# Homework 8

## Contents

- References
- Instructions
- Problem 1 - Bayesian Linear regression on steroids

## References

- Lectures 27-28 (inclusive).

## Instructions

- Type your name and email in the “Student details” section below.
- Develop the code and generate the figures you need to solve the problems using this notebook.
- For the answers that require a mathematical proof or derivation you should type them using latex. If you have never written latex before and you find it exceedingly difficult, we will likely accept handwritten solutions.
- The total homework points are 100. Please note that the problems are not weighed equally.

▶ Show code cell source

```
# Run this on Google colab
!pip install pyro-ppl
```

```
import pyro
import pyro.distributions as dist
```

Skip to main content

```
from pyro.infer import MCMC, NUTS
import torch
```

## Problem 1 - Bayesian Linear regression on steroids

The purpose of this problem is to demonstrate that we have learned enough to do very complicated things. In the first part, we will do Bayesian linear regression with radial basis functions (RBFs) in which we characterize the posterior of all parameters, including the length-scales of the RBFs. In the second part, we are going to build a model that has an input-varying noise. Such models are called heteroscedastic models.

We need to write some `pytorch` code to compute the design matrix. This is absolutely necessary so that `pyro` can differentiate through all expressions.

```
class RadialBasisFunctions(torch.nn.Module):
    """Radial basis functions basis.

    Arguments:
    X   -  The centers of the radial basis functions.
    ell -  The assumed length scale.
    """
    def __init__(self, X, ell):
        super().__init__()
        self.X = X
        self.ell = ell
        self.num_basis = X.shape[0]
    def forward(self, x):
        distances = torch.cdist(x, self.X)
        return torch.exp(-.5 * distances ** 2 / self.ell ** 2)
```

Here is how you can use them:

```
# Make the basis
x_centers = torch.linspace(-1, 1, 10).unsqueeze(-1)
ell = 0.2
basis = RadialBasisFunctions(x_centers, ell)

# Some points (need to be N x 1)
x = torch.linspace(-1, 1, 100).unsqueeze(-1)

# Evaluate the basis
# Here is the shape of Phi
print(Phi.shape)
```

Here is how they look like:

```
fig, ax = plt.subplots()
for i in range(Phi.shape[1]):
    ax.plot(x, Phi[:, i], label=f"$\phi_{i}$")
ax.set(xlabel="$x$", ylabel="$\phi(x)$")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True);
```

## Part A - Hierarchical Bayesian linear regression with input-independent noise

We will analyze the motorcycle dataset. The data is loaded below.

```
url = "https://github.com/PredictiveScienceLab/data-analytics-se/raw/master/lectureboo
download(url)
```

We will work with the scaled data:

```
from sklearn.preprocessing import StandardScaler

data = np.loadtxt('motor.dat')
scaler = StandardScaler()
data = scaler.fit_transform(data)
X = torch.tensor(data[:, 0], dtype=torch.float32).unsqueeze(-1)
Y = torch.tensor(data[:, 1], dtype=torch.float32)

fig, ax = plt.subplots()
ax.plot(X, Y, 'x')
ax.set(xlabel="$x$", ylabel="$y$")
sns.despine(trim=True);
```

## Part A.I

Your goal is to implement the model described below. We use the radial basis functions

Skip to main content

and maximum of the observed inputs:

$$\phi_i(x;\ell) = \exp\left(-\frac{(x-x_i)^2}{2\ell^2}\right),$$

for $i = 1,\dots,m$. We denote the vector of RBFs evaluated at $x$ as $\boldsymbol{\phi}(x;\ell)$.

We are not going to pick the length-scales $\ell$ by hand. Instead, we will put a prior on it:

$$\ell \sim \text{Exponential}(1).$$

The corresponding weights have priors:

$$w_j|\alpha_i \sim N(0,\alpha_j^2),$$

and its $\alpha_j$ has a prior:

$$\alpha_j \sim \text{Exponential}(1),$$

for $j = 1,\dots,m$.

Denote our data as:

$$x_{1:n} = (x_1,\dots,x_n)^T,\ \text{(inputs)},$$

and

$$y_{1:n} = (y_1,\dots,y_n)^T,\ \text{(outputs)}.$$

The likelihood of the data is:

$$y_i|\mathbf{w},\sigma \sim N(\mathbf{w}^T\boldsymbol{\phi}(x_i;\ell),\sigma^2),$$

for $i = 1,\dots,n$.

$$y_n|\ell,\mathbf{w},\sigma \sim N(\mathbf{w}^T\boldsymbol{\phi}(x_n;\ell),\sigma^2).$$

Skip to main content

**Answer:**

```
def model(X, y, num_centers=50):
    with pyro.plate("centers", num_centers):
        alpha = pyro.sample("alpha", dist.Exponential(1.0))
        # Notice below that dist.Normal needs the standard deviation - not the varianc
        # We follow a different convention in the lecture notes
        w = pyro.sample("w", dist.Normal(0.0, alpha))
    ell = # Complete the code assign to ell the correct prior distribution (an Exponer
    # Hint: Look at alpha.
    sigma = # Complete the code assign to sigma the correct prior distribution (an Exp
    x_centers = torch.linspace(X.min(), X.max(), num_centers).unsqueeze(-1)
    Phi = RadialBasisFunctions(x_centers, ell)(X)
    with pyro.plate("data", X.shape[0]):
        pyro.sample("y", dist.Normal(Phi @ w, sigma), obs=y)
    # Notice that I'm making the model return all the variables that I have made.
    # This is not essential for characterizing the posterior, but it does reduce redur
    # when we are trying to get the posterior predictive.
    return locals()
```

The graph will help to understand the model:

```
pyro.render_model(model, (X, Y), render_distributions=True)
```

Use `pyro.infer.autoguide.AutoDiagonalNormal` to make the guide:

```
guide = pyro.infer.autoguide.AutoDiagonalNormal(model)
```

We will use variational inference. Here is the training code from the hans-on activity:

```
def train(model, guide, data, num_iter=5_000):
    """Train a model with a guide.

    Arguments
    ---------
    model    -- The model to train.
    guide    -- The guide to train.
    data     -- The data to train the model with.
    num_iter -- The number of iterations to train.

    Returns
    -------
    elbos -- The ELBOs for each iteration.
    param_store -- The parameters of the model
```

Skip to main content

```
    pyro.clear_param_store()

    optimizer = pyro.optim.Adam({"lr": 0.001})

    svi = pyro.infer.SVI(
        model,
        guide,
        optimizer,
        loss=pyro.infer.JitTrace_ELBO()
    )

    elbos = []
    for i in range(num_iter):
        loss = svi.step(*data)
        elbos.append(-loss)
        if i % 1_000 == 0:
            print(f"Iteration: {i} Loss: {loss}")

    return elbos, pyro.get_param_store()
```

## Part A.II

Train the model for 20,000 iterations. Call the `train()` function we defined above to do it. Make sure you store the returned elbo values because you will need them later.

**Answer:**

```
# Your code here
```

## Part A.III

Plot the evolution of the ELBO.

**Answer:**

```
# Your code here
```

## Part A.IV

Skip to main content

**Answer:**

I'm giving you this one because it is a bit tricky. You need to use the `pyro.infer.Predictive` class to do it. Here is how you can use it:

```
post_samples = pyro.infer.Predictive(model, guide=guide, num_samples=10)(X, Y)
# Just modify the call to get the right number of samples
```

## Part A.V

Plot the histograms of the posteriors of $\ell$, $\sigma$, $\alpha_{10}$ and $w_{10}$.

**Answer:**

```
# First, here is how to extract the samples.
ell = post_samples["ell"]
# You can do `post_samples.keys()` to see all the keys.
# But they should correspond to the names of the latent variables in the model.
sigma = # Your code here
alphas = # Your code here
ws = # Your code here

# Here is the code to make the histogram for the length scale.
fig, ax = plt.subplots()
# **VERY IMPORTANT** - You need to detach the tensor from the computational graph.
# Otherwise, you will get very very strange behavior.
ax.hist(ell.detach().numpy(), bins=20, alpha=.5)
ax.set(xlabel="$\ell$", ylabel="Frequency")
sns.despine(trim=True);

# Your code for the other histograms here
```

## Part A.VI

Let's extend them model to make predictions.

**Answer:**

```
# Again, I'm giving you most of the code here.
```

Skip to main content

```
    params = model(X, y, num_centers)
    # Here is how you can access the variables
    w = params["w"]
    ell = # Access the length scale
    sigma = # Access the standard deviation of the measurement noise
    x_centers = # Access the centers of the radial basis functions
    # Here are the points where we want to make predictions
    xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
    # Evaluate the basis on the prediction points
    Phi = RadialBasisFunctions(x_centers, ell)(xs)
    # Make the predictions - we use a deterministic node here because we want to
    # save the results of the predictions.
    predictions = pyro.deterministic("predictions", Phi @ w)
    # Finally, we add the measurement noise
    predictions_with_noise = pyro.sample("predictions_with_noise", dist.Normal(predict
    return locals()
```

## Part A.VII

Extract the posterior predictive distribution using 10,000 samples. Separate aleatory and epistemic uncertainty.

**Answer:**

```
# Here is how to make the predictions. Just change the number of samples to the right
post_pred = pyro.infer.Predictive(predictive_model, guide=guide, num_samples=10)(X, Y
# We will predict here:
xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
# You can extract the predictions from post_pred like this:
predictions = post_pred["predictions"]
# Note that we extracted the deterministic node called "predictions" from the model.
# Get the epistemic uncertainty in the usual way:
p_500, p_025, p_975 = np.percentile(predictions, [50, 2.5, 97.5], axis=0)
# Extract predictions with noise
predictions_with_noise = # Your code here
# Get the aleatory uncertainty
ap_025, ap_975 = # Your code here
```

## Part A.VIII

Plot the data, the median, the 95% credible interval of epistemic uncertainty and the 95%

Skip to main content

**Answer:**

```
# Your code here. You have everything you need to make the plot.
```

## Part B - Heteroscedastic regression

We are going to build a model that has an input-varying noise. Such models are called heteroscedastic models. Here I will let you do more of the work.

Everything is as before for $\ell$, the $\alpha_j$'s, and the $w_j$'s. We now introduce a model for the noise that is input dependent. It will use the same RBFs as the mean function. But let's use a different length-scale, $\ell_\sigma$. So, we add:

$$\ell_\sigma \sim \text{Exponential}(1),$$

$$\alpha_{\sigma,j} \sim \text{Exponential}(1),$$

and

$$w_{\sigma,j}|\alpha_{\sigma,j} \sim N(0, \alpha_{\sigma,j}^2),$$

for $j = 1, \dots, m$.

Our model for the input-dependent noise variance is:

$$\sigma(x; \mathbf{w}_\sigma, \ell) = \exp\left(\mathbf{w}_\sigma^T \boldsymbol{\phi}(x; \ell_\sigma)\right).$$

So, the likelihood of the data is:

$$y_i|\mathbf{w}, \mathbf{w}_\sigma \sim N\left(\mathbf{w}^T \boldsymbol{\phi}(x_i; \ell), \sigma^2(x_i; \mathbf{w}_\sigma, \ell)\right),$$

You will implement this model.

Skip to main content

## Part B.I

Complete the code below:

```
def model(X, y, num_centers=50):
    with pyro.plate("centers", num_centers):
        alpha = pyro.sample("alpha", dist.Exponential(1.0))
        w = pyro.sample("w", dist.Normal(0.0, alpha))
        # Let's add the generalized linear model for the log noise.
        alpha_noise = # Your code here
        w_noise = # Your code here
    ell = pyro.sample("ell", dist.Exponential(1.))
    ell_noise = # Your code here
    x_centers = torch.linspace(X.min(), X.max(), num_centers).unsqueeze(-1)
    Phi = RadialBasisFunctions(x_centers, ell)(X)
    Phi_noise = # Your code here
    # This is the new part 2/2
    model_mean = Phi @ w
    sigma = # Your code here (torch.exp(<something>))
    with pyro.plate("data", X.shape[0]):
        pyro.sample("y", dist.Normal(model_mean, sigma), obs=y)
    return locals()
```

Make a `pyro.infer.autoguide.AutoDiagonalNormal` guide:

```
# Your code here
```

Make the graph of the model using `pyro` functionality:

```
# Your code here
```

## Part B.II

Train the model using 20,000 iterations. Then plot the evolution of the ELBO.

**Answer:**

```
# Your code here
```

Skip to main content

## Part B.III

Extend the model to make predictions.

**Answer:**

```
def predictive_model(X, y, num_centers=50):
    params = model(X, y, num_centers)
    w = params["w"]
    w_noise = # Your code here
    ell = # Your code here
    ell_noise = # Your code here
    sigma = # Your code here
    x_centers = params["x_centers"]
    xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
    Phi = # Your code here
    Phi_noise = # Your code here
    predictions = pyro.deterministic("predictions", Phi @ w)
    sigma = # Your code here (pyro.deterministic("sigma", <something>))
    predictions_with_noise = # Your code here
    return locals()
```

## Part B.IV

Now, make predictions and calculate the epistemic and aleatory uncertainties as in part A.VII.

**Answer:**

```
# Your code here
```

## Part B.V

Make the same plot as in part A.VIII.

**Answer:**

```
# Your code here
```

## Part B.VI

Plot the estimated noise standard deviation as a function of of the input along with a 95% credible interval.

**Answer:**

```
# Your code here
```

## Part B.VII

Which model do you prefer? Why?

**Answer:**

## Part B.IX

Can you think of any way to improve the model? Go crazy! This is the last homework assignment! There is no right or wrong answer here. But if you have a good idea, we will give you extra credit.

```
## Your code and answers here
```

Skip to main content

Previous
Homework 7

Next
Bibliography