Bayesian Optimal Design - Weekly Report

Rasmus Hag Løvstad, pgq596

June 17, 2023

Contents

1 Week 1

1.1 Objective

For week 1, we're going to explore the linear regression problem from a Bayesian perspective. First we are going to state essential model assumptions, from which we can examine the consequences of these using Bayes' rule. This will be used to see how we can incorporate previous knowledge into the model and see how observing new data will change our model. While this is apparent for any regression model, we will also demonstrate how we can actually derive an exact posterior parameter distribution in the case of linear regression.
In the end, we are going to implement a simple linear regression model in Python, and explore how the prior model assumptions affect our resulting model.

1.2 Theory

1.2.1 The regression problem and Bayes' rule

First, we are going to state the regression problem in general: Given a dataset $\mathcal{D} = (\mathbf{d}, \mathbf{y})$, where $\mathbf{d} \in \mathbb{R}^{\ell \times d}$ and $\mathbf{y} \in \mathbb{R}^{\ell}$, we wish to find some function $\phi_\theta \in \mathbb{R}^{\ell \times d} \to \mathbb{R}^{\ell}$ such that

$$\phi(\mathbf{d}) \approx \mathbf{y} \tag{1}$$

where ϕ takes some vector of parameters θ.
In the Bayesian approach, we are going to describe the probability distribution of $\mathbf{y}$ with no additional information as $p(\mathbf{y})$, called the *marginal*. If we have observed any data $\mathbf{d}$ and have some set of parameters θ such

that equation 1 is upheld, then we have additional information about the distribution of $\mathbf{y}$. This will be described as the *likelihood* $p(\mathbf{y}|\mathbf{d}, \theta)$.
We might have some prior information on how the parameters θ are distributed - perhaps from prior experiments, domain knowledge or qualified guessing. This can be encoded into the *prior* distribution $p(\theta)$. Given a likelihood, a marginal, and a prior, we can use Bayes' rule to find the *posterior* distribution of θ given the data $\mathbf{d}$ and target values $\mathbf{y}$:

$$p(\theta|\mathbf{d}, \mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{d}, \theta)p(\theta)}{p(\mathbf{y})} \tag{2}$$

What the posterior distribution $p(\theta|\mathbf{d}, \mathbf{y})$ models is the *change of belief* from the prior model $p(\theta)$ given the *evidence* $\mathbf{y}$, $\mathbf{d}$[5]. It gives us a new best bet for the parameters θ, which we can use to produce a better ϕ for equation 1.

1.2.2 The linear regression problem

In the case of linear regression, we assume that ϕ is a linear function, i.e. $\phi(\mathbf{d}) = \mathbf{d}\theta$ where $\theta \in \mathbb{R}^d$. It is also common to augment the dataset with an additional dimension set to 1, such that ϕ also models the intercept such that $\mathbf{d} \in \mathbb{R}^{\ell \times d+1}$ and $\theta \in \mathbb{R}^{d+1}$. In this case θ_{d+1} will then be the intercept.

We are going to assume that the each target value $\mathbf{y}$ can accurately be described as

$$\mathbf{y}^{(i)} = \mathbf{d}^{(i)}\theta + \epsilon \tag{3}$$

for some $\mathbf{d}$, θ, ϵ, where $\epsilon \sim \mathcal{N}(0, \sigma_{\mathbf{y}}^2)$ is some normally distributed noise term. Thus, the likelihood distribution can be described like a normal distribution.

$$p(\mathbf{y}^{(i)}|\mathbf{d}^{(i)}, \theta) = \mathcal{N}(\mathbf{y}^{(i)}; \mathbf{d}^{(i)}\theta, \sigma_{\mathbf{y}}^2) \tag{4}$$

Assuming that the target values are independent, it must follow that

$$p(\mathbf{y}|\mathbf{d}, \theta) = \prod_{i=0}^{\ell} p(\mathbf{y}^{(i)}|\mathbf{d}^{(i)}, \theta) = \mathcal{N}(\mathbf{y}; \mathbf{d}\theta, \sigma_{\mathbf{y}}^2 I_\ell) \tag{5}$$

When modelling the prior information, one could choose to model the parameter distribution as a multivariate gaussian distribution as well

$$p(\theta) = \mathcal{N}(\theta; \mu_\theta, \Sigma_\theta) \tag{6}$$

where μ_θ, Σ_θ are chosen by the model designer.
The last term of equation 2 is the marginal distribution $p(\mathbf{y})$, which is the probability of observing the target values $\mathbf{y}$ without any additional information. This term is difficult to make any reasonable assumptions about, and can thus hard to compute in practice. Lukily, it mostly acts as a normalization term that makes sure that the posterior distribution integrates to 1, so it can for many use cases be safely ignored. As we will see, for a linear regression problems with these assumptions, we will indeed not need it.

1.2.3 An analytical solution for the posterior distribution

Let us outline a possible derivation for the posterior distribution: Given the likelihood and prior, we can describe the joint distribution $p(\theta, \mathbf{y}|\mathbf{d})$ as

$$p(\theta, \mathbf{y}|\mathbf{d}) = p(\mathbf{y}|\mathbf{d}, \theta)p(\theta|\mathbf{d}) = p(\mathbf{y}|\mathbf{d}, \theta)p(\theta) \tag{7}$$

where the last equality is due to the model parameters being independent from the controlled data points.
From this point, we will need to condition the joint distribution on $\mathbf{y}$:

$$p(\theta|\mathbf{y}, \mathbf{d}) = \frac{p(\theta, \mathbf{y}|\mathbf{d})}{p(\mathbf{y})} \tag{8}$$

To perform these steps, we are going to use some useful lemmas about the multivariate normal distribution.

Lemma 1. *Let $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ be a multivariate normal random variable. We can regard these variables in block notation:*

$$\mathbf{X} = \left[\begin{array}{c} \mathbf{X}_1 \\ \hline \mathbf{X}_2 \end{array}\right], \quad \mu = \left[\begin{array}{c} \mu_1 \\ \hline \mu_2 \end{array}\right], \quad \Sigma = \left[\begin{array}{c|c} \Sigma_{11} & \Sigma_{12} \\ \hline \Sigma_{21} & \Sigma_{22} \end{array}\right] \tag{9}$$

then we must have

$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(X_1 - \mu_1), \quad \Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{21}^T \tag{10}$$

Proof. See Krause [5] □

TODO: implement proof

Lemma 2. *For random variables $\mathbf{X} \sim \mathcal{N}(\mu_\mathbf{X}, \Sigma_\mathbf{X})$, $\mathbf{Y}|\mathbf{X} \sim \mathcal{N}(\mu_\mathbf{Y} + A\mathbf{X}, \Sigma_\mathbf{Y})$, for some A, the joint distribution $p(\mathbf{X}, \mathbf{Y})$ is given by*

$$\left[\begin{array}{c} \mathbf{X} \\ \hline \mathbf{Y} \end{array}\right] = \mathcal{N}\left(\left[\begin{array}{c} \mu_\mathbf{X} \\ \hline \mu_\mathbf{Y} + A\mu_\mathbf{X} \end{array}\right], \left[\begin{array}{c|c} \Sigma_\mathbf{X} & \Sigma_\mathbf{X}A^T \\ \hline A\Sigma_\mathbf{X} & A\Sigma_\mathbf{X}A^T + \Sigma_\mathbf{Y} \end{array}\right]\right) \tag{11}$$

Proof. Follows from Lemma 1 and the definition of the multivariate normal distribution. □

If we regard the likelihood from equation 5, we can see that it is only dependent of θ in its mean-term. Thus we can use Lemma 2 to get the joint distribution $p(\theta, \mathbf{y}|\mathbf{d})$:

$$\left[\begin{array}{c} \theta \\ \hline \mathbf{y} \end{array}\right] = \mathcal{N}\left(\left[\begin{array}{c} \mu_\theta \\ \hline \mathbf{d}\theta + \mathbf{d}\mu_\theta \end{array}\right], \left[\begin{array}{c|c} \Sigma_\theta & \Sigma_\theta\mathbf{d}^T \\ \hline \mathbf{d}\Sigma_\theta & \mathbf{d}\Sigma_\theta\mathbf{d}^T + \sigma_\mathbf{y}^2 I_\ell \end{array}\right]\right) \tag{12}$$

To get the posterior distribution, we will need to use the conditioning Lemma 1 on a reordered term:

$$\left[\begin{array}{c} \mathbf{y} \\ \hline \theta \end{array}\right] = \mathcal{N}\left(\left[\begin{array}{c} \mathbf{d}\theta + \mathbf{d}\mu_\theta \\ \hline \mu_\theta \end{array}\right], \left[\begin{array}{c|c} \mathbf{d}\Sigma_\theta\mathbf{d}^T + \sigma_\mathbf{y}^2 I_\ell & \mathbf{d}\Sigma_\theta \\ \hline \Sigma_\theta\mathbf{d}^T & \Sigma_\theta \end{array}\right]\right) \tag{13}$$

With the conditioning lemma, we get the posterior distribution $p(\theta|\mathbf{y}, \mathbf{d}) = \mathcal{N}(\theta; \mu_{\theta|\mathbf{y},\mathbf{d}}, \Sigma_{\theta|\mathbf{y},\mathbf{d}})$:

$$\mu_{\theta|\mathbf{y},\mathbf{d}} = \mu_\theta + \Sigma_\theta\mathbf{d}^T(\mathbf{d}\Sigma_\theta\mathbf{d}^T + \sigma_\mathbf{y}^2 I_\ell)^{-1}(\mathbf{y} - \mathbf{d}\theta + \mathbf{d}\mu_\theta) \tag{14}$$

$$\Sigma_{\theta|\mathbf{y},\mathbf{d}} = \Sigma_\theta - \Sigma_\theta\mathbf{d}^T(\mathbf{d}\Sigma_\theta\mathbf{d}^T + \sigma_\mathbf{y}^2 I_\ell)^{-1}\mathbf{d}\Sigma_\theta \tag{15}$$

Thus we have an analytical expression for the posterior distribution.

1.2.4 Using the posterior to optain model parameters

From this distribution, we can optain model parameters in several ways. A common choice is to pick the θ that maximizes the posterior. This is called the *Maximum-a-posteriori* (MAP) estimate, and will for a normal distribution just be the posterior mean $\mu_{\theta|\mathbf{y},\mathbf{d}}$. For other cases than linear regression, one can also use a mean over samples of θ from the posterior distribution, but for our case, this will converge against the posterior mean $\mu_{\theta|\mathbf{y},\mathbf{d}}$. Another solution is to define a distribution called the *posterior predictive* distribution, that for some new data points $\mathbf{d}_{\text{new}}$ computes the corresponding $\mathbf{y}_{\text{new}}$:

$$p(\mathbf{y}_{\text{new}}|\mathbf{d}_{\text{new}},\mathbf{d},\mathbf{y}) = \int p(\theta|\mathbf{y},\mathbf{d})p(\mathbf{y}_{\text{new}}|\theta,\mathbf{d}_{\text{new}})d\theta \tag{16}$$

Integrals like these are usually solved by sampling - examples of this will be seen later. For now, we can use Lemma 2 to get

$$p(\mathbf{y}_{\text{new}}|\mathbf{d}_{\text{new}},\mathbf{d},\mathbf{y}) = \mathcal{N}(\mathbf{y}_{\text{new}}; \mathbf{d}_{\text{new}}^T\mu_{\theta|\mathbf{y},\mathbf{d}}, \sigma_{\mathbf{y}}^2 + \mathbf{d}_{\text{new}}^T\Sigma_{\theta|\mathbf{y},\mathbf{d}}\mathbf{d}_{\text{new}}) \tag{17}$$

Having the posterior predictive allows us to both sample a $\mathbf{y}$ for a given $\mathbf{d}$, choose the $\mathbf{y}$ that maximises the distribution, and to give us some measure of the inherent uncertancy in the model [5].

1.2.5 Linear regression being invariant to translation

A relevant side-note worth exploring is the effect of the choice of prior mean μ_θ on the posterior mean $\mu_{\theta|\mathbf{y},\mathbf{d}}$.

TODO: Finish this section with information from Oswin - and edit posterior predictive if needed

1.3 Implementation

Implementing a Bayesian linear regression model is fairly simple. To begin with, one needs to decide on a prior distribution for θ, as well as what the variance of the noise $\sigma_{\mathbf{y}}^2$ should be used to generate the example data. In the real world, $\sigma_{\mathbf{y}}^2$ would be measured from observed data, and the prior would be chosen based on the expected distribution of θ based on as much prior knowledge as possible , but for our toy representation, we are going to play around with many different possible priors and noise parameters. For a toy implementation, one can start by generating some data points $\mathbf{d}$ and noise samples ϵ as well as deciding on some true, underlying weight parameters θ_{true}. Then we can compute $\mathbf{y}$ as in equation 3. From this, we can simply implement the posterior distribution as in equation 14 and 15. In Python, this can be done as follows:

```
1 def posterior_distribution(theta, d, y):
2     mu = cov_prior @ d.T @ np.linalg.inv(d @ cov_prior @ d.T
    ↪  + noise * np.eye(l)) @ y
```

```
    cov = cov_prior - cov_prior @ d.T @ np.linalg.inv(d @
    ↪  cov_prior @ d.T + noise * np.eye(l)) @ d @ cov_prior
    return stats.multivariate_normal.pdf(theta, mean=mu,
    ↪  cov=cov)
```

1.4 Results

An example of how 20 samples of θ from the posterior changes with the size of the dataset can be seen in Figure 1. As one can see, the posterior becomes better at estimating the true θ as the number of data points increases, since the lines both become more similar and closer to the mean, as well as how they describe the data better.
An example of how the posterior changes from the prior can be seen in figure 2, where 40 random prior means and covariances are plotted alongside their respective posterior. It can be seen that even when the priors are very different, the posterior usually ends up quite close to the true weight.
An example of how the posterior predictive distribution changes is seen in figure 3. As one can see, the distribution becomes less confident as the noise increases.
Thus we've seen that the Bayesian linear regression model is able to learn the true weight parameters from data, and that it is able to do so even when the prior is very different from the true weight parameters. Now, we will move on to regarding the Bayesian Optimal Design problem.

TODO: Make plots nicer, add "true" line for reference, add legend

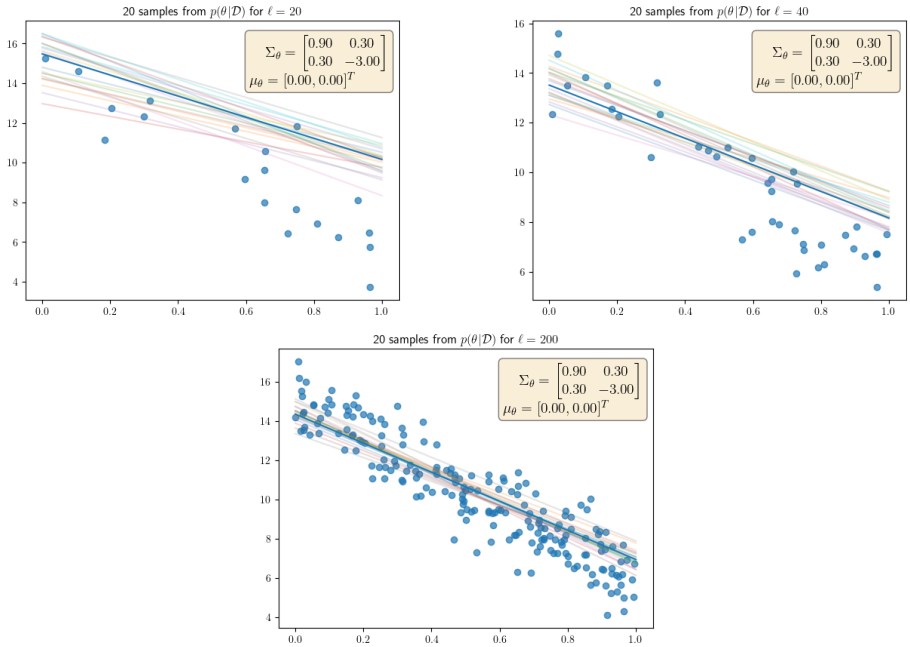
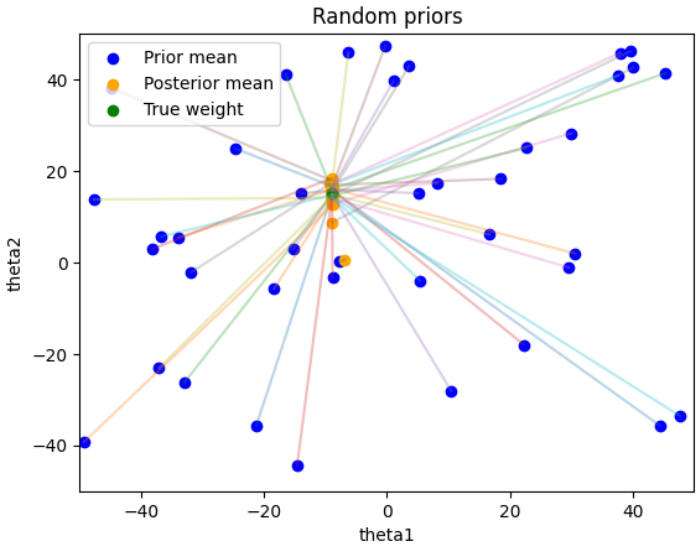
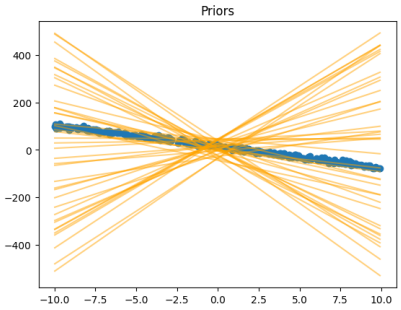


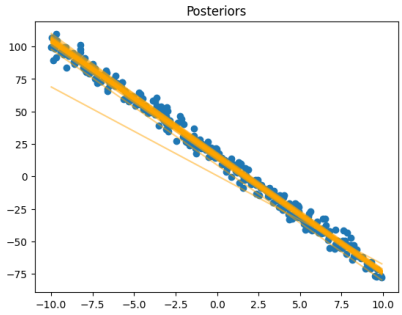
Figure 1: The model trained on data sets with 20, 40 and 200 data points respectively with a fixed prior. The blue line is $\mu_{\theta|\mathcal{D}}$.



(a) Plot showing how the posterior mean changes compared to the prior mean.



(b) Prior mean samples



(c) Posterior mean samples

Figure 2: Priors vs Posteriors for 40 randomly sampled priors on the same data set. Note that the variance is due to random prior covariances.

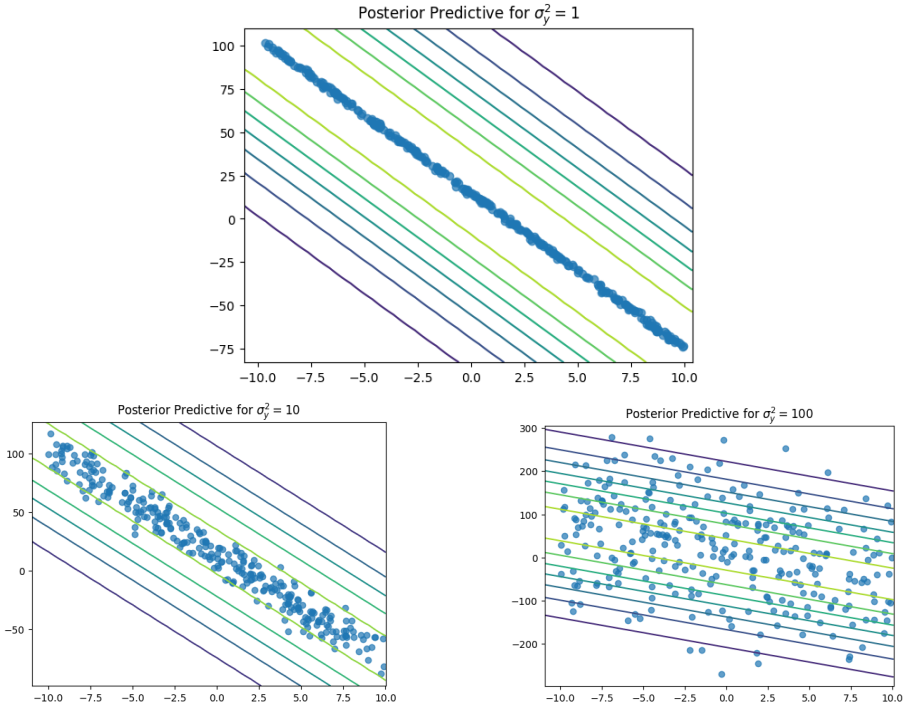


Figure 3: Posterior predictive for different noise levels

2 Week 2

2.1 Objective

For week 2, we are going to regard the *Bayesian Optimal Design* problem for linear regression, and use this to implement a reference implementation, to be used later. We can then also explore the effect of different data sizes, different priors and the difference between estimating an objective function through sampling versus calculating it analytically.

2.2 Theory

2.2.1 Bayesian Optimal Design

Often in scientific contexts as well as other cases, one might have a model that one wishes to strengthen in one way or another using experimental data. Performing the experiments needed to strengthen one's model can be expensive however, so having an efficient strategy to do such can save important ressources. This is where *Bayesian Optimal Design* comes in.

The Bayesian Optimal Design problem is about finding a design $\mathbf{d}$ from a design space $\mathbf{D}$, that optimizes some kind of utility function.[2] For this project, we wish to maximize the expected information gain from the prior to the posterior. To find the optimal design, we want to find a maximizer $\mathbf{d}^*$ defined as such:

$$\mathbf{d}^* = \arg\max_{\mathbf{d}\in\mathbf{D}} I(\mathbf{d}) \tag{18}$$

Where $I(\mathbf{d})$ is the *Mutual Information* between the prior and posterior

when adjusted for data observed at $\mathbf{d}$.

2.2.2 The Nature of the Mutual Information metric

The amount of information of an experiment is often defined as the negative differential entropy defined as such[1]:

$$H(X) = \int_X p(x) \log p(x) dx \tag{19}$$

Thus, the information known before $\mathbf{y}$ is observed from $\mathbf{d}$ is

$$H_1 = \int_\Theta p(\theta) \log p(\theta) d\theta \tag{20}$$

and after is

$$H_2(\mathbf{y}, \mathbf{d}) = \int_\Theta p(\theta|\mathbf{y}, \mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d}) d\theta \tag{21}$$

The gain of information must thus be

$$H_{\text{gain}}(\mathbf{y}, \mathbf{d}) = H_2(\mathbf{y}, \mathbf{d}) - H_1 \tag{22}$$

If we instead regard equation 20 and 21 as expectations we get

$$= \mathbb{E}_\theta[\log p(\theta|\mathbf{y}, \mathbf{d})] - \mathbb{E}_\theta[\log p(\theta)] = \mathbb{E}_\theta[\log(p(\theta|\mathbf{y}, \mathbf{d})) - \log(p(\theta))] \tag{23}$$

Before we perform the experiment, we do not know what the outcome will be. Instead, we'll just regard the expected outcome by taking the expectation over $\mathbf{y}$:

$$\mathbb{E}_\mathbf{y}[H_{\text{grain}}(\mathbf{y}, \mathbf{d})] = \mathbb{E}_\mathbf{y}[\mathbb{E}_\theta[\log(p(\theta|\mathbf{y}, \mathbf{d})) - \log(p(\theta))]] \tag{24}$$

This expression is called the *Mutual Information* between the prior and posterior, and will be denoted $I(\mathbf{d})$. We can put a new interpretation upon this by expanding equation 24 using Bayes' rule:

$$I(\mathbf{d}) = \mathbb{E}_\mathbf{y}[\mathbb{E}_\theta[\log(p(\mathbf{y}|\theta, \mathbf{d})) - \log(p(\mathbf{y}|\mathbf{d}))]] \tag{25}$$

Then we can use that $\frac{p(\mathbf{y}, \theta|\mathbf{d})}{p(\theta)} = p(\mathbf{y}|\theta, \mathbf{d})$:

$$I(\mathbf{d}) = \mathbb{E}_\mathbf{y}[\mathbb{E}_\theta[\log\left(\frac{p(\mathbf{y}, \theta|\mathbf{d})}{p(\theta)}\right) - \log(p(\mathbf{y}|\mathbf{d}))]] = \mathbb{E}_\mathbf{y}[\mathbb{E}_\theta[\log\left(\frac{p(\mathbf{y}, \theta|\mathbf{d})}{p(\theta)p(\mathbf{y}|\mathbf{d})}\right)]] \tag{26}$$

$$= \text{KL}(p(\mathbf{y}, \theta|\mathbf{d})||p(\theta)p(\mathbf{y}|\mathbf{d}))$$

Two random variables X and Y are said to be *independent* if the product of their distributions is the same as their joint distribution i.e.

$$p(X, Y) = p(X)p(Y) \tag{27}$$

Thus, Mutual Information measures how close the prior and the evidence are to be independent. If $\mathbf{d}$ is picked such that the prior has a high proba-

TODO: check if indeed this is evidence

bility of being able to predict $\mathbf{y}|\mathbf{d}$, then the mutual information is going to be close to 0. If, on the contrary, $\mathbf{d}$ is picked such that the prior has a low probability of being able to predict $\mathbf{y}|\mathbf{d}$, the mutual information is going to be large. Thus one could expect that an optimizer would prefer to pick a $\mathbf{d}$ within an area where the prior is not very representative of the underlying generating function. Of course, in an experimental design context we do not have access to this underlying function as we might not be able to simulate experiments accurately. Instead, the expectation expressions in 24 makes it such that we only regard the expected information gain for any given underlying function.

2.2.3 Evaluating the Mutual Information through sampling

Without using any assumptions about the nature of our model or data, we can utilize Monte Carlo sampling to optain an accurate estimate on the expectations in equation 24. For N samples of $\theta \sim p(\theta)$ and M samples of $\mathbf{y} \sim p(\mathbf{y}|\theta, \mathbf{d})$ this thus looks like

$$I(\mathbf{d}) \approx \frac{1}{NM} \sum_{i=0}^{N} \sum_{j=0}^{M} (\log(p(\theta_i|\mathbf{y}_j, \mathbf{d})) - \log(p(\theta_i))) \tag{28}$$

Picking samples of $\mathbf{y}$ nescessitates that we can simulate the experiment however. Instead, we will use the reparamerization trick to sample M samples of $\mathbf{z} \in \mathcal{N}(0, 1)$ such that

$$\mathbf{y}_{ij} = \mu_{\mathbf{y}|\theta,\mathbf{d}} + A_{\mathbf{y}|\theta,\mathbf{d}}\mathbf{z}_j \tag{29}$$

where $A_{\mathbf{y}}$ is a matrix such that $A_{\mathbf{y}}A_{\mathbf{y}}^T = \Sigma_{\mathbf{y}}$. From equation 5 we have that $p(\mathbf{y}|\theta, \mathbf{d}) = \mathcal{N}(\mathbf{y}; \mathbf{d}\theta, \sigma_{\mathbf{y}}^2 I_n)$ so we must have $A_{\mathbf{y}|\theta,\mathbf{d}} = \sigma_{\mathbf{y}}^2 I_n$ thus

$$I(\mathbf{d}) \approx \frac{1}{NM} \sum_{i=0}^{N} \sum_{j=0}^{M} (\log(p(\theta_i|\mathbf{d}\theta_i + \sigma_{\mathbf{y}}^2\mathbf{z}_j, \mathbf{d})) - \log(p(\theta_i))) \tag{30}$$

2.2.4 Evaluating the Mutual Information through analytical solutions

It also happens that when we transform equation 24 using sum of expectations, one can use the known solution to entropy of multivariate normal distributions:

$$I[(\mathbf{y}, \mathbf{d})] = \mathbb{E}_{\mathbf{y}}[\mathbb{E}_{\theta}[\log(p(\theta|\mathbf{y}, \mathbf{d}))]] - \mathbb{E}_{\theta}[\log(p(\theta))] \tag{31}$$

$$= \mathbb{E}_{\mathbf{y}}[\frac{1}{2} \ln \det(2\pi e \Sigma_{\theta|\mathbf{y},\mathbf{d}})] - \frac{1}{2} \ln \det(2\pi e \Sigma_{\theta})$$

It also happens to be that $\Sigma_{\theta|\mathbf{y},\mathbf{d}}$ is independent from $\mathbf{y}$, as we saw last week, thus we get.

$$I[\mathbf{y}, \mathbf{d}] = \frac{1}{2} \ln \det(2\pi e \Sigma_{\theta|\mathbf{y},\mathbf{d}}) - \frac{1}{2} \ln \det(2\pi e \Sigma_{\theta}) \tag{32}$$

2.2.5 Stochastic Optimization

For the sampling approach here, and in the rest of the project, we will use a stochastic gradient descent algorithm to perform the optimization nescessary to solve 18. From some starting point $\mathbf{d}_0$, iteratively update $\mathbf{d}_i$ by

$$\mathbf{d}_i = \mathbf{d}_{i-1} + \alpha \frac{1}{10^c + i \times 10^{-\beta}} \nabla_{\mathbf{d}} I(\mathbf{d}_{i-1}) \tag{33}$$

where $\alpha, \beta, c \in \mathbb{R}$ are hyperparameters that ensures a slow converges to taylor the natural variance that occurs when using Monte Carlo methods.

2.3 Implementation

The mutual information metric can be implemented using the sampling method like so:

```
def mutual_information(d):
  N = 50 # amount of theta samples
  M = 50 # amount of z samples
  thetas = np.random.multivariate_normal(mean_prior,
  ↪  cov_prior, N)
  zs = np.random.randn(M)
  results = []
  for theta in thetas:
    ys = np.array([d @ theta + sigma_y * z for theta in
    ↪  thetas for z in zs])
    for y in ys:
      log_posterior = np.log(posterior_distribution(theta, d,
      ↪  y)) # using posterior_distribution from last week
      log_prior = multivariate_normal.logpdf(theta,
      ↪  mean_prior, cov_prior)
      results.append(log_posterior - log_prior)
  return np.mean(results)
```

And using the analytical solution like so:

```
def mutual_information(d):
  cov_posterior = get_cov_posterior(d) # using method from
  ↪  from last week
  return 0.5 *
  ↪  np.log(np.linalg.det(2*np.pi*np.e*cov_posterior)) - 0.5
  ↪  * np.log(np.linalg.det(2*np.pi*np.e*covariance_prior))
```

Finding the gradient of these solutions can be done using `autograd.grad`. An example of this can be seen in this implementation of the gradient descent algorithm:

```
def optimize(f, d0, alpha, beta, c, iterations):
  d = d0
  g = grad(f)
  for i in range(iterations):
    d = d + alpha / (10**c + i * 10**(-beta)) * g(d)
  return d
```

2.3.1 Optimizing over 2-d matrices

A small, but important note in these implementations is that $\mathbf{d}$ is two-dimensional. To make computation of gradients and optimization much easier, whenever we are in the context of the optimization algorithm, we will regard $\mathbf{d}$ and the gradient as vectors of the shape

$$\mathbf{d}_v = \begin{bmatrix} \mathbf{d}_{1,1} \\ \vdots \\ \mathbf{d}_{1,d} \\ \mathbf{d}_{2,1} \\ \vdots \\ \mathbf{d}_{\ell,d} \end{bmatrix}$$

Whenever we are in the context of Mutual information, we will regard $\mathbf{d}$ as a matrix. The following helper functions can thus be used to flatten and reconstruct the matrices as needed:

```
def encode_d(d): # matrix to vector
    return d.flatten()
def decode_d(encoded_d, dim=2): # vector to matrix
    return encoded_d.reshape(int(len(encoded_d)/dim), dim)
```

2.4 Results

If we regard the problem with $\ell = 10$ datapoints and $d = 2$ dimensions with a zero-prior mean and identity prior covariance, we will data that is closer to the optimum to be spread out wider and more evenly in the parameter space. This can be seen if one regards equation 30. If $\mathbf{d}$ is picked such that $\mathbf{d}\theta_i$ has a high probability of being a small number, then $\mathbf{d}\theta_i + \sigma_{\mathbf{y}}^2 \mathbf{z}_j$ is going to get dominated by the noise term $\sigma_{\mathbf{y}}^2 \mathbf{z}_j$. Thus, the log-posterior $\log p(\theta_i | \mathbf{d}_i + \sigma_{\mathbf{y}}^2 \mathbf{z}_j, \mathbf{d})$ will be smaller. If on the other hand, $\mathbf{d}$ is picked such that $\mathbf{d}\theta_i$ has a high probability of being a large number, then $\mathbf{d}\theta_i$ is going

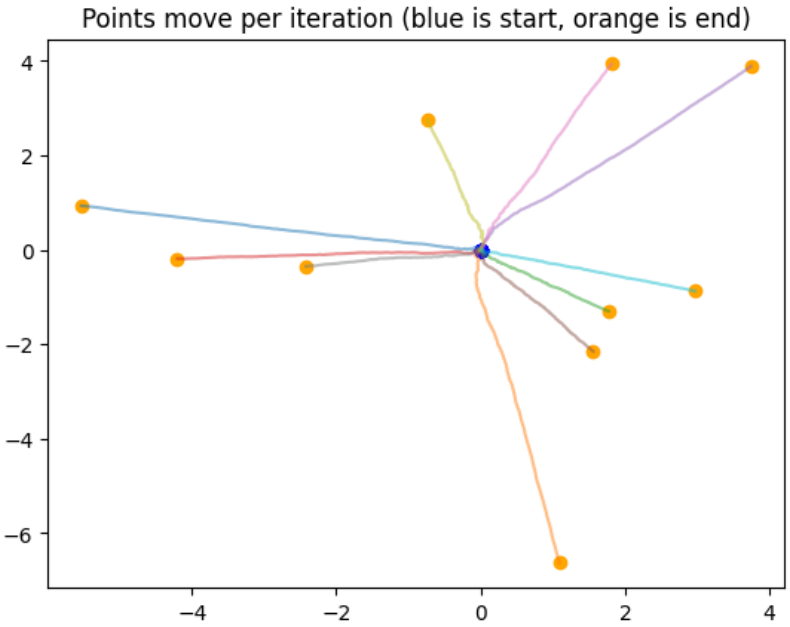


Figure 4: Points moving from their initial position at (0;0) to a more optimal, spread out position.

to dominate the noise term and the posterior has a higher probability of being a large number.
In figure 4, we can see if start position of 11 datapoints are on top of eachother, all at $(0; 0)$, they will swiftly move to a more spread out formation.
Thus we've seen how optimizing over the Mutual Information metric can help find solutions to the Bayesian Optimal Design problem. Now we will move on to exploring how to solve regression problems, even when analytical solutions are not available with the hopes of opening up our Mutual Information optimizer to all kinds of models.

3 Week 3-4

3.1 Objective

This week, we will study and implement the variational inference framework with the objective of estimating the posterior in a way that is indifferent to the type of regression performed. We are going to pose the problem as an optimization problem and then derive an objective function that we can optimize over. We will then try to implement it and perform some adjustments for efficiency and numerical stability.

3.2 Theory

3.2.1 Variational Inference and Variational Families

In week 1, we were able to calculate the exact posterior using the right assumption and conjugacy between the prior and likelihood. This turns

out to be a rarity - in most cases, a closed form solution for the posterior is not readily available. A common approach to this problem is to approximate the posterior distribution, using a *variational distribution* q picked from a family of distributions that we might imagine could approximate the posterior. By picking a suitable objective function, we can use optimization techniques to approximate the posterior. This is called *variational inference*. Thus, our goal is to find a q^* such that for any $\theta \in \Theta$, $q^*(\theta) \approx p(\theta|\mathbf{y}, \mathbf{d})$.

3.2.2 Deriving a suitable objective function

Without any futher assumptions about our variational distribution, an immediate idea could be to use the KL-divergence between q and p as an objective function:

$$q^*(\theta) = \arg\min_q \mathrm{KL}(q(\theta)||p(\theta|\mathbf{y}, \mathbf{d})) \tag{34}$$

Calculating this requries us to calculate the posterior, which we would like to avoid since we are often not guaranteed to be able to compute the evidence term $\log(p(\mathbf{y}|\mathbf{d}))$ from equation 2. We can however rewrite the KL-divergence as follows [3]:

$$\mathrm{KL}(q(\theta)||p(\theta|\mathbf{y}, \mathbf{d})) = \mathbb{E}_{q(\theta)}[\log q(\theta)] - \mathbb{E}_{q(\theta)}[\log p(\theta|\mathbf{y}, \mathbf{d})] \tag{35}$$

and by conditioning

$$= \mathbb{E}_{q(\theta)}[\log q(\theta)] - \mathbb{E}_{q(\theta)}[\log p(\theta, \mathbf{y}|\mathbf{d})] + \log p(\mathbf{y}|\mathbf{d}) \tag{36}$$

One can now notice that the last term is actually constant with regards to q. Thus, when we optimize, we can ignore it. The remaining terms form the negative *evidence lower bound* (ELBO) [3]:

$$\mathrm{ELBO}(q) = \mathbb{E}_{q(\theta)}[\log p(\theta, \mathbf{y}|\mathbf{d})] - \mathbb{E}_{q(\theta)}[\log q(\theta)] \tag{37}$$

Since the KLD consists of the negative ELBO and a constant, we can minimize the KLD by maximizing the ELBO. Thus, we can reformulate our objective:

$$q^*(\theta) = \arg\max_q \mathrm{ELBO}(q) \tag{38}$$

We can derive equation 37 to a more easily computable term:

$$\mathrm{ELBO}(q) = \mathbb{E}_{q(\theta)}[\log(p(\mathbf{y}|\theta, \mathbf{d})p(\theta|\mathbf{d}))] - \mathbb{E}_{q(\theta)}[\log q(\theta)] \tag{39}$$

$$= \mathbb{E}_{q(\theta)}[\log p(\mathbf{y}|\theta, \mathbf{d})] + \mathbb{E}_{q(\theta)}[\log p(\theta|\mathbf{d})] - \mathbb{E}_{q(\theta)}[\log q(\theta)] \tag{40}$$

Using that θ is independent of $\mathbf{d}$, we have

$$= \mathbb{E}_{q(\theta)}[\log p(\mathbf{y}|\theta, \mathbf{d})] - (-\mathbb{E}_{q(\theta)}[\log p(\theta)] + \mathbb{E}_{q(\theta)}[\log q(\theta)]) \tag{41}$$

$$= \mathbb{E}_{q(\theta)}[\log p(\mathbf{y}|\theta, \mathbf{d})] - \mathbb{E}_{q(\theta)}[\log(\frac{q(\theta)}{p(\theta)})] \tag{42}$$

Using the definition of the KLD:

$$= \mathbb{E}_{q(\theta)}[\log p(\mathbf{y}|\theta, \mathbf{d})] - \mathrm{KL}(q||p) \tag{43}$$

Thus, this variational inference problem can work for any posterior, as long as one can calculate the expectation of the likelihood, and the KL-divergence between the variational distribution and the prior.

TODO: rewrite expectations as $\theta \sim q$

3.2.3 The Linear Regression Case

Let us now look at how one can find a good variational family for the linear regression case. An easy guess would be to pick q from a family of multivariate gaussian distributions, since we know from conjugacy between the prior and likelihood that the posterior must also be multivariate gaussian. Thus, one can expect q to be able to estimate p perfectly.
This has the nice consequence of turning the KL-divergence into a closed form expression, since both q and p follow multivariate normal distributions.
We do have a problem, however: The first expectation of 43 is an integral without a closed-form solution. Thus, we wish to approximate it using Monte Carlo integration. To be able to sample θs from q, we will need to use the reparameterization trick: By property of the multivariate normal distribution, there must exist some matrix $\mathbf{A}$ such that $\theta \sim \mathcal{N}(\mu, \mathbf{A}\mathbf{A}^T)$ [5] and such that

TODO: add reference to closed form expression

$$\theta = \mu + \mathbf{A}\mathbf{z}, \quad \text{where } \mathbf{z} \sim \mathcal{N}(0, I_d) \tag{44}$$

Thus, we can simply sample N samples of $\mathbf{z}$ and calculate θ from that. This means that our ELBO estimate can be written as follows:

$$\text{ELBO}(q) \approx \frac{1}{N}\sum_{i=0}^{N}[\log p(\mathbf{y}|\mu + \mathbf{A}\mathbf{z}^{(i)}, \mathbf{d})] - \text{KL}(q||p) \tag{45}$$

μ and $\mathbf{A}$ are then the parameters of q that we will try to find through optimization.

3.3 Implementation

3.3.1 Implementing the log-pdf of multivariate normal distribution

Optimizing over this ELBO objective function means calculating the gradient of it. We will again use autograd.grad to do this, which nescessitates implementing the log-pdf of the multivariate normal distribution by hand, since the one supplied by autograd.scipy.stats does not handle the case where the mean and covariance carries information about the computation graph.
The log-pdf of the multivariate normal distribution is given by:

$$\log \mathcal{N}(x; \mu, \Sigma) = -\frac{1}{2}(n\log(2\pi) + \log(\det(\Sigma)) + (x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{46}$$

3.3.2 Main implementation

The main work of the algorithm is in calculating the ELBO. This can then be plugged into the same optimizer as in week 2. The ELBO can be implemented as follows in Python:

```
1 def ELBO(d, y, mean, A): # optimizing for mean, A
2   zs = np.random.normal(size=(N, len(mean))) # N samples of
    ↪  size d
```

```
  likelihood_samples = []
  for z in zs:
    theta = mean + A @ z
    likelihood_samples.append(log_likelihood(y, theta, d))
  return 1/N * np.sum(likelihood_samples, axis=0) - KLD(mean,
  ↪  A, prior_mean, prior_A)
```

where the log-likelihood is implemented after equation 5:

```
def log_likelihood(y, theta, d):
  return log_pdf(y, theta @ d, noise * np.eye(len(theta)))
```

3.3.3 Optimizing over a mean and a matrix

Like in week 2, it is not trivial to optimize over both a mean and a matrix. Thus, we will encode μ and $\mathbf{A}$ as a vector $\mathbf{v}$ of the shape

$$\mathbf{v} = \begin{bmatrix} \mu_0 \\ \vdots \\ \mu_d \\ \mathbf{A}_{1,1} \\ \vdots \\ \mathbf{A}_{1,d} \\ \mathbf{A}_{2,1} \\ \vdots \\ \mathbf{A}_{d,d} \end{bmatrix}$$

whenever we need to take the gradient or regard them in the context of the optimizer. The following help functions can help encode and decode μ and $\mathbf{A}$:

```
def encode_q_params(q_params): # mean, A to vector
    mean, A = q_params
    return np.array(list(mean) + list(A.flatten()))
def decode_q_params(encoded_q, dim = 3): # vector to mean, A
    shape = len(encoded_q)
    A_shape = (int(np.sqrt(shape - dim)), int(np.sqrt(shape -
    ↪  dim)))
    mean = encoded_q[0:dim]
    A = encoded_q[dim:shape].reshape(A_shape)
    return mean, A
```

3.4 Results

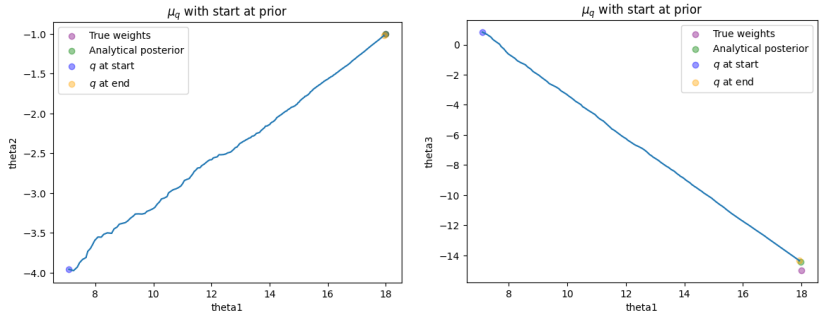


Figure 5: Change in mean of q after 500 iterations of the algorithm with $N = 10$, $\ell = 100$.

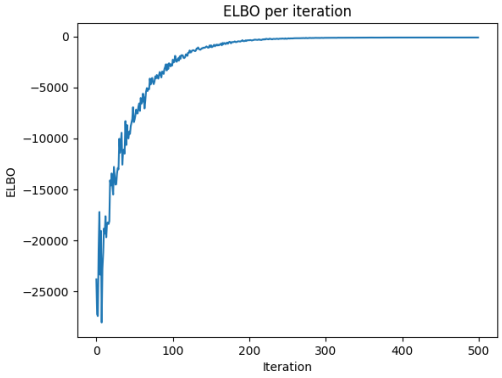


Figure 6: ELBO after each iteration of the algorithm with $N = 10$, $\ell = 100$.

As one can see in figure **??**, the mean of q confidently moves towards the analytical posterior (and true weights). It can also been seen in figure **??** that the ELBO converges after 200 or so iterations, indicating that we have indeed found an optimum.

3.5 Evaluation

The algorithm presented here works as expected, and functions as a possible replacement for the analytical posterior. We've been able to utilize the ELBO as a proxy for the KL-divergence and created a framework that can fitted to many different kind of regression problems, as long as one can estimate the log-likelihood and compute the KL-divergence between the variational distribution and the prior.

4 Week 5-6

4.1 Objective

For the final section of the project, we will try to see how to combine the Bayesian Optimal Design problem from week 2 with the variational

inference framework from week 3-4, leading to a general framework for Bayesian Optimal Design that scales to many different kinds of models.

4.2 Theory

4.2.1 Finding the gradient of the Mutual Information

In week 2, we neatly assumed that autograd.grad was able to find the gradient of the mutual information with respect to $\mathbf{d}$. When one uses variational inference to compute the posterior, this becomes nontrivial, since one needs to differentiate through the parameters as a function of the design $\mathbf{d}$ [4]. We can see this from equation 24:

$$I(\mathbf{d}) = \mathbb{E}_{\mathbf{y}}[\mathbb{E}_{\theta}[\log(p(\theta|\mathbf{y},\mathbf{d})) - \log p(\theta)]] \tag{47}$$

When using variational inference, this changes to

$$I(\mathbf{d}) \approx \mathbb{E}_{\mathbf{y}}[\mathbb{E}_{\theta}[\log(q^*(\theta)) - \log p(\theta)]], \text{ where } q^*(\theta) = \arg\max_q \text{ELBO}_{\mathbf{d},\mathbf{y}}(q) \tag{48}$$

From now on, we will explicitly denote what $\mathbf{d}$ and $\mathbf{y}$ the ELBO is taken with regards to. Taking the gradient of (48) results in

$$\begin{aligned} \nabla_{\mathbf{d}} I(\mathbf{d}) &= \nabla_{\mathbf{d}} \mathbb{E}_{\mathbf{y}\sim q^*}[\mathbb{E}_{\theta\sim q^*}[\log(q^*(\theta)) - \log p(\theta)]] \\ &\approx \mathbb{E}_{\epsilon}[\mathbb{E}_{\mathbf{z}}[\nabla_{\mathbf{d}} \log(q^*_{\mathbf{y}',\mathbf{d}}(\theta'))]], \\ \text{where } \mathbf{y}' = \mathbf{d}\theta' + \epsilon,\ \theta' = \mu_\theta + \mathbf{A}_\theta \mathbf{z},\ \epsilon \sim \mathcal{N}(0, \sigma^2_{\mathbf{y}} I_N),\ \mathbf{z} \sim \mathcal{N}(0, I_M) \end{aligned} \tag{49}$$

Using the chain rule gives us

$$= \mathbb{E}_{\epsilon}[\mathbb{E}_{\mathbf{z}}[\frac{1}{q^*_{\mathbf{y}',\mathbf{d}}(\theta')} \times \nabla_{\mathbf{d}} q^*_{\mathbf{y}',\mathbf{d}}(\theta'))]], \tag{50}$$

Thus we need to calculate $\nabla_{\mathbf{d}} q^*_{\mathbf{y}',\mathbf{d}}(\theta')$, or more specifically

$$\nabla_{\mathbf{d}} q^*_{\mathbf{y}',\mathbf{d}}(\theta') = \nabla_{\mathbf{d}} \arg\max_q \text{ELBO}_{\mathbf{d},\mathbf{y}'}(q)\Big|_{\theta} \tag{51}$$

If we let λ be the parameters of q encoded as a vector, we can regard q^* as a function of $\mathbf{d}$. Thus we can use the chain rule.

$$\nabla_{\mathbf{d}} q^*_{\mathbf{y}',\mathbf{d}}(\theta') = \nabla_{\lambda} q^{\lambda}_{\mathbf{y}',\mathbf{d}}(\theta') \underbrace{\nabla_{\mathbf{d}} \arg\max_{\lambda} \text{ELBO}_{\mathbf{d},\mathbf{y}'}(q^{\lambda})}_{\text{Jacobian}} \tag{52}$$

The gradient, $\nabla_{\lambda} q^{\lambda}_{\mathbf{y}',\mathbf{d}}$, is simple to perform using automatical differentiation. The Jacobian of the variational parameters with regards to the design $\mathbf{d}$, needs very careful consideration. We will approach this using the Implicit Function Theorem [4]:

Theorem 1. *Let f be a continuously differentiable function from $\mathbb{R}^n \times \mathbb{R}^m$ to $\mathbb{R}^m$. Fix a point (a, b) such that $f(a, b) = \mathbf{0}$. If the Jacobian $J^f_b(a, b)$ is invertible, then there must exist an open set $U \subseteq \mathbb{R}^n$ containing a such that there exists a continuously differentiable function $g : U \to \mathbb{R}^m$ such that $g(a) = b$ and $f(x, g(x)) = \mathbf{0}$ for all $x \in U$. Futhermore,*

$$\nabla_x g(x) = -[J^f_y(x, g(x))]^{-1} J^f_x(x, g(x))$$

We will now adapt Theorem 1 to our problem. From now on, every mention of ELBO is with regards to $\mathbf{y} = \theta\mathbf{d} + \epsilon$ from (49) . Let $x = \mathbf{d}$, $y = \lambda$, $g(\mathbf{d}) = \arg\max_\lambda \text{ELBO}_\mathbf{d}(q^\lambda)$, $f(\mathbf{d}, \lambda) = \nabla_\lambda \text{ELBO}_\mathbf{d}(q^\lambda)$.
First, we need to fix a point $(\mathbf{d}', \lambda')$ such that $\nabla_\lambda \text{ELBO}_{\mathbf{d}'}(q^{\lambda'}) = 0$. If we take any $\mathbf{d}'$, and pick λ' to be any local optimum $\lambda' = g(\mathbf{d})$, then the gradient must be 0 at that point. If the hessian then is invertible, we must have:

$$\nabla_\mathbf{d} \arg\max_\lambda \text{ELBO}_\mathbf{d}(q^\lambda)\Big|_\theta = -[\underbrace{\nabla^2_\lambda \text{ELBO}_\mathbf{d}(q^*)}_{\text{variational hessian}}]^{-1} \underbrace{\nabla_\lambda \nabla_\mathbf{d} \text{ELBO}_\mathbf{d}(q^*)}_{\text{variational mixed partials}} \tag{53}$$

Computing both the variational hessian and the variational mixed partials should then be relatively simple using automatic differentiation.

4.2.2 Adapting to sampling

Again, we will approach calculating the expectations using Monte Carlo sampling. From (50), we will instead compute

$$\nabla_\mathbf{d} I(\mathbf{d}) \approx \sum_i^N \sum_j^M \frac{1}{q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j)} \times \nabla_\mathbf{d} q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j) \tag{54}$$

where $\mathbf{y}_{ij} = \mathbf{d}\theta_j + \epsilon_i,\ \theta_j = \mu_\theta + \mathbf{A}_\theta \mathbf{z}_j,\ \epsilon_i \sim \mathcal{N}(0, \sigma^2_\mathbf{y} I_\ell),\ \mathbf{z} \sim \mathcal{N}(0, I_M)$

To try to keep numerically stable, we will use the log-sum-exp trick:

$$= \sum_i^N \sum_j^M \exp(\log \nabla_\mathbf{d} q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j) - \log q^*_{\mathbf{d},\mathbf{y}_{ij}}(\theta_j) I_\ell) \tag{55}$$

4.3 Implementation

If, for every sample in (55), one needs to calculate a new q^*, then at least $N \times M$ q-optimizations are needed. Each q-optimization then requires at most T iterations of the optimization algorithm and L evaluations of the ELBO. At last, the outer optimization algorithm requires at most R iterations. Thus, the total number of evaluations of the inner part of (45) is $R \times N \times M \times T \times L$, which when perfomed as loops in Python is very slow. To help this, one can implement batching to reduce the amount of for-loop iterations, improving the computation time substantially.

4.4 old stuff

This is a working draft and should be sectioned into a more readable format, as well as have made notation consistent Let us first regard our mutual information objective function from week 2:

$$MI(\mathbf{d}) = \int_\Theta \int_\mathbf{Y} p(\theta, \mathbf{y}|\mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d}) d\mathbf{y} d\theta - \int_\Theta p(\theta) \log p(\theta) d\theta$$

Since we are optimizing, let us throw away the second term, since it is constant in terms of $\mathbf{d}$:

$$= \int_{\Theta} \int_{\mathbf{Y}} p(\theta, \mathbf{y}|\mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d}) d\mathbf{y} d\theta$$

To optimize the mutual information, we will need the derivative of it in terms of $\mathbf{d}$:

$$\frac{\partial}{\partial \mathbf{d}} MI(\mathbf{d}) = \frac{\partial}{\partial \mathbf{d}} \int_{\Theta} \int_{\mathbf{Y}} p(\theta, \mathbf{y}|\mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d}) d\mathbf{y} d\theta$$

$$= \int_{\Theta} \int_{\mathbf{Y}} \frac{\partial}{\partial \mathbf{d}} p(\theta, \mathbf{y}|\mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d}) d\mathbf{y} d\theta$$

Let us then use the product rule

TODO: maybe change left derivative

$$= \int_{\Theta} \int_{\mathbf{Y}} (\frac{\partial}{\partial \mathbf{d}} p(\theta, \mathbf{y}|\mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d})) + (p(\theta, \mathbf{y}|\mathbf{d}) \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d})) d\mathbf{y} d\theta$$

Now, let us use the fact that $\frac{\partial}{\partial \mathbf{d}} p(\theta, \mathbf{y}|\mathbf{d}) = p(\theta, \mathbf{y}|\mathbf{d}) \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d})$

TODO: prove this lemma

$$= \int_{\Theta} \int_{\mathbf{Y}} p(\theta, \mathbf{y}|\mathbf{d}) \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d}) \log p(\theta|\mathbf{y}, \mathbf{d}) + p(\theta, \mathbf{y}|\mathbf{d}) \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d}) d\mathbf{y} d\theta$$

$$= \int_{\Theta} \int_{\mathbf{Y}} p(\theta, \mathbf{y}|\mathbf{d}) \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d}) (\log p(\theta|\mathbf{y}, \mathbf{d}) + 1) d\mathbf{y} d\theta$$

$$= \int_{\Theta} \int_{\mathbf{Y}} p(\mathbf{y}|\theta, \mathbf{d}) p(\theta) \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d}) (\log p(\theta|\mathbf{y}, \mathbf{d}) + 1) d\mathbf{y} d\theta$$

Solving this double integral can be hard. Let us consider it as an expectation of the form

$$\mathbb{E}[f(\theta, \mathbf{y})] = \int_{(\theta, \mathbf{y})} p(\theta, \mathbf{y}) f(\theta, \mathbf{y}) d(\theta, \mathbf{y})$$

with $p(x) = p(\mathbf{y}|\theta, \mathbf{d}) p(\theta)$ and $f(x) = \frac{\partial}{\partial \mathbf{d}} \log p(\theta|\mathbf{y}, \mathbf{d}) (\log p(\theta|\mathbf{y}, \mathbf{d}) + 1)$. We can then approximate this expectation by sampling by reducing the expectation to:

$$\mathbb{E}[f(x)] \approx \frac{1}{N} \sum_{i=0}^{N} f(\theta_i, \mathbf{y}_i), \quad (\theta_i, \mathbf{y}_i) \sim p(\theta_i, \mathbf{y}_i)$$

which leads to

TODO: Figure out specific notation here

$$\frac{\partial}{\partial \mathbf{d}} MI(\mathbf{d}) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial \mathbf{d}} \log p(\theta_i|\mathbf{y}_i, \mathbf{d}) (\log p(\theta_i|\mathbf{y}_i, \mathbf{d}) + 1)$$

where $(\theta_i, \mathbf{y}_i) \sim p(\mathbf{y}_i|\theta_i, \mathbf{d}) p(\theta_i)$. Sampling θ_i is easy from our prior, and we can do reparameterization to sample $\mathbf{y}_{ij} = \theta_i^T \mathbf{d} + z_j$ where $z_j \sim \mathcal{N}(0, \sigma_{\mathbf{y}}^2)$.

4.4.1 Finding the gradient of the posterior

Notation:
$\vartheta = \mu_\theta$ and A_θ for use in q_ϑ
$\mathbf{y_d} = \mathbf{y}$ calculated from $\mathbf{d}$.
Now, let us consider the posterior $p(\theta_i|\mathbf{y}, \mathbf{d})$. This is the distribution that we try to approximate when performing variational inference. Thus we can expect our variational distribution $q(\theta_i)$ to reasonably approximate it after our inference algorithm has run. We will denote the optimal parameters found $\vartheta^*(\mathbf{d}, \mathbf{y_d}) = \arg\max_\vartheta \text{ELBO}_{\mathbf{d},\mathbf{y(d)}}(q_\vartheta)$ such that $q_{\vartheta^*}(\theta_i) \approx p(\theta_i|\mathbf{y_d}, \mathbf{d})$.
In our refactored expression for mutual information, we have a term containing $\frac{\partial}{\partial \mathbf{d}} \log p(\theta_i|\mathbf{y_d}, \mathbf{d})$.

$$\frac{\partial}{\partial \mathbf{d}} \log p(\theta_i|\mathbf{y_d}, \mathbf{d}) \approx \frac{\partial}{\partial \mathbf{d}} \log q_{\vartheta^*}(\theta_i)$$

Since ϑ^* is a function of $\mathbf{d}$, and q^* is a function of θ^*, then we can use the chain rule.

$$= \frac{\partial}{\partial \vartheta^*} \log q_{\vartheta^*}(\theta_i) \frac{\partial}{\partial \mathbf{d}} \vartheta^*(\mathbf{y_d}, \mathbf{d})$$

4.4.2 Using The Implicit Function Theorem for finding the indirect gradient

Let $\mathcal{D}$ be ($\mathbf{d}$, $\mathbf{y}$) encoded in some vector.
If for some $(\mathcal{D}', \vartheta')$, $\frac{\partial}{\partial \vartheta} \text{ELBO}_{\mathcal{D}'}(q_\vartheta)\Big|_{\mathcal{D}=\mathcal{D}', \vartheta=\vartheta'} = 0$ and the Jacobian is invertible, then there exists an open set of datapoints $\mathcal{D} \in \mathcal{X} \times \mathcal{Y}$ such that there exists a function $\vartheta^* \colon \mathcal{X} \times \mathcal{Y} \to \Theta$ such that

TODO: add reference! very similar to litterature

$$\vartheta^*(\mathcal{D}') = \vartheta' \text{ and } \forall \mathcal{D} \in \mathcal{X} \times \mathcal{Y}, \frac{\partial}{\partial \vartheta} \text{ELBO}_{\mathcal{D}}(q_\vartheta)\Big|_{\mathcal{D}, \vartheta^*(\mathcal{D})} = 0$$

Another consequence of this is that we can write

$$\frac{\partial \vartheta^*}{\partial \mathbf{d}}\Big|_{\mathbf{d}'} = \left(- \left[\frac{\partial^2 \text{ELBO}_{\mathbf{d},\mathbf{y}_i}(q_\vartheta)}{\partial \vartheta \partial \vartheta^T} \right]^{-1} \times \frac{\partial^2 \text{ELBO}_{\mathbf{d},\mathbf{y}_i}(q_\vartheta)}{\partial \vartheta \partial \mathbf{d}^T} \right) \Big|_{\mathbf{d}', \vartheta^*(\mathbf{d}', \mathbf{y}'_i)}$$

Where $\mathbf{y}'_i = \theta^T \mathbf{d}' + \mathbf{z}$. Now we have the indirect gradient.

4.5 Design

4.6 Results

4.7 Evalution

References

[1] D. V. Lindley, "On a Measure of the Information Provided by an Experiment," *The Annals of Mathematical Statistics*, vol. 27, no. 4, pp. 986–1005, 1956. DOI: 10.1214/aoms/1177728069. [Online]. Available: https://doi.org/10.1214/aoms/1177728069.

[2] E. G. Ryan, C. C. Drovandi, J. M. McGree, and A. N. Pettitt, "A review of modern computational algorithms for bayesian optimal design," *International Statistical Review*, vol. 84, no. 1, pp. 128–154, 2016. DOI: https://doi.org/10.1111/insr.12107. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/insr.12107. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/insr.12107.

[3] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, Apr. 2017. DOI: 10.1080/01621459.2017.1285773. [Online]. Available: https://doi.org/10.1080%2F01621459.2017.1285773.

[4] J. Lorraine, P. Vicol, and D. Duvenaud, *Optimizing millions of hyperparameters by implicit differentiation*, 2019. arXiv: 1911.02590 [cs.LG].

[5] O. Krause, *Pml lecture notes for lectures by oswin*, 2022.