Lab session 3: Dimensionality Reduction using Gaussian Processes

GP Summer School – Sheffield, 10-13th of June 2013

The aim of this lab session is to illustrate the concepts seen during the lectures. We will focus on three aspects of GPs: sampling, the design of Experiments and uncertainty propagation. If you are new to Python, you may be interested in the cheatsheet at the end of this document.

Since the background of the attendees is very diverse, this session will attempt to cover a large spectrum from the very basic of GPs to more technical questions. This in mind, the difficulties of the questions of the lab session has been ranked from $\star$ (easy) to $\star\star\star$ (difficult). Feel free to skip the parts that are either too easy or too technical.

In this section you are asked to work with a dataset called `digits.npy`. You can download the dataset and some convenience scripts from `http://staffwww.dcs.sheffield.ac.uk/people/J.Hensman/gpsummer/Lab3.zip`. We will call this folder in the following your `Lab3/` folder.

Setting up ipython for this session

This session works with datasets provided in your `Lab3/` folder. Thus, make sure to traverse into this directory, to be able to use the code samples in this script. Otherwise, you will have to adjust the code snippets accordingly. In order to make all the code snippets work you need to traverse into your `Lab3/` folder and run:

```
import numpy as np, matplotlib.pyplot as pb
import string
from load_plotting import *
pb.ion()
```

Tip: Optimizations at the end of this session will take a long time, thus you can skip the questions and answer them while your optimizations are running.

1 Introduction: Principal Component Analysis

Principal component analysis (PCA) finds a rotation of the observed outputs, such that the rotated principal component (PC) space maximizes the variance of the data observed, sorted from most to least important (most to least variable in the corresponding PC).

To demonstrate PCA we created a dataset `Lab3/digits.npy` containing all digits from 0 – 9 handwritten, provided by de Campos et al. [2009]. All digits were cropped and scaled down to an appropriate format. We will only use some of the digits for the optimizations and plots in this script, but feel free to exchange the filtering as you wish.

```
digits = np.load('digits.npy')
which = [0,3,6,7,8,9] # which digits to work on
digits = digits[which,:,:,:]
num_classes, num_samples, height, width = digits.shape
labels = np.array([[str(l)]*num_samples for l in which])
```

You can try to plot some sample using `pb.matshow`. In order to apply PCA in an easy way, we have included a PCA module in your `Lab3/pca.py`. You can import the module by `import <path.to.pca>` (without the ending `.py`!). To run PCA on the digits we have to reshape (Hint: `np.reshape`) `digits`. ⋆ What is the right shape $N \times D$ to use? We will call the reshaped observed outputs `Y` in the following.

So now let's run PCA on the reshaped dataset `Y`:

```
p = pca.PCA(Y) # create PCA class with digits dataset
p.plot_fracs(20) # plot first 20 eigenvalue fractions
p.plot_2d(Y,labels=labels.flatten(), colors=colors)
pb.legend()
```

The resulting plot will show the lower dimensional representation of the digits in 2 dimensions and it should look alike the result shown in 1(a).

2 Gaussian Process Latent Variable Model

The Gaussian Process Latent Variable Model (GPLVM) introduced by Lawrence [2004] is the embedding of PCA into a GP framework, where the latent inputs $\mathbf{X}$ are learnt as hyperparameters and the mapping variables $\mathbf{V}$ are integrated out. This allows us to apply non linear PCA using a non-linear kernel. But first, let's see how GPLVM is equivalent to PCA using an automatic relevance determination (ARD, see e.g. Bishop et al. [2006]) linear kernel (Figure 1(b)):

```
import GPy # import GPy package
Yn = Y-Y.mean()
input_dim = 4 # How many latent dimensions to use
kernel = GPy.kern.linear(input_dim, ARD=True) # ARD kernel
kernel += GPy.kern.white(input_dim) + GPy.kern.bias(input_dim)
m = GPy.models.GPLVM(Yn, input_dim=input_dim, kernel=kernel)
m.ensure_default_constraints()
m['noise'] = m.likelihood.Y.var()/20. # start noise is 5% of datanoise
m.optimize(messages=1, max_f_eval=1000) # optimize for 1000 iterations
m.kern.plot_ARD()
plot_model(m, m['linear_variance'].argsort()[-2:], labels.flatten())
pb.legend()
```

As you can see the solution with a linear kernel is the same as the PCA solution with the exception of rotational changes. The solution you see was only running for 1000 iterations, thus it might not be converged fully yet.

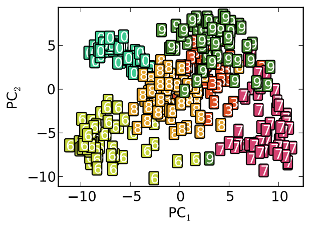
The next step is to use a non-linear mapping between inputs $\mathbf{X}$ and ouputs $\mathbf{Y}$, as e.g. the `GPy.kern.rbf` kernel . You can try using different kernels and see the different characteristics the kernels pick up. You can also try combinations of kernels if there is enough time. In case you run into stability problems try initializing the kernel parameters differently.

Question 1

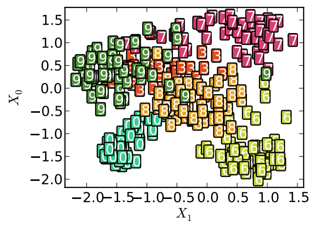
- $\star$ How do your linear solutions differ between PCA and GPLVM with a linear kernel?
- $\star$ Which characteristics can GPLVM with an RBF kernel differentiate, which PCA is not able to?
- $\star\star$ How do linear ARD parameters compare to PC variances?
- $\star\star$ How do they differ in non-linear mappings?
- $\star\star\star$ How does ARD work? Bishop et al. [2006]

3 Bayesian GPLVM

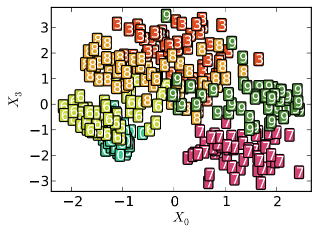
In GPLVM we use a point estimate of the distribution of the input $\mathbf{X}$, through the maximum *a posteriori* (MAP) approach. In Bayesian GPLVM we approximate the true distribution $p(\mathbf{X}|\mathbf{Y})$ by a variational approximation $q(\mathbf{X})$ and integrate $\mathbf{X}$ out. This allows the algorithm to fully switch off latent dimensions, as noise dimensions can be explained inside the distribution $q(\mathbf{X})$ and do not have to be accounted for in the ARD parameters. If you have got enough time try running a `GPy.models.BayesianGPLVM` with a `GPy.kern.rbf` kernel (figure 2):



(a)

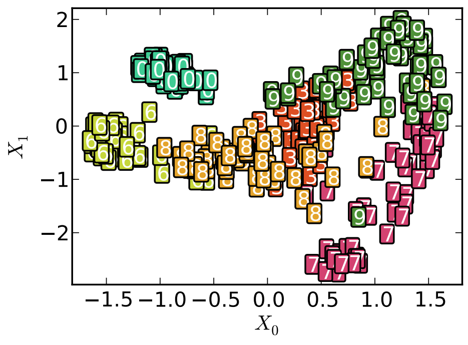


(b)

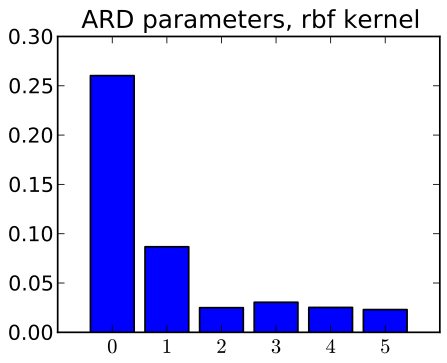


(c)

Figure 1: Two dimensional representations of the `digits` dataset. (a) PCA solution to the `digits.npy` dataset. (b) GPLVM solution to the `digits.npy` dataset using linear kernel. (c) GPLVM solution to the `digits.npy` dataset using rbf kernel



(a)



(b)

Figure 2: Two dimensional representations of the `digits` dataset. (a) Bayesian GPLVM solution to the digits dataset. (b) ARD parameters learnt by Bayesian GPLVM.

```
import GPy # import GPy package
Yn = Y-Y.mean(0)
input_dim = 6 # How many latent dimensions to use
kernel = GPy.kern.rbf(input_dim,ARD=True) # ARD kernel
kernel += GPy.kern.white(input_dim) + GPy.kern.bias(input_dim)
m = GPy.models.BayesianGPLVM(Yn, input_dim=input_dim, kernel=kernel, num_inducing=15)
# start noise is 1% of datanoise
m.ensure_default_constraints()
m['noise'] = m.likelihood.Y.var()/100.
m.optimize('bfgs',messages=1, max_f_eval=10000, max_iters=10000)
plot_model(m, m['rbf_len'].argsort()[:2], labels.flatten())
pb.legend()
m.kern.plot_ARD()
```

Depending on computing power this optimization can take a long time. You are free to interrupt the optimization at any point in time by hitting `Ctrl+c`. This will leave you with the model `m` in the current state and you can plot and look into the models parameters. If you run GPy locally on a laptop, you can just skip to the next paragraph which loads a pre-optimized model.

Interactive demo

From your `Lab3/` folder run the module:

```
python load_bgplvm_dimension_select.py
```

(replace "python" with "run" in the ipython environment). This module loads a pre-optimized Bayesian GPLVM model (like the one you just trained) and allows you to interact with the latent space. Three interactive figures pop up: the latent space, the ARD scales and a sample in the output space (corresponding to the current selected latent point of the other figure). You can sample with the mouse from the latent space and obtain samples in the output space. You can select different latent dimensions to vary by clicking on the corresponding scales with the left and right mouse buttons. This will also cause the latent space to be projected on the selected latent dimensions in the other figure.

Observations

Confirm the following observations by interacting with the demo:

- We tend to obtain more "strange" outputs when sampling from latent space areas away from the training inputs.
- When sampling from the two dominant latent dimensions (the ones corresponding to large scales) we differentiate between all digits. Also note that projecting the latent space into the two dominant dimensions better separates the classes.
- When sampling from less dominant latent dimensions the outputs vary in a more subtle way.

You can also run the dimensionality reduction example

```
GPy.examples.dimensionality_reduction.bgplvm_simulation()
```

which requires less computing power. The simulation is a sinusoid and a double frequency sinusoid function as input signals, which are retrieved completely by the Bayesian GPLVM. All dimensions of the initial latent space that are irrelevant get switched off.

Question 2

⋆ Can you see a difference in the ARD parameters to the non Bayesian GPLVM?

⋆ ⋆ ⋆ How does the Bayesian GPLVM allow the ARD parameters of the RBF kernel magnify the two first dimensions?

- ⋆ Is Bayesian GPLVM better in differentiating between different kinds of digits?
- ⋆⋆ Why does the starting noise variance have to be lower then the variance of the observed values?
- ⋆⋆ How come we use the lowest variance when using a linear kernel, but the highest lengtscale when using an RBF kernel?

4 Manifold Relevance Determination

In Manifold Relevance Determination we try to find one latent space, common for K observed output sets (modalities) $\{\mathbf{Y}_k\}_{k=1}^{K}$. Each modality is associated with a separate set of ARD parameters so that it switches off different parts of the whole latent space and, therefore, X is softly segmented into parts that are private to some, or shared for all modalities. Can you explain what happens in the following example?

```
m = GPy.examples.dimensionality_reduction.mrd_simulation(optimize = False)
m.optimize(messages = True, max_f_eval=5000, optimizer = 'SCG')
```

Again, you can stop the optimizer at any point and explore the result obtained with the so far training:

```
m.plot_scales()
m.plot_X_1d()
```

Which signal is shared across the three datasets? Which are private? Are there signals shared only between two of the three datasets?

References

C. M. Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

T. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. 2009.

N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in neural information processing systems*, 16(329-336):3, 2004.