CS 6362 - Advanced Machine Learning

Assignment 3

In this assignment you will implement a Bayesian mixture model using gradient-based variational inference. Specifically, you will implement the method of automatic differentiation variational inference (ADVI for short).

Mixture model parameters are typically found through maximum likelihood estimation. However, point estimates of a mixture model fail to account for uncertainty in the model parameters. We might want to place some doubt on the component-specific mean vectors, and covariance matrices. Perhaps more importantly, we would like to account for uncertainty in (1) the number of components in the mixture, and (2) the mixture weights.

Variational inference gives us an approximation to the posterior distribution over mixture models: we can inspect a collection of models, rather than just one. This has downstream implications on model selection (number of components), generation (for each component, we have a posterior on its model parameters, mean and covariance), as well density estimation (is a given sample in distribution or not).

The code for this assignment can be found here.

Automatic differentiation variational inference

ADVI approaches variational inference by defining a single multivariate Gaussian for the variational distribution. Thus, our variational parameters consist of (1) a mean vector for this distribution, and (2) a covariance matrix for this distribution. These parameters are the subject of optimization.

In ADVI, to approximate the evidence lower bound (ELBO) via MC integration, we take the following steps (see the ADVI paper, and the lecture notes on gradient-based VI):

1. Compute the entropy of the variational distribution - being Gaussian, this has a closed-form expression only involving the covariance parameter.

2. Draw a sample from the variational distribution.
3. Apply a transformation to the sample to ensure it is in the appropriate domain of the model parameter, e.g. a model parameter might be constrained to be strictly positive.
4. Compute the log-determinant of the Jacobian of the inverse transformation, given the above sample. Note: you can also consider the forward transformation & the unconstrained parameter, but for this assignment, the convention will be to stick with the inverse transformation.
5. Compute the log prior for the drawn (constrained) model parameters.
6. Evaluate the log likelihood, given the (constrained) model parameters.

Then, simply combine the above to compute the ELBO. At that point, we can take a gradient, and maximize the ELBO to find the variational parameters.

Simplifying assumptions

We will make two assumptions that considerably simplify the problem.

The posterior factorizes

This is otherwise known as the “mean-field” assumption. That is, the variational distribution can be written as a product of marginals over all variables.

And since the variational distribution is a Gaussian, this implies **the covariance matrix is diagonal**. In practice, you should represent this as a vector of variances, in correspondence with the variational mean parameter.

The priors factorize

In the model, we will associate a prior with each model parameter (e.g. a mean vector, a covariance matrix).

We will assume that **all such priors are independent of one another**.

Model specification

You will implement ADVI in a modular way. Starting out, you will first verify your VI code is correct by approximating the posterior of a simple 1D Gaussian (whose posterior is in fact known). You will then move on to a multivariate Gaussian. And last, a Gaussian mixture model.

The model is decomposed by **likelihood**, **priors**, and **transformations**.

One thing to note: throughout this assignment, with the exception of the variational distribution, Gaussians will be parameterized by **precision** (scalar / matrix) rather than variance, or covariance. In particular, a precision matrix will be identified by its Cholesky factor - a lower triangular matrix, whose diagonal entries must be strictly positive, and (lower) off-diagonal entries can be arbitrary.

Log likelihoods

- You will implement a 1D Gaussian log likelihood, provided a mean parameter and precision parameter.
- You will implement a multivariate Gaussian log likelihood, provided a mean parameter and precision parameter (re: Cholesky factor).
- You will implement a mixture model. **Note**: for numerical stability, it will be necessary to use the log-sum-exp trick. See Ch. 28.2.5.3 in PML-2.

Additional implementation details can be found in the code documentation.

Log priors

- You will implement a Gaussian prior - applicable to any real-valued scalar or a vector. For simplicity, you may assume the precision matrix for the prior is a scaled identity matrix, and thus you need only supply a single scalar representing precision.
- You will implement a gamma prior. This will be applicable to a strictly positive scalar, representing a precision parameter. It contains two prior parameters - shape, and rate.
- You will implement a Wishart prior, applicable to a positive definite matrix representing a precision matrix. It contains two prior parameters - degrees of freedom, and a scale matrix. Note: assume that the supplied scale matrix has already been inverted.
- You will implement a Dirichlet prior, applicable to a discrete distribution (in our case, mixture weights). It contains a single prior parameter. In particular, this presumes a symmetric Dirichlet, whereby its vector of concentration values are identical, all set to the provided scalar.

Note: in your implementation, you should be concerned with the **log** of a prior. For simplicity, you may drop all terms in the log prior that do not depend on the input model parameter.

Transformations

For each transformation, you will be responsible for mapping an unconstrained parameter to its constrained space, as well as computing the log-determinant of the Jacobian of the inverse mapping.

- You will implement a transformation for a precision scalar - this should be strictly positive.

- You will implement a transformation for a precision matrix. What will be provided is a vector of values, from which you will need to separate out the diagonal from the off-diagonal, and map the diagonal to be strictly positive. You should then return the Cholesky factor. See the code documentation for more details.
- You will implement a transformation for the mixture weights. This should return a discrete probability distribution - all values being strictly positive, and summing to 1.

ELBO maximization

The likelihood, priors, and transformations will be combined to allow you to specify a probabilistic model. From here, you can then compute the ELBO, and maximize it, with respect to the variational parameters.

As for the optimization procedure, you should implement the update rule found in the ADVI paper. You should use the paper-suggested defaults for optimization hyperparameters, with the exception of two parameters:

1. The starting step size η.
2. The α term, which controls for the history of gradient updates.

Experiments

The notebooks are organized as follows:

1. Verify that your ADVI implementation is correct for simple problems: a Gaussian likelihood in 1D and 2D.
2. Verify that your mixture model is correct for 2D data, and gain some insight on the choice of hyperparameters.
3. Experiment on real data - build a Bayesian mixture model on MNIST, and explore what is learned by the model.

Simple inference

Please see notebook `simple_inference.ipynb` for this experiment.

Here you will be presented with three datasets.

The first 2 datasets are 1D samples drawn from the same underlying distribution (`X_dense` and `X_sparse`). For both of these datasets, you will be inferring the mean parameter and precision parameter of a Gaussian distribution. To this end, you will build a probabilistic model with a Gaussian likelihood, a Gaussian prior for the mean, and a gamma prior for precision. You should then plot the following:

1. Evaluation of the likelihood at the posterior mean, evaluated over the domain of the data (to compare with the data observations). This should be drawn as an area mark. **Note**: although your code implements the log likelihood, simply apply an exponential to the output to obtain the likelihood.
2. Samples from the posterior distribution of the mean parameter. This should be shown as a histogram.
3. Samples from the posterior distribution of the precision parameter. This should be shown as a histogram.

Comment on your findings: how do the above plots differ between `X_dense` and `X_sparse`? What is the cause of the difference? As mentioned, there is a ground-truth solution for the posterior in this scenario ...

The third dataset is a set of 2D observations. You will be inferring the mean vector, and precision matrix, corresponding to a Gaussian distribution. The probabilistic model is one that has a Gaussian likelihood, a Gaussian prior for the mean vector, and a Wishart distribution for the precision matrix. After ELBO maximization, plot the following:

1. Evaluation of the likelihood at the posterior mean over the 2D domain. A grid of 2D points has been created for you - evaluate the likelihood at these points, and then supply the result, as well as the original observations, to the function `plot_2d_likelihood_with_samples` - this will plot the likelihood as a set of contours over the domain.
2. Show draws from the posterior distribution of the mean parameter, as a scatterplot.
3. Show draws from the posterior distribution of the precision parameter, as a set of contours. Each contour should correspond to a separate model drawn from the posterior - in practice, this is the likelihood evaluated at a set of grid points. Use the function `plot_2d_contour_distribution` to overlay contours for all of these grids.

No need to include discussion for the 2D example - this is intended to help you debug your code, and make sure your implementation is correct.

2D Bayesian mixture model

Please see notebook `simple_mixture.ipynb` for this experiment.

You are presented with 2 datasets in this experiment. For both you will build a Bayesian GMM.

The first dataset (`X_mixture_simple`), you should assemble a probabilistic model that has 5 components, each component consisting of a mean vector and precision matrix, as well as a set of mixture weights. Then, run VI. Last, plot the results with `plot_2d_likelihood_with_samples`. You should show the mixture model corresponding to the posterior mean, as well as several models drawn from the posterior.

Treat this dataset as a way to verify your mixture model is correct - no need to provide discussion.

The second dataset (`X_mixture_complex`) is ... more complex. Assemble the same probabilistic model as described above. But this time, vary the concentration parameter for the Dirichlet prior. Try out: 0.01, 2, 100. For each, plot the results, and inspect the variational distribution for the mixture weights. Explain the findings: what impact does the concentration parameter have on the mixture model?

Bayesian mixture for MNIST

Please see notebook `mnist_mixture.ipynb` for this experiment.

In this experiment you will apply your Bayesian mixture model to the MNIST digits dataset. Now, instead of using the original images, linear projections of the images have been provided to reduce the dimensionality of the data. Supporting code for projecting back to images, and plotting image grids, has been provided for you - use this as a starting point for displaying your results.

First, you should try your Bayesian mixture model out on `X_mnist_2` - images of the number 2. Note: `X_mnist_2_proj` is a projection matrix for mapping back to the image domain, this should be passed in to `plot_image_grid` .

Build a 6-component model, under two different concentration priors - .001, and 100. For each, you should show two kinds of plots:

1. At the posterior mean, show the mean images for each of the components. These should *mostly* look like very clear digits, effectively serving as cluster centers. For images that look a bit noisy, why do we see this? What in the model parameters explains this result?
2. Sample a model from the posterior. Then for each component, sample an image. Show the result as a grid of images. These should look noisier, as they represent the variations in the dataset.

Next, build a model for `X_mnist_evens` - images of numbers 4, 6, and 8. Build a 12-component model, again under the same concentration priors. Show the model in the same way as described above. A good model should:

1. Cluster the images along different digit categories.
2. For a single digit that is assigned to multiple components, we should see different “styles” (e.g. pose, size, etc..).
3. Within a single component, draws should have subtle variations.

I will be looking for these properties when evaluating your submission.

Submission

When submitting your assignment to Brightspace, be sure to include all of your code, and only your code. **Exclude datasets from your submission**. Please prepare your assignment so that I can easily run your notebook.

This assignment is due on **October 28**.

Grading criteria

Bayesian mixture model (70 points)

- Implementation of likelihoods (20 points)
  - 1D Gaussian (5 points)
  - Multivariate Gaussian (5 points)
  - Mixture (10 points)
- Implementation of priors (10 points)
  - Gaussian prior (2 points)
  - Gamma prior (2 points)
  - Wishart prior (4 points)
  - Dirichlet prior (2 points)
- Implementation of transformations (inclusive of log-det Jacobian) (20 points)
  - Precision scalar (5 points)
  - Precision matrix (5 points)
  - Mixture weights (10 points)
- ELBO computation (10 points)
- ELBO maximization (10 points)

Experiments (10 points each, 30 total)

Implementation Tips

You should use pytrees for this assignment.

Please go here for the docs on pytrees.

Although it is not strictly necessary, you will save a *ton of time* if you use pytrees in your implementation. To be more precise, your ELBO function can be implemented in as little as 8 lines of code, while your optimization loop (with the update scheme described in ADVI) can

also be implemented with 8 lines of code. Overall, this assignment should not require that much code, so it is worth investing the time to understand pytrees, and how to use them.

Your probabilistic model will be comprised of a number of parameters that have (1) certain constraints (e.g. positivity), and (2) certain priors associated with them. As a result, it will benefit you to use good data structures in organizing your model.

As an example, suppose you want to represent a single component of your mixture model (or just a single Gaussian, as part of the first experiment). You can create a dictionary, with the mean vector associated with key "mean" and precision matrix associated with key "precision". For a mixture model, you will then have a list of these dictionaries - one per component. And then at the root, you will have a dictionary with two keys: one for the list of components, and the other for your mixture weights.

This resulting data structure is a pytree. And you can use JAX's pytree functionality to process this data structure. Specifically:

- jax.tree.map will iterate over all of the leaves of your data structure, and allow you to specify a function to act on the leaves (e.g. your variational parameters).
- jax.tree.reduce will also iterate over all leaves, and allow you to aggregate their contents. This will be quite helpful for *summing* over derived values (derived via `jax.tree.map`).

Now, the above applies not just to your variational parameters, but also their corresponding (1) transformations, and (2) priors. If they are all in correspondence, then `jax.tree.map` can be used over multiple arguments. So your operations can easily combine variational parameters, with their transformations, to draw constrained samples. And subsequently, evaluate the log prior.

One last tip: represent your variational parameters as a `2 x D` array, with the first row being the mean, and second row being variance (recall: diagonal covariance).