Exercise 6

Deadline: 09.07.2019

In this exercise, we revisit an exercise from "Fundamentals of Machine Learning", where we reconstructed an image from a few given pixels using Kernel Ridge Regression (for convenience we repeat the text of this exercise below), and optimize its hyperparameters using Gaussian Processes.

Regulations

Please create a Jupyter notebook `gp.ipynb` for your solution and export it into `gp.html`. Zip both files along with the extended `ridge_regression.py` and the solution to task 1 into a single archive with naming convention (sorted alphabetically by last names)

`lastname1-firstname1_lastname2-firstname2_exercise06.zip`

or (if you work in a team of three)

`lastname1-firstname1_lastname2-firstname2_lastname3-firstname3_exercise06.zip`

and upload it to Moodle before the given deadline. We will give zero points if your zip-file does not conform to the naming convention.

1 Comment on your solution to exercise 5

Study the sample solution `ex05_solution.html` on Moodle and use it to comment on your own solution to this exercise. Specifically, copy your notebook `cvae.ipynb` to `cvae-commented.ipynb` and export it to `cvae-commented.html` in the end. Insert comments as markdown cells starting with

```
<span style="color:green;font-weight:bold">Comment</span>
```

in order to clearly distinguish your comments from other cell types. The point of these comments is that you identify your errors and bugs yourselves, so that you learn from your mistakes. In addition, the tutor will have an easier time distinguishing between the initial mistake and consequential errors caused by the first one and will only deduct points for the former. If you fail to hand in comments, the tutor is not required to make this distinction and will deduct points for all errors alike.

Gaussian Process Regression (= Kernel Ridge Regression)

The task is to reconstruct an image from about 10% of its pixels. The training data are in image `cc_90.png` (to be found on Moodle), where the missing pixels have the grayvalue 0. From the lecture you know that Gaussian process regression can be interpreted as kernelized ridge regression. This Gaussian Process will be called GP1 in the sequel. Recall that we can compute the regressed values y_{new} by:

$$\mathbf{y}_{new} = \kappa^T \left(K + \tau \cdot \mathbb{I}\right)^{-1} \mathbf{y} \; . \tag{1}$$

with kernel matrix K and kernel vector κ. We can precompute the interpolation coefficients $\tilde{\mathbf{y}} = \left(K + \tau \cdot \mathbb{I}\right)^{-1} \mathbf{y}$. The kernel vector $\kappa = \kappa(X_{new}, X_i)$ must be computed from every new instance X_{new} (i.e. for every target pixel). Note that you must also predict new values for the training pixels, because a regularized regression will not interpolate their original values.

We will change this exercise as follows

1. Instead of a Gaussian kernel we use a generalized exponential kernel.

2. The input data are now noisy so that the regularization parameter τ needs to be chosen carefully.
3. We add a second Gaussian Process (GP2) on top in order to learn GP1's hyperparameters.
4. We provide the original image as test data for the hyperparameter optimization.
5. We provide a reference solution `ridge_regression.py` of the old exercise as a starting point.

2 GP Optimization of a Toy Problem (16 points)

We first solve a toy problem to learn how GP optimization works and to debug the code in a simpler context. To this end, use conda or pip to install the Python packages GPy, Optunity and sobol.

The function we are going to optimize is

$$f(x, y) := x^2 - x + y^2 + y - \cos(2\pi x - \pi)) - \cos(2\pi y + \pi)) + 2.5$$

It has many local minima and exactly one global minimum at $(\frac{1}{2}, -\frac{1}{2})$ with function value 0. For the exercise, we assume that the direct optimization of this function is infeasible and use Gaussian process optimization instead.

1. Implement $f(x, y)$. Plot it in the rectangle $[-6, 6]^2$ using matplotlib.
2. A *utility function* estimates how useful a point $q = (x, y)$ might be for finding the desired optimum. A popular choice in the field of Bayesian optimization is *Expected Improvement*:

   $$u(q, E_{\text{best}}) = \sqrt{\text{Var}(q)}\Big(\gamma(q)\Phi(\gamma(q)) + \phi(\gamma(q))\Big)$$

   with $\gamma(q) = \frac{E_{\text{best}} - E(q)}{\sqrt{\text{Var}(q)}}$. Here, $E(q)$ and $\text{Var}(q)$ are estimates of the mean and variance of the target function, as returned by the Gaussian process model at point q, E_{best} is our current best guess of the target function's minimal value, and Φ and ϕ are the cdf and pdf of the normal distribution. (Hint: You can get both of them via `scipy.stats.norm.pdf` and `scipy.stats.norm.cdf`).
3. Create an initial set of 30 training points $Q \in R^{2\times 30}$ by drawing semi-random locations from a sobol sequence (see `https://en.wikipedia.org/wiki/Sobol_sequence`):

   ```
   import sobol
   number_of_samples = 30
   parameterUpperLimits = np.array([6,6])
   parameterLowerLimits = np.array([-6,-6])
   for i in range(number_of_samples):
       x,y = sobol.i4_sobol(2,i)[0] * (parameterUpperLimits -
           parameterLowerLimits) + parameterLowerLimits
   ```

4. Create the initial training set by computing the true function values $E = \{f(x, y) : (x, y) \in Q\}$ for the points in Q.
5. Use the GPy module to train a GP model `GPy.models.GPRegression` for the current training set, using the kernel `GPy.kern.RBF(2) + GPy.kern.White(2)`.
6. Use the Optunity module to find the point $q = (x, y) \in [-6, 6]^2$ which maximizes the expected improvement $u(q, E_{\text{best}})$.
7. Evaluate $f(q = (x, y))$ at the new candidate point and add $(q, f(q))$ to the training set Q.
8. Go to step 5 and repeat the optimization 30 times. Print the true function values and the function values predicted by the GP as a function of iteration number.
9. Mark the global optimum and the points in your final training set Q in the plot from step 1. What do you observe?

10. Repeat the experiment, but alternate between two utility functions in consecutive iterations: maximal expected improvement and maximal variance of the GPy model. This encourages the search to explore a larger part of the function domain. Create the same plots as in the initial experiment and comment on the differences you observe. As a baseline, also compute 60 values of $f(x, y)$ for uniformly random points in $[-6, 6]^2$ and compare their minimum with the best value found by Bayesian optimization.
11. Plot the GP's approximation for $f(x, y)$ before and after optimization (i.e. trained from the initial 30 points, and trained with 60 optimally chosen points).
12. Which requirements should the function $f(x, y)$fulfill in order for this method to work?

3 Reconstruction of a Corrupted Image (8 points)

Now, we return to the original task. A reference solution for Gaussian process interpolation of an image can be found in file `ridge_regression.py` on Moodle. We will use the generalized exponential kernel:

$$G(\mathbf{x}, \mathbf{x}'; \gamma, \rho) = \exp\left(-r(x, z, \rho)^\gamma\right) \tag{2}$$

$$r(\mathbf{x}, \mathbf{x}'; \rho) = \sqrt{\frac{|\mathbf{x} - \mathbf{x}'|^2}{\rho^2}} \tag{3}$$

The implementation in `ridge_regression.py` must be slighlty modified to accomodate this kernel (include the modified file in your Moodle upload). Fix hyperparameters $\theta = (\tau, \rho, \gamma) = (0.8, 3.0, 1.0)$ for now. We will use GP optimization to learn these parameters in the next task. Plot the reconstructed image alongside the corrupted one and the ground truth image `charlie-chaplin.jpg` (also provided on Moodle).

Since Gaussian process reconstruction will be called multiple times during optimization, the solution in `ridge_regression.py` uses a faster implementation than the naive one. Inspect the file `ridge_regression.py` to answer the following questions:

- The code does not contain explicit matrix inversion. How is the inverse of K applied instead?
- Which data structure was used to speed up the prediction and why is this a good idea?

4 Bayesian Optimization of Hyperparameters (16 points)

We now use GP optimization according to task 2 in order to find optimal hyperparameters $\theta = (\tau, \rho, \gamma)$. The quality of the reconstruction is measured by the correlation coefficient between the reconstructed image A and ground-truth image B:

$$COR(A, B) = \frac{\sum_{i=0}^{w-1} \sum_{j=0}^{h-1} (A_{i,j} - \bar{A})(B_{i,j} - \bar{B})}{\sqrt{\sum_{i=0}^{w-1} \sum_{j=0}^{h-1} (A_{i,j} - \bar{A})^2} \sqrt{\sum_{i=0}^{w-1} \sum_{j=0}^{h-1} (B_{i,j} - \bar{B})^2}} \tag{4}$$

where $\bar{A}$ and $\bar{B}$ are the mean grey values of A and B. Note that COR is implicitly a function of θ, because the reconstructed image A depends on these hyperparameters.

A 3-dimensional grid search to find optimal hyperparameters is expensive since every parameter set requires a full sweep of training and prediction with GP1. To reduce the number of calls to GP1, we approximate $COR(\theta)$ by a second Gaussian process model GP2 (i.e. $COR(\theta)$ replaces $f(x, y)$ from task 2). Evaluating this approximation is much cheaper, so we can probe it much more densely. Moreover, we can utilize the uncertainty provided by GP2 in the utility function.

1. Implement the COR calculation.

2. Adapt your solution from task 2 to optimize the hyperparameters $\theta = (\tau, \rho, \gamma)$ so that the correlation coefficient is maximized (hint: you can realize this by minimizing $(1 - COR)$ instead). The parameter search regions shall be $\tau \in [0.005, 1]$, $\rho \in [1, 7]$, and $\gamma \in [1, 4]$. Implement GP2 with the kernel `GPy.kern.Matern52` (as recommended in Snoek's original paper) with

$$r = \sqrt{(\tau - \tau')^2 + \left(\frac{\rho - \rho'}{4}\right)^2 + (\gamma - \gamma')^2}$$

   Don't forget to change the sobol sequence generator from 2D to 3D.

3. Plot the true and approximated values of COR as a function of iteration number.

4. Plot the image obtained with your final parameters θ alongside the reconstruction from task 3 and the ground-truth image and comment on the result.