecture notes on EM algorithms. However, this question is self-contained, and you do not have to read Section 4 to be able to solve this problem.

[3] Note that the main difference from the mixture of Gaussians model covered in the lecture is that the latent variable here is continuous instead of discrete.

[4] Deep latent variable models, such as the variational autoencoder, are actively studied in generative models research.

(a) [5 points] **Exact marginal inference.** By exploiting the linearity of f, we will first show that $p(x)$ can be determined analytically. We shall do so by explicitly finding a closed-form expression for $p(x)$.

To begin, we make the following observation: letting $\delta \sim \mathcal{N}(0, \gamma^2 I_d)$, we can see that the process for generating x is the same as

$$x = Wz + b + \delta. \tag{4}$$

Since z and δ are Gaussian random variables, x must also be a Gaussian random variable. Thus, $p(x)$ must be a Gaussian distribution $\mathcal{N}(\nu, \Gamma)$ for some choice of mean vector $\nu \in \mathbb{R}^d$ and covariance matrix $\Gamma \in \mathbb{R}^{d \times d}$, where Γ is symmetric and positive definite.

Task: Express ν and Γ as functions of (W, b, γ). You do not need to prove that x is a Gaussian random variable or that $p(x)$ is a Gaussian distribution in your answer; you only need to provide expressions for ν and Γ.

Hint: By definition of the Gaussian distribution parameters, $\nu = \mathbb{E}[x]$ and $\Gamma = \mathrm{cov}(x)$.

(b) [5 points] **Understanding the ELBO.** From Q3a, we see that we can exploit the linearity of f to determine $\ln p(x)$ analytically. In general, however, exact calculation of $\ln p(x)$ is often intractable, and we must develop methods to instead *approximate* $\ln p(x)$. One such method is variational inference, which converts the estimation of $\ln p(x)$ into an optimization problem by using the Evidence Lower Bound (ELBO),

$$\mathrm{ELBO}(x; q) = \mathbb{E}_{z \sim q} \ln \frac{p(x, z)}{q(z)}, \tag{5}$$

where q is some choice of distribution over the space of z. Note that in the equation above $q(t)$ denotes the density of q at t, and $z \sim q$ means a random variable z is sampled from the distribution of q. Crucially, as shown in the lecture, the ELBO always lower bounds $\ln p(x)$,

$$\ln p(x) \geq \mathrm{ELBO}(x; q) = \mathbb{E}_{z \sim q} \ln \frac{p(x, z)}{q(z)}, \tag{6}$$

no matter what choice of q you use. Since this bound holds for any choice of q, we can approximate $\ln p(x)$ by optimizing q over some space of distributions $\mathcal{Q}$,

$$\ln p(x) \geq \max_{q \in \mathcal{Q}} \mathrm{ELBO}(x; q), \tag{7}$$

so that the ELBO is as large as possible (thus giving the best approximation for $\ln p(x)$). We refer to $\mathcal{Q}$ as the variational family, and each $q \in \mathcal{Q}$ as a proposal or variational distribution. Before we discuss how to optimize the ELBO, let us briefly familiarize ourselves with the ELBO by considering two decompositions of the ELBO that exposes its relation to the Kullback-Leibler (KL) divergence.

i. **Task:** Prove that

$$\mathrm{ELBO}(x; q) = \mathbb{E}_{z \sim q} \ln p(x \mid z) - D_{KL}(q \parallel p_z). \tag{8}$$

ii. **Task:** Prove that

$$\mathrm{ELBO}(x; q) = \ln p(x) - D_{KL}(q \parallel p_{z|x}). \tag{9}$$

Note that p_z denotes the distribution of $p(z)$, and $p_{z|x}$ denotes the conditional distribution of $p(z|x)$.

Remark 1: Eq. (8) decomposes the ELBO into a "reconstruction term" and "regularization term": if we think of q as seeking to encode x into a latent code z (stochastically), then the first term $\mathbb{E}_{z\sim q} \ln p(x \mid z)$ can be interpreted as a reconstruction term that measures how much the density $p(x \mid z)$ is concentrated on x when conditioned on $z \sim q$; the second term $D_{KL}(q \parallel p_z)$ regularizes q toward the prior distribution p_z, thus restricting the ability of q to encode x.

Remark 2: Eq. (9) reveals how the ELBO is a lower bound of $\ln p(x)$: the gap between the ELBO and $\ln p(x)$ is exactly quantified by the KL divergence between the distribution q that you have proposed versus the actual posterior $p_{z|x}$. This also means, for a given x, that the ELBO matches $\ln p(x)$ if and only if q matches $p_{z|x}$. Taking a step back, we see that from Eq. (9) that optimizing the ELBO achieves two important goals simultaneously: not only does the ELBO serve as an estimate for the log-likelihood $\ln p(x)$, but q itself also serves as an estimate for the posterior $p_{z|x}$.

(c) [10 points] **Optimizing the ELBO via gradient ascent.** There are many methods for optimizing the ELBO. In this question, we will consider gradient-based Variational Bayes, which tackles the optimization of the ELBO via gradient ascent. Abstractly speaking, if $\mathcal{Q} = \{q_\phi(z) : \phi \in \Phi\}$ is a parametric distribution family where $q_\phi(z)$ is parameterized by ϕ, then we can update the ELBO via gradient ascent using the update rule

$$\phi \leftarrow \phi + \alpha \nabla_\phi \text{ELBO}(x; q_\phi) = \phi + \alpha \nabla_\phi \mathbb{E}_{z\sim q_\phi} \ln \frac{p(x,z)}{q_\phi(z)}, \tag{10}$$

where $\alpha > 0$ is some choice of learning rate. This is a powerful optimization method, since it can be applied as long as the ELBO is differentiable[5]—which means this optimization method is viable even when f is a nonlinear function.

We now return to the linear setting and instantiate the gradient ascent algorithm specifically for the Linear Gaussian Model: given pre-specified choices for (W, b, γ) and some choice of observation x, we are interested in optimizing the ELBO. To simplify our optimization problem, we shall restrict Q to the space of factorized Gaussian distributions $\mathcal{Q} = \{\mathcal{N}(\mu, \text{diag}(\sigma^2)) \colon \mu \in \mathbb{R}^m, \sigma \in \mathbb{R}^m_+\}$. Let q be a shorthand for the distribution $\mathcal{N}(z \mid \mu, \text{diag}(\sigma^2))$ and let $q(z \mid \mu, \sigma)$ denote its density. This allows us to re-express the optimization problem as

$$\max_{\mu,\sigma} \text{ELBO}(x\,; \mu, \sigma), \tag{11}$$

where

$$\text{ELBO}(x\,; \mu, \sigma) = \mathbb{E}_{z\sim q} \ln \frac{p(x,z)}{q(z \mid \mu, \sigma)} \tag{12}$$

For notational simplicity, we shall hide the dependency of q on its parameters (μ, σ) in the subsequent text. Our goal is to solve the optimization problem in Eq. (11) via gradient ascent, so we will need to compute $\nabla_{\mu,\sigma}$ELBO. We shall specifically make use of the ELBO decomposition shown in Eq. (8),

$$\text{ELBO}(x\,; \mu, \sigma) = \mathbb{E}_{z\sim q} \ln p(x \mid z) - D_{KL}(q \parallel p_z), \tag{13}$$

[5]For the ELBO to be differentiable, it suffices that $q_\phi(z)$ is differentiable with respect to ϕ. However, if we further know that $p(x, z)$ is differentiable with respect to z, we can exploit this to estimate the gradient efficiently using the *reparameterization trick*, which we shall explain in the subsequent text.

and tackle $\nabla_{\mu,\sigma}\mathbb{E}_{z\sim q}\ln p(x \mid z)$ and $\nabla_{\mu,\sigma}D_{KL}(q \parallel p_z)$ separately.

i. To simplify your task of determining $\nabla_{\mu,\sigma}D_{KL}(q \parallel p_z)$, we have provided the KL divergence between q and p_z for you as follows,

$$D_{KL}(q \parallel p_z) = \sum_{i=1}^{m}\left[-\ln\sigma_i + \frac{1}{2}(\sigma_i^2 + \mu_i^2 - 1)\right]. \tag{14}$$

Task: Provide closed-form expressions for

$$\nabla_{\mu} D_{KL}(q \parallel p_z) \tag{15}$$
$$\nabla_{\sigma} D_{KL}(q \parallel p_z), \tag{16}$$

in terms of (μ, σ). You do not need to prove Eq. (14).

ii. Calculating the gradient of $\mathbb{E}_{z\sim q}\ln p(x \mid z)$ with respect to the parameters of q is tricky since the expectation is dependent on q. We can circumvent this issue by observing that sampling $z \sim q$ is equivalent to sampling $\epsilon \sim \mathcal{N}(0, I_m)$ and then calculating $z = \mu + \sigma \odot \epsilon$.[6] Thus,

$$\mathbb{E}_{z\sim q}\ln p(x \mid z) = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I_m)}\ln p(x \mid \mu + \sigma \odot \epsilon). \tag{17}$$

Since the RHS expectation is not dependent on the parameters of q, we can push the gradient operator through the expectation as follows,

$$\nabla_{\mu,\sigma}\mathbb{E}_{z\sim q}\ln p(x \mid z) = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I_m)}\nabla_{\mu,\sigma}\ln p(x \mid \mu + \sigma \odot \epsilon). \tag{18}$$

This technique is commonly referred to as the reparameterization trick.[7]
Task: Provide closed-form expressions for

$$\nabla_{\mu}\ln p(x \mid \mu + \sigma \odot \epsilon) \tag{19}$$
$$\nabla_{\sigma}\ln p(x \mid \mu + \sigma \odot \epsilon). \tag{20}$$

in terms of $(x, \gamma, W, b, z, \epsilon)$, where $z = \mu + \sigma \odot \epsilon$.
Hint: First determine the closed-form expression for

$$\nabla_{z}\ln p(x \mid z) \tag{21}$$

in terms of (x, γ, W, b, z). Then, using $z = \mu + \sigma \odot \epsilon$, apply the chain rule to determine $\nabla_{\mu}\ln p(x \mid \mu + \sigma \odot \epsilon)$ and $\nabla_{\sigma}\ln p(x \mid \mu + \sigma \odot \epsilon)$.
Remark: Exact calculation of the expected gradient $\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I_m)}\nabla_{\mu,\sigma}\ln p(x \mid \mu+\sigma\odot\epsilon)$ is often intractable,[8] and must instead be estimated via Monte Carlo sampling (thus the final algorithm's namesake *Stochastic* Gradient Variational Bayes). Although this expectation is actually tractable in the Linear Gaussian Model setting, we shall—for the sake of implementing SGVB faithfully—estimate the expectation via Monte Carlo sampling anyway in the coding problem Q3d.

(d) [15 points] **Coding problem.** Follow the instructions in `src/linear_gaussian/lgm.py` and implement `log_likelihood`, `elbo`, and `sgvb` based on the previous questions. Please note the following important points for your implementation:

[6] Note that the Hadamard product $\odot$ denotes the element-wise multiplication of two same-size arrays.
[7] Further exposition on the reparameterization trick is available in Section 4 of the CS229 course lecture notes on the EM algorithm.
[8] This occurs when $q(z)$ is a complicated distribution, or if f is a complicated nonlinear function.

i. When implementing `elbo`, calculate the ELBO based on the decomposition in Eq. (13). Your implementation should calculate $D_{KL}(q \parallel p_z)$ analytically according to Eq. (14). However, for the expectation $\mathbb{E}_{z\sim q} \ln p(x \mid z)$, use the following Monte Carlo estimator,

$$\mathbb{E}_{z\sim q} \ln p(x \mid z) \approx \frac{1}{N} \sum_{i=1}^{N} \ln p(x \mid z^{(i)}), \tag{22}$$

where $z^{(1:N)}$ are N independent draws from the distribution $q(z)$. The choice of N is given to you by `num_samples` in the code.

ii. Similarly, when implementing `sgvb`, you should implement $\nabla D_{KL}(q \parallel p_z)$ analytically and implement $\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I_m)} \nabla \ln p(x \mid \mu + \sigma \odot \epsilon)$ via Monte Carlo sampling by using

$$\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I_m)} \nabla_{\mu,\sigma} \ln p(x \mid \mu + \sigma \odot \epsilon) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mu,\sigma} \ln p(x \mid \mu + \sigma \odot \epsilon^{(i)}), \tag{23}$$

where $\epsilon^{(1:N)}$ are N independent draws from the distribution $\mathcal{N}(0, I_m)$. The choice of N is given to you by `num_samples` in the code.

iii. Doing gradient updates in the space of the standard deviation parameter σ runs the risk of updating to an invalid choice of σ (i.e., $\sigma \leq 0$). To prevent this, we will instead perform updates in the space of the log-standard deviation $\rho = \ln(\sigma)$, where the natural log is applied element-wise to σ. Your implementation should calculate the gradient of $\text{ELBO}(x\,; \mu, \exp(\rho))$ with respect to the parameters μ and ρ. Since σ is computed via element-wise exponentiation of ρ and

$$\nabla_\rho \sigma = \nabla_\rho \exp(\rho) = \exp(\rho) = \sigma, \tag{24}$$

we note by chain rule that

$$\nabla_\rho D_{KL}(q \parallel p_z) = [\nabla_\sigma D_{KL}(q \parallel p_z)] \odot \sigma \tag{25}$$

$$\nabla_\rho \ln p(x \mid \mu + \sigma \odot \epsilon) = [\nabla_\sigma \ln p(x \mid \mu + \sigma \odot \epsilon)] \odot \sigma. \tag{26}$$

Run `python lgm.py`. The script `lgm.py` runs two experiments:

i. In the first experiment, the code will load a pre-trained Linear Gaussian Model (denoted by model-1 here) along with a specific observed sample $x^{(1)}$. The script then reports $\ln p_{\text{model-1}}(x^{(1)})$ as well as its corresponding ELBO both before and after the stochastic gradient optimization of the proposal distribution q.

ii. The second experiment repeats the first experiment, but with a different pre-trained Linear Gaussian Model (model-2) and a different choice of observed sample $x^{(2)}$. Model-2 loads a different choice of W. We intentionally chose $x^{(2)}$ so that $\ln p_{\text{model-1}}(x^{(1)})$ and $\ln p_{\text{model-2}}(x^{(2)})$ are roughly identical (up to at least the fourth significant figure).

Task: Record the output of `lgm.py` in your write-up.

Remark: According to the theory, the value of the ELBO (with either the initial q, the optimized q, or any other q) is always less than or equal to $\ln p(x)$ for the same example x. In your experiment, however, you use a Monte Carlo estimate of the ELBO, which is stochastic. The variance of this estimator depends on the number of samples N.

(e) [10 points] **Experimental results analysis.** Despite having similar log-likelihoods, i.e. $\ln p_{\text{model-1}}(x^{(1)}) \approx \ln p_{\text{model-2}}(x^{(2)})$, you should observe that the optimized ELBO in Experiment 1, denoted by $\text{ELBO}^*_{\text{model-1}}(x^{(1)})$, is larger than that in Experiment 2, denoted by

$\text{ELBO}^*_{\text{model-2}}(x^{(2)})$. In other words, the optimized ELBO achieves a tighter lower bound in Experiment 1 than in Experiment 2.

In this question, we will analyze why the tightness of the lower bound is different in these two experiments based on the decomposition in Eq. (9),

$$\text{ELBO}(x; q) = \ln p(x) - D_{KL}(q \parallel p_{z|x}).$$

i. To conduct our analysis, we shall take advantage of the linear setting to identify a closed-form expression for the posterior distribution $p(z \mid x)$. Since we are dealing with a Linear Gaussian Model, it turns out that $p(z \mid x)$ is a conditional Gaussian distribution $\mathcal{N}(\mu_x, \Sigma_x)$ for some choice of mean vector $\mu_x \in \mathbb{R}^d$ and covariance matrix $\Sigma_x \in \mathbb{R}^{d \times d}$, where Σ_x is symmetric and positive definite.
Task: Express μ_x and Σ_x as functions of (W, b, γ, x). You are allowed to assume (without proof) the fact that, in the Linear Gaussian Model, (x, z) is jointly Gaussian. You only need to provide expressions for μ_x and Σ_x.
Hint: Given any jointly Gaussian distribution (a_1, a_2) distributed as

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right), \tag{27}$$

the conditional distribution $p(a_1 \mid a_2)$ is Gaussian with mean $\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(a_2 - \mu_2)$ and covariance $\Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$. You do not need to prove this in your answer.

ii. **Task:** Based on your analytic solution for $p(z \mid x)$, analyze the choice of W in Experiment 1 versus Experiment 2 and explain why the gap between the log-likelihood and the optimized ELBO is different in the two experiments.

Final Remark: As alluded to in the second paragraph, computing the posterior can be used as a building block in learning the latent variable model. In this specific case, when W and b are unknown and to be learned, the ELBO becomes a function of W, b and μ, σ: $\text{ELBO}(x; W, b, \mu, \sigma)$. To learn W and b as well, we can alternate between optimizing W, b and μ, σ. (There are other nuances because μ and σ should depend on x. See Section 4 of the EM algorithm lecture notes for more discussion.)