Deep Learning FSS22
Assignment 3: Hyperparameter Optimization

Timur Michael Carstensen - 1722194

05.06.2022

Contents

1 Basic Hyperparameter Tuning **1**
1.1 a) . . . 1
1.2 b) . . . 1
1.3 c) . . . 2
1.4 d) . . . 3
1.5 e) . . . 4

2 Gaussian Process Regression **5**
2.1 a) . . . 5
2.2 b) . . . 6
2.3 c) . . . 7
2.4 d) . . . 9
2.5 e) . . . 10

3 Bayesian Optimization **10**

1 Basic Hyperparameter Tuning

1.1 a)

The results for the manual hyperparameter search can be seen in Figure 1. We first tried to get an overview of the hyperparameter space by exploring evenly in the first nine trials. That is, we tried all combinations of $x_1 \in \{0, 0.5, 1\}$ and $x_2 \in \{0, 0.5, 1\}$. Then, we observed that lower function values were seen in the top-right quadrant of the hyperparameter space and focussed our search on that area. After observing that the function values increased around $[0.7, 1]$, we evaluated the function more in the middle of the top-right quadrant and achieved a function value of -3.0 with $[0.7, 0.7]$. Finally, we tried to evaluate f_1 at $[0.8, 0.8]$ and achieved a function value of -7.83. Given that the function value decreased so quickly for such a small difference in function inputs, we can conjecture that the validation loss surface is not well-behaved and that there are many local optima to this function. Therefore, we believe that we found a good result.

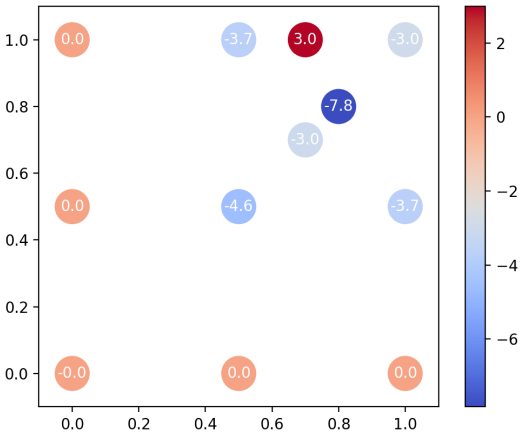


Figure 1: Results obtained for the manual hyperparameter search.

1.2 b)

The plot of f_1 shown in Figure 2 confirms our belief from subsection 1.1. We found a function value of -7.83 with $[0.8, 0.8]$ while the true global minimum of the function is -7.88 with $[0.7917, 0.7917]$. That is, we got extremly close to the global minimum within the 12 trials that we performed. The plot also shows that the surface of f_1 is not well-behaved as it has many (4) local minima. That is the surface has many peaks and valleys and is not convex.

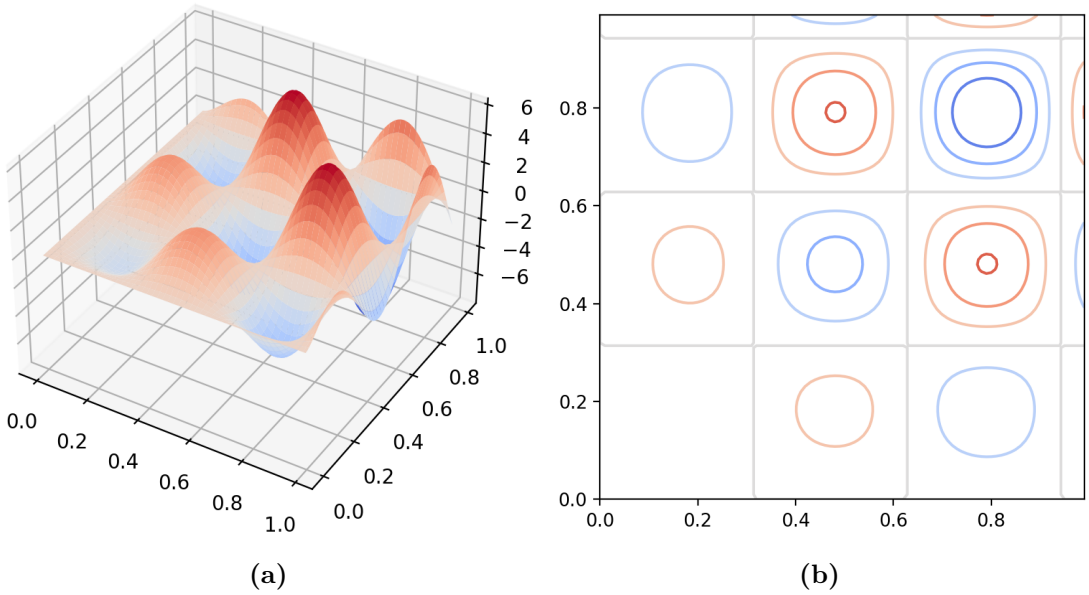


Figure 2: Function surface of f_1; (a) 3D plot, (b) contour plot

1.3 c)

The results of the grid search can be seen in Figure 3. We split the two hyperparameters (x_1, x_2) into 3 and 4 equidistant points ($x_1 \in \{0, 0.5, 1\}$ and $x_2 \in \{0, \frac{1}{3}, \frac{2}{3}, 1\}$) and evaluated f_1 for all possible combinations of the two paramters. The grid search method did not find the global minimum, which is to be expected given the knowledge about it's location. This is coherent with what we learned in the lecture. That is, grid search only explores $n^{1/L}$ values per hyperparameter for n trials and L hyperparameters. In our case: $L = 2, n = 12$ and $12^{1/2} = 3,46$ which is very close to the average of the parameters that we did explore: $\frac{4+3}{2} = 3.5$.

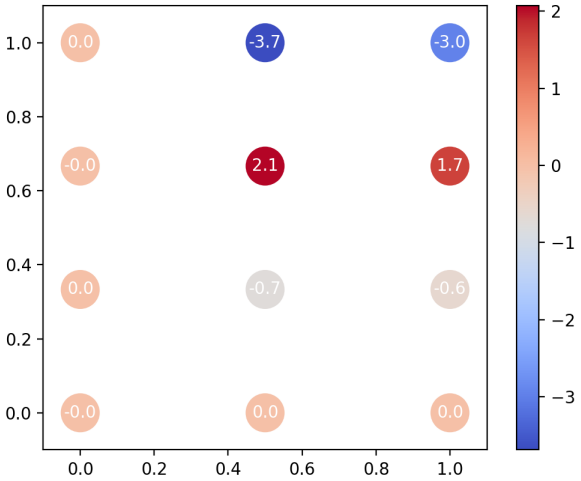
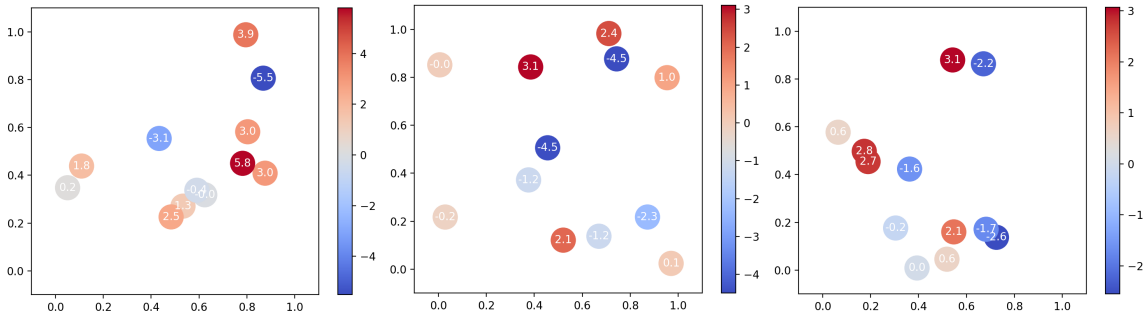


Figure 3: Results obtained for the manual hyperparameter search.

1.4 d)

The results of multiple runs of random search are shown in Figure 4. We can see that a lot of the hyperparameter configurations are "clustered" together. That is, there are many redundant trials. For example, in Figure 4 (a) we can observe that the hyperparameter space is not explored in the top-left quadrant (not at all, actually). Conversely, the range of hyperparameter configurations between $x_1 \in [0.5, 0.65]$ and $x_2 \in [0.2, 0.4]$ are explored with four trials. A similar clustering of evaluated hyperparameter configurations can be see in Figure 4 (c). Out of the three runs that we performed, only the one in Figure 4 (b) achieves a somewhat good exploration of the hyperparameter space, given the prespecified budget.

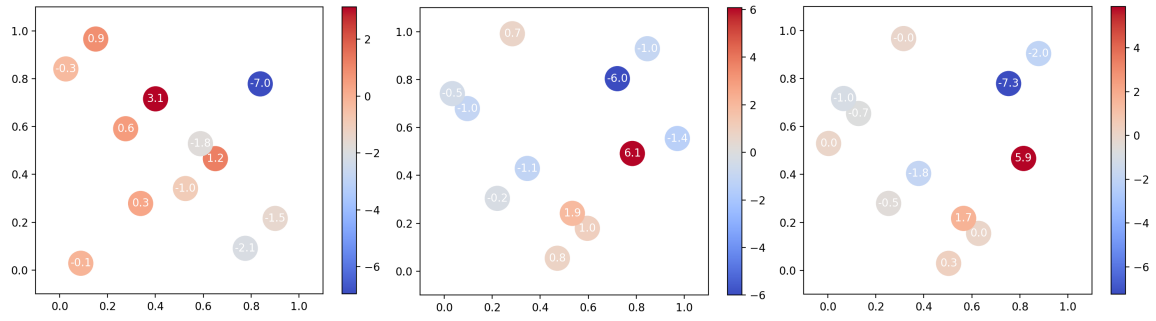


(a) **(b)** **(c)**

Figure 4: Results of three random search trials using random uniform sampling for the hyperparameters for f_1 shown in (a), (b), and (c).

We also used Sobol sequences to generate the values for the hyperparameter configurations to be explored in the trials. Thre results are shown in Figure 5. The first observation that

we can directly make, is that we achieve better results in that we get much closer to the global optimum of f_1. The second observation that we can make is that the hyperparameter space is much better (i.e. more evenly) explored since Sobol sequences generate more evenly spaced random values than directly sampling from a uniform distribution [BB12].



(a) **(b)** **(c)**

Figure 5: Results of three random search trials using Sobol sequences for the hyperparameters for f_1 shown in (a), (b), and (c).

1.5 e)

We can treat the good fraction of the HP configuration space as a probability. That is, in each trial we have the probability p of picking a good configuration/being the good area of the HP configuration space. Given that we perform n trials, the probability x of finding at least one good HP configuration is the counterprobability of finding no good HP configuration:

$$x \leq 1 - (1-p)^n \tag{1}$$

We can solve this by replacing the inequality with an equality:

$$x = 1 - (1-p)^n \tag{2}$$

$$(1-p)^n = 1 - x \tag{3}$$

$$n \log(1-p) = \log(1-x) \tag{4}$$

$$n = \frac{\log(1-x)}{\log(1-p)} \tag{5}$$

That is, the number of trials depends on both the probability x which we are trying to achieve and p, the good fraction of the HP space. When we have L hyperparameters, the

probability of finding a good HP configuration is the probability of all hyperparameters being in the good fraction of their configuration space at the same time p^L:

$$x = 1 - (1 - p^L)^n \tag{6}$$

$$n = \frac{\log(1-x)}{\log(1-p^L)} \tag{7}$$

That is, the number of trials necessary to find at least good HP configuration grows quasi exponentially in the number of hyperparameters L. This is akin to the curse of dimensionality. That is, our problem gets exponentially harder to solve as we increase the dimensionality.

When using grid search, we only explore $n^{1/L}$ configurations per parameter (vs. n in Random Search). That is, we explore fewer configurations for the same amount of trials and will therefore need a higher number of trials to find at least one good HP configuration with a high probability x:

$$n^{1/L} = \frac{\log(1-x)}{\log(1-p^L)} \tag{8}$$

$$n = \left(\frac{\log(1-x)}{\log(1-p^L)}\right)^L \tag{9}$$

So, in grid search, the number of trials needed grows exponentially in the number of hyperparameters. However, we assumed that the good fraction of the HP space is interpretable as a probability. This is intuitive when using uniform random sampling. In grid search however, we do not perform random sampling but split our grid into equidistant points or use some other kind heuristic to predefine a set of values per hyperparameter to explore. Thus, the interpretation of p as a probability does not work out necessarily.

2 Gaussian Process Regression

2.1 a)

Figure 6 shows the provided scaled RBF kernel for multiple values of the *scale* and *length-scale* parameters. Intuitively, the *length-scale* parameter controls what values are seen as similar and what values are not. For example, when inspecting the kernel with $scale = 1$ and $length - scale = 3$ (i.e. the solid green line), we can see that kernel assigns much higher values for a distance of 2.5 between inputs (roughly 0.7), whereas another kernel with $length - scale = 1$ (i.e. the solid orange line) assigns a value of almost 0 for samples that have the same distance between them.

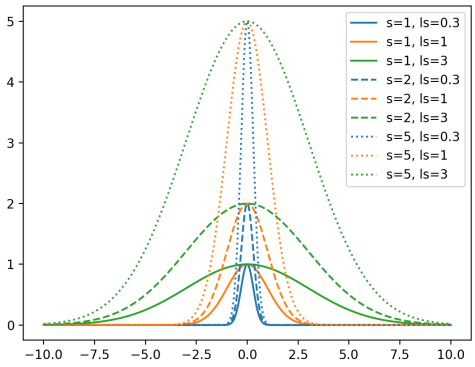


Figure 6: Scaled RBF kernel for different values of the *scale* and *length-scale* paramters. The x-axis displays the distance between two samples and the y-axis the assigned similarity values by the kernel.

The *scale* parameter scales the similarity scores assigned to two samples by the kernel. In Figure 6, we can see that when we keep the *length-scale* parameter fixed and vary the *scale*, the similarity values for the same distances between samples are either scaled up or down.

The used kernel function is a scaled RBF kernel. The RBF kernel is multiplied with the constant kernel, whose value is controlled by the *scale* parameter:

$$\kappa_{scaled-RBF}(\mathbf{x_1}, \mathbf{x_2}) = const * \exp\left(\frac{\|\mathbf{x_1} - \mathbf{x_1}\|^2}{2\sigma^2}\right)$$
$$\text{where } const = scale \text{ and } \sigma = length - scale$$

2.2 b)

As discussed in the previous subsection (cf. subsection 2.1), the *scale* parameter scales the outputs of the kernel whereas the *length-scale* parameter controls which values are seen as similar by the kernel. Now, when we sample from a GP prior we can see the effect of these parameters in Figure 7. We vary the *scale* by column and the *length-scale* by row. We can observe that smaller length scale values correspond to "noisier" functions (i.e. the function values vary more). Higher *scale* values just scale the y-axis up or down. So, we can say that we sample smoother functions for larger *length-scales* and vice-versa for smaller values.

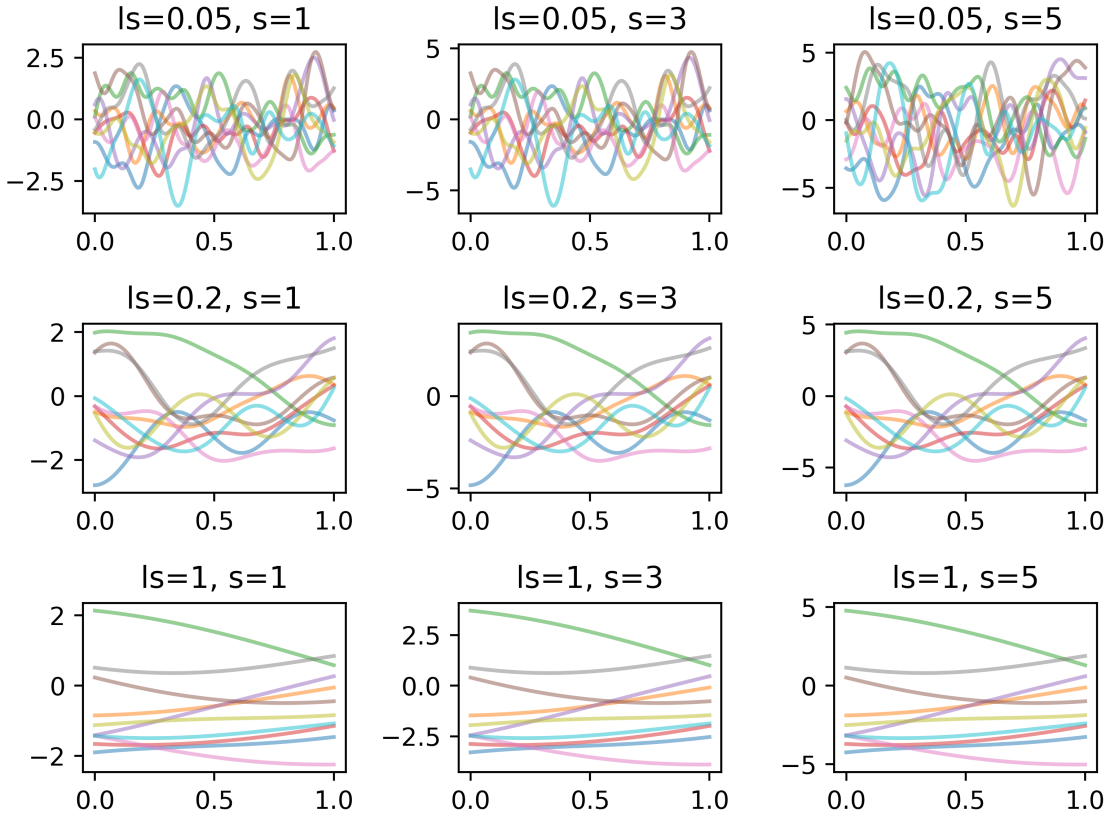


Figure 7: Samples from a Gaussian Process priors using the provided sclaed RBF kernel with different *length-scale* and *scale* parameters.

2.3 c)

Figure 8 shows the plot for function f_2. We can see that the minimum of the function is to the right of our observation at $x = 0.5$. With three observations, we evaluate the function at $x \in \{0, 0.5, 1\}$. However, we can see that the (global) minimum is to the right of our observation at $x = 0.5$ and that the function values have a lot higher variance in the right side of the input domain. Also, with only three observations, we will not be able to approximate f_2 very well with Gaussian Process Regression. Hence, we will need to use a prior that encodes a high uncertainty such that we are also likely to find our global minimum. Therefore, we used a very low *length-scale* of 0.03 and a *scale* of 0.36.

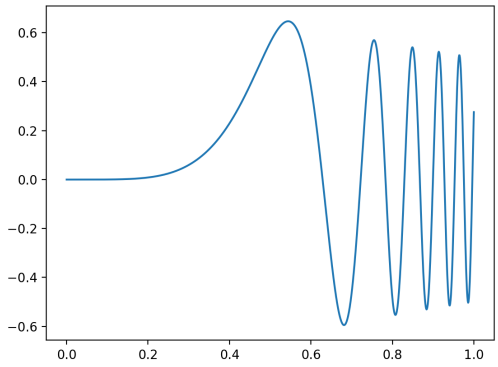


Figure 8: Plot showing f_2

As expected, we cannot approximate the function well with only three observations. Our posterior mean does not capture any of the variance in f_2 in the right side of the input domain and misses our global minimum completly. The posterior variance is low for values (very) close to our observations, but quickly increases for values that are further away as can be seen by the orange bounds in Figure 9.

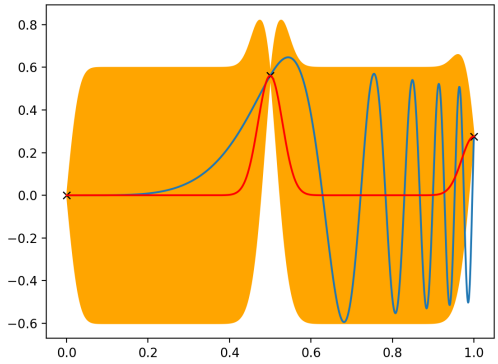


Figure 9: Posterior of the Gaussian Process Regression for f_2 with $s = 0.36$ and $ls = 0.03$.

We can also see that our global optimum lies within the region of uncertainty (posterior variance). Therefore, we are likely to evaluate f_2 at input values close to the global optimum during bayesian optimization. We can therefore conjecture that we are likely to converge to the global optimum and that are our kernel parameters were chosen properly given that we only have three observations. However, if we had more oberservations we could use a prior that encodes a lower uncertainty and likely need less trials to find the global optimum.

2.4 d)

In Figure 10 we varied the number of samples ($n \in \{6, 12, 18\}$) by column and the *length-scale* ($ls \in \{0.03, 0.045, 0.0675\}$) while keeping the *scale* fixed at 0.36.

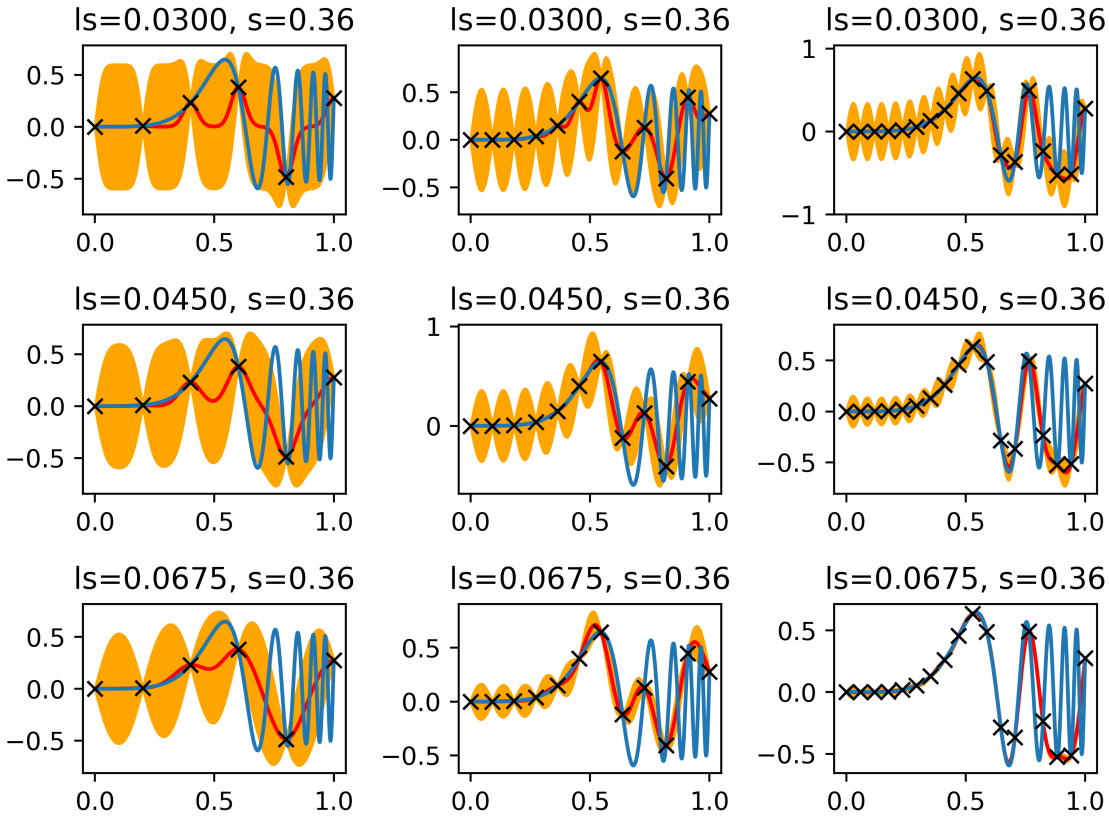


Figure 10: Varying the number of samples (by column: 6, 12, 18) and the magnitude of the *length-scale* parameter (multiplied by 1.5 in each row, starting with 0.03).

We can observe that our approximation of the underlying function f_2 improves as we increase the number of samples, which is to be expected. We can also observe that increasing the *length-scale* does not necessarily improve our fit to the function. In fact, increasing the length-scale does nothing to capture the variability of function values in the right side of the input domain, as discussed previously. With $n = 18$ observations and $ls = 0.0675$, we fit the function quite well and would find the global optimum with bayesian optimization. We can also see that we underfit f_2 everywhere, though this does not prevent us from finding the global minimum. That is, for all cases with $n = 18$, we would likely find the global optimum as visible in the figure above. Also, it is entirely possible that our prior is not suitable for the functional form of f_2 and that we may want to consider a combination of multiple kernels (i.e. perhaps two: one for the right side of the domain, and one for the

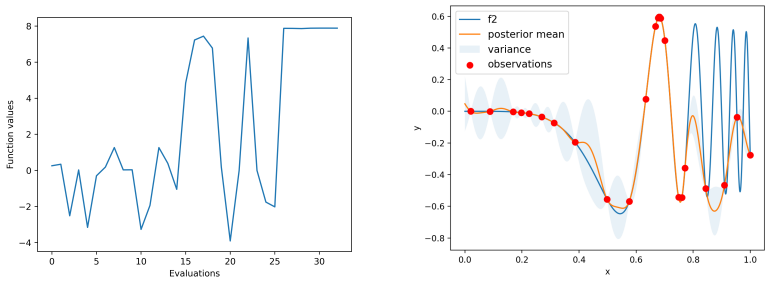
left).

2.5 e)

If we were to use bayesian optimization with our prior from Figure 9 and three observations, it would not be guaranteed that we find the global minimum. In fact, it would be very likely that the model would converge in one of the (four) local minimima (cf. Figure 8). This is because our prior is quite uninformative for large regions of the input domain (i.e. everything that is not in the direct vicinity of the given observations). That is, we would likely start sampling randomly in areas where the uncertainty is at it's maximum (most of the input domain).

3 Bayesian Optimization

Figure 11 shows the results of bayesian optimization for f_1 and f_2. To run bayesian optimization, the package BayesianOptimization [Nog] was used with standard settings (prior observations = 3; number of iterations = 30). We, again, used three prior observations as our history and then ran 30 trials of bayesian optimization for both functions. Both optimizers converged to the global minimum (or got very close): $\hat{y} = -7.886$ and $[0.792, 0.7916]$ for f_1, and $\hat{y} = -0.5947$ and $[0.6812]$ for f_2. Note that we had to rewrite the functions: since the package is only meant to find the maximum of a given function we wrote wrapper functions that return the negative function values. Hence, the plot in Figure 11 (b) shows an inverted version of f_2. We can see the difference between exploration and exploitation in the Figure 11 (a) where in the beginning many different HP configurations are tested our and then after $\sim$ 25 evaluations, our model switches to exploitation.



(a) **(b)**

Figure 11: Results of runnning bayesian optimization for f_1 and f_2: (a) shows the evaluated function values for f_1; (b) shows f_2, the posterior mean and variance and the observations of function values that were evaluated during bayesian optimization.

References

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[Nog] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.