Chapter 1

The Multi-Output GP Duality

1.1 Vector-Valued Function View

We first consider the vector-valued function perspective on multi-output GPs. From this perspective, the kernel of such as process as a matrix-valued binary function. For concreteness, consider a simple multi-output GP of the Semi-Parametric Latent Factor (SLFM) (Seeger, Teh, and Jordan 2005) form:

$$\mathbf{f}(\mathbf{x}) := \mathbf{A}\mathbf{g}(\mathbf{x}) + \mathbf{e}(\mathbf{x})\,, \tag{1.1}$$

where $\mathbf{A} \in \mathbb{R}^{P \times Q}$, $\mathbf{x} \in \mathcal{X}$, and

$$\begin{aligned} \mathbf{g}_q &\sim \mathcal{GP}\big(m_q^{\mathrm{g}}, \kappa_q^{\mathrm{g}}\big)\,, \\ \mathbf{e}_p &\sim \mathcal{GP}\big(m_p^{\mathrm{e}}, \kappa_p^{\mathrm{e}}\big)\,. \end{aligned}$$

Viewing $\mathbf{g}$ as a distribution over vector-valued functions, we can write its mean as a vector-valued function and kernel as a matrix-valued function:

$$\mathbf{m}^{\mathrm{g}}(\mathbf{x}) = \begin{bmatrix} m_1^{\mathrm{g}}(\mathbf{x}) \\ \vdots \\ m_Q^{\mathrm{g}}(\mathbf{x}) \end{bmatrix} \quad \mathbf{K}^{\mathrm{g}}\left(\mathbf{x}, \mathbf{x}'\right) = \begin{bmatrix} \kappa_1^{\mathrm{g}}(\mathbf{x}, \mathbf{x}') & & \\ & \ddots & 0 \\ 0 & & \\ & & \kappa_Q^{\mathrm{g}}(\mathbf{x}, \mathbf{x}') \end{bmatrix}, \tag{1.2}$$

with a similar definition for $\mathbf{e}$. From this is follows that the mean and kernel of $\mathbf{f}$ is

$$\begin{aligned} \mathbf{m}^{\mathrm{f}}(\mathbf{x}) &= \mathbf{A}\mathbf{m}^{\mathrm{g}}(\mathbf{x}) + \mathbf{m}^{\mathrm{e}}(\mathbf{x}), \\ \mathbf{K}^{\mathrm{f}}\left(\mathbf{x}, \mathbf{x}'\right) &= \mathbf{A}\mathbf{K}^{\mathrm{g}}\left(\mathbf{x}, \mathbf{x}'\right)\mathbf{A}^{\top} + \mathbf{K}^{\mathrm{g}}\left(\mathbf{x}, \mathbf{x}'\right). \end{aligned} \tag{1.3}$$

Thus we find that the mean $\mathbf{m}^{\mathrm{f}} : \mathcal{X} \to \mathbb{R}^P$ and covariance $\mathbf{K}^{\mathrm{f}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{P \times P}$ functions are vector- and matrix-valued respectively.

1.2 Real-Valued Function View

Equation 1.1 can be written in *component*-form as follows:

$$\mathbf{f}_p(\mathbf{x}) = \sum_{q=1}^{Q} \mathbf{A}_{pq}\mathbf{g}_q(\mathbf{x}) + \mathbf{e}_p(\mathbf{x}) \tag{1.4}$$

This suggests a way to turn the vector-valued function into a scalar-valued function:

$$f((\mathbf{x}, p)) := \mathbf{f}_p(\mathbf{x}). \tag{1.5}$$

Letting $\mathcal{X}_a := \mathcal{X} \times \{1, ..., P\}$, or equivalently $\mathcal{X}_a := \{(\mathbf{x}, p) : \mathbf{x} \in \mathcal{X}, p \in \{1, ..., P\}\}$, $f : \mathcal{X}_a \to \mathbb{R}$ maps from pairs / 2-tuples to real-numbers, where the first element of the tuple specifies a location in the original input domain and the second element the *output*. For example $f((\mathbf{x}, 2))$ corresponds to the second output at location $\mathbf{x}$.

Performing a similar trick to obtain scalar variants of $\mathbf{e}$ and $\mathbf{g}$, it follows that

$$f((\mathbf{x}, p)) = \sum_{q=1}^{Q} \mathbf{A}_{pq} \, g((\mathbf{x}, q)) + e((\mathbf{x}, p)) \, . \tag{1.6}$$

This process has mean function and kernel

$$\begin{aligned} m^{\mathrm{f}}((\mathbf{x}, p)) &= \sum_{q=1}^{Q} \mathbf{A}_{pq} m^{\mathrm{g}} \left((\mathbf{x}, q) \right) + m^{\mathrm{e}} \left((\mathbf{x}, p) \right) \\ \kappa^{\mathrm{f}}\big((\mathbf{x}, p), (\mathbf{x}', p')\big) &= \sum_{q=1}^{Q} \mathbf{A}_{pq} \, \kappa^{\mathrm{g}} \left((\mathbf{x}, q), (\mathbf{x}', q) \right) \mathbf{A}_{p'q} + \kappa^{\mathrm{e}} \left((\mathbf{x}, p), (\mathbf{x}', p') \right) \end{aligned} \tag{1.7}$$

where $m^{\mathrm{f}} : \mathcal{X}_a \to \mathbb{R}$ and $\kappa^{\mathrm{f}} : \mathcal{X}_a \times \mathcal{X}_a \to \mathbb{R}$.

1.3 When is each perspective is useful?

The second perspective is certainly the more general one (although we started with the first perspective, we could equally have derived it from the second – they are equivalent for this class of models). The first perspective arguably has the appeal of yielding slightly more compact notation, and aught to be useful in the production of efficient code e.g. by exploiting fast implementations of operations on matrices and vectors. However, the second perspective has the crucial benefit of fitting in well with all of the machinery used for single-output GPs – all that is required is a specifically-chosen domain $\mathcal{X}_a$ and mean function / kernel, all of which are perfectly normal things to do with single-output GPs. This becomes particularly important when considering how to *implement* multi-output GPs inside an existing software, in the context of abstractions that exist for single-output GPs. By framing multi-output GPs as single-output GPs we aught to be able to recycle much existing code – fallback implementations, plotting recipes, approximate inference schemes for non-Gaussian likelihoods, pseudo-point approximations, test suites and, arguably most crucially, all of the existing user- and developer-facing API.

Certainly there is extra code that must be implemented to create multi-output GPs – in particular new abstract vector types to express certain types of input structure (e.g. the same input repeated for each output or a subset-thereof) and the logic to exploit such structure, and obviously the kernels that know about these structures. There is not, however, a need to determine ways to represent missingness in outputs and there does at least *exist* a way to perform all of the computations that you might want to for a given MOGP, with the option to carefully accelerate the ones that matter for overall performance.

Bibliography

Seeger, Matthias, Yee-Whye Teh, and Michael Jordan (2005). *Semiparametric latent factor models*. Tech. rep.