Inducing Clusters Deep Kernel Gaussian Process for Longitudinal Data

Junjie Liang, Weijieying Ren, Hanifi Sahar, Vasant Honavar

The Pennsylvania State University
jliang282@outlook.com, {wjr5337, szh6071, vuh14}@psu.edu

Intuition of Assuming Mixture of Gaussians (MoG) in Latent Space

Lemma 2 assumes that data follows a Mixture of Gaussians (MoG) distributions in their latent space $e(\mathcal{X})$. The intuition behind such an assumption is that, unlike input space, the low-dimensional latent representation is dense and continuous. When mapping data from the input space into a latent space, Euclidean distance is often chosen as the distance metric (Xie, Girshick, and Farhadi 2016). From a generative process viewpoint, the Euclidean distance depicts the variance between data points generated by the same Gaussian distribution. When data in latent space display cluster structure, the generative process can be seen as a Mixture of Gaussians (MoG) (Jiang et al. 2017). In our relaxed formulation (*i.e.,* (3)), a Gaussian prior is used to ensure the latent space follows an MoG prior. Please see more details in the following section.

Formulation of Applying Gaussian Prior to the Latent Representation

Lemma 2 shows that $\mathcal{L}_2$ is equivalent to $\mathcal{L}_1$ when latent space $e(\mathcal{X})$ exhibits a Mixture of Gaussian (MoG) structure. Specifically, in the proof of Lemma 2, we showed that the critical assumption lies in that the data point $\boldsymbol{x} \in \mathcal{X}$ can be generated with $e(\boldsymbol{x}) = e(\boldsymbol{z}^*) + \boldsymbol{\xi}$, where $\boldsymbol{\xi}$ follows a zero-mean Multivariate Gaussian Distribution. With sufficiently small η and ϵ, the inducing clusters are forced to be mutually independent and associated with the cluster centers among the latent space of the training data. Therefore, to enforce a Mixture of Gaussian (MoG) prior to the latent space $e(\mathcal{X})$, we need to introduce a Gaussian prior to each data point centered towards its closest inducing clusters.

Let $e(\boldsymbol{z}^*)$ be the closest inducing clusters for data point $e(\boldsymbol{x})$, the Gaussian prior for data point can be formulated as:

$$e(\boldsymbol{x}) \sim \mathcal{N}(e(\boldsymbol{z}^*), \sigma_x^2 I) \tag{1}$$

Here σ_x is the model parameter introduced along with the Gaussian prior. Assuming the Gaussian prior is applied to each training data independently, then the log-likelihood of Gaussian prior on the entire training data can be represented by:

$$\mathcal{L}_{\text{Gau}}(e(\mathcal{X})) = \log \prod_{\boldsymbol{x} \in \mathcal{X}} \mathcal{N}(e(\boldsymbol{z}^*), \sigma_x^2 I) \tag{2}$$

where $\boldsymbol{z}^* = \arg\max_z K_{xz}$.

Combining the Gaussian prior to the relaxed optimization formulation, we arrive at the final objective function:

$$\begin{aligned} \arg\min_{\Theta} \mathcal{L}_3 = & -\mathbb{E}_{q(\mathbf{f},\mathbf{u})}[\log p(\boldsymbol{y}|\mathbf{f})] + \text{KL}[q(\mathbf{u})||p(\mathbf{u})] \\ & + \lambda_1 \max \text{diag}(BB^\top) - \lambda_2 \min_{x \in \mathcal{X}} \max_{z \in \mathcal{Z}} s_{xz} - \lambda_3 \mathcal{L}_{\text{Gau}}(e(\mathcal{X})) \end{aligned} \tag{3}$$

To summarize, the learnable model parameters in Θ includes (i) parameters in the kernel function. This includes parameters in the encoder network $e^{(v)}$, individual embeddings $e^{(i)}$, kernel coefficients $\alpha^{(v)}, \alpha^{(i)}$, and length scale parameters in the RBF kernels. (ii) Parameters related to the inducing clusters, *i.e.,* the variational prior $\boldsymbol{m}_Z$ and S and embeddings for the inducing clusters Z.

Implementation Details and Parameter Setup

The pseudo-code of ICDKGP is presented in Alg. 1. We implement ICDKGP using PyTorch (Paszke et al. 2019). For ICDKGP, the state encoding size for the mean model is chosen from $\{10, 30, 50\}$ based on the performance in the validation set. Both the mean model and GP models are trained for $2,000$ iterations with early stopping. We use $M = 10$ inducing clusters for all data sets. τ is picked from $\{0.3, 0.5, 0.8, 0.9, 1\}$, $\lambda_i, i = 1, 2, 3$ are picked separately from $\{0.001, 0.01, 0.1, 0.5, 1, 3\}$. In our experiments, we observe that the variance parameter σ_x for $x \in \mathcal{X}$ related to $\mathcal{L}_{Gau}$ has a strong negative impact on the model performance when being learned freely. For a stable performance, we fix σ_x to 0.1 on all experiments. All hyper-parameters corresponding to the deep kernel including dimensions of latent space and network structures are the same as in (Liang et al. 2021). All model parameters are updated using the Adam optimizer. Learning rates are picked from $\{0.01, 0.001\}$. All hyperparameters are chosen with a validation set.

As for the implementation of our baseline methods, we use the implementations of GLMM, GEE, and LGPR available in the `lmer4`, `PGEE` and `lgpr` packages, respectively from CRAN.[1]. We use the LMLFM implementation from https://github.com/junjieliang672/LMLFM. Implementation of SVGP and DSVGP can be found through Gpytorch (Gardner et al. 2018) at https://docs.gpytorch.ai/en/v1.2.1/examples/04_Variational_and_Approximate_GPs/SVGP_Regression_CUDA.html. Implementation for SKIPGP can be found at https://docs.gpytorch.ai/en/v1.2.1/examples/02_Scalable_Exact_GPs/Scalable_Kernel_Interpolation_for_Products_CUDA.html. For GLMM, we keep most hyper-parameters to their default values but increase the maximum iteration to 200. In GEE, we use a first-order auto-regressive correlation structure. The maximum iteration is fixed at 200. For SVGP, the number of inducing points is picked from $\{100, 200, 500\}$. The learning rate is picked from $\{0.01, 0.001\}$. For DSVGP, we reuse our encoder network from the time-varying deep kernel component. The hyper-parameters are picked from the same set of candidates as in SVGP. For SKIPGP, the max root decomposition size is fixed at 300 (as used by default). Other hyper-parameters corresponding to the optimizer are also chosen from the same set of candidates as in SVGP.

For real-world data, we conducted KMeans clustering on the latent representation learned from Module 1 with K ranging from 2 to 20. We observe that when $K \geq 8$, the silhouette score seems to be saturated. Inspired by the fact that L-DKGPR achieves good performance with only 10 inducing points, we fixed the number of including clusters in all our experiments at 10 and initialized the latent representation of inducing clusters to be the centers learned from KMeans. Interestingly, we observe that ICDKGP can effectively learn the representation of inducing clusters to match the number of clusters in the latent space. From Fig. 4, we observe that the number of clusters in the T-SNE visualization is consistently less than 10.

All experiments are conducted on a desktop machine with AMD Ryzen9 5900X CPU, 32GB RAM, and RTX 3070 graphics card.

Complexity Analysis for inducing clusters vs. inducing points

Assuming N, M, d is the training/testing size, inducing point/cluster size, and latent space size respectively. Typically, we have $N >> M \approx d$. The major computation blocks for inducing points are K_{XZ} ($O(NMd)$) and K_{ZZ}^{-1} ($\max(O(M^3), O(M^2d))$. Thus computational complexity for inducing points per iteration is $O(NMd)$. Inducing clusters introduces a few more steps, *i.e.,* representation mapping ($O(NM)$) and computing the regularization terms ($O(NMd)$). Therefore, the overall computational complexity remains the same as inducing points. For space complexity, both methods would need to store the latent representation $O(Nd)$, K_{XZ} ($O(NM)$) and K_{ZZ} ($O(M^2)$), thus the space complexity is $\max(O(Nd), O(NM))$. In conclusion, both the time and space complexity are scaled linearly to the data size.

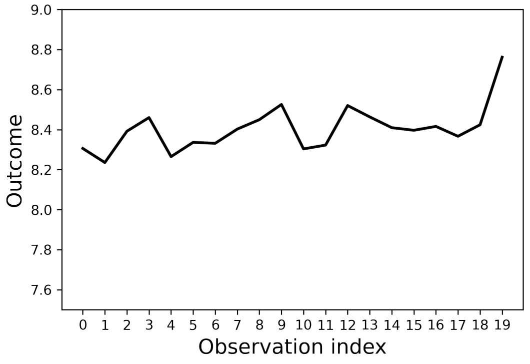
[1]https://cran.r-project.org/

Algorithm 1: ICDKGP

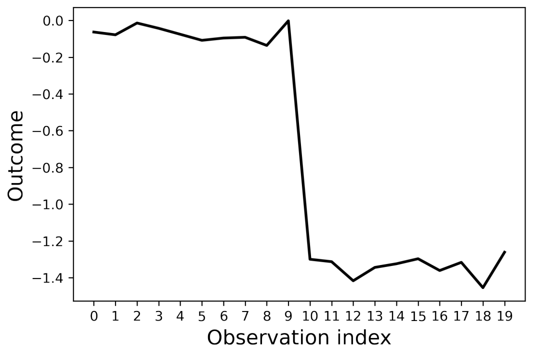
```
Input: Training set $S = \{X, \boldsymbol{y}\}$, latent dimension
    $D_v, D_i$, number of inducing points $M$,
    gradient-based optimizer and its related
    hyper-parameters (i.e., learning rate, weight
    decay, mini-batch size), regularization
    coefficients $\lambda, \beta$, temperature $\tau$.
Output: $\Theta$.
1 Initialize the parameters $\hat{\Theta}$    // Initialize
    either randomly or with
    pre-trained parameters
2 Train the Zero-mean Deep Kernel GP (Module 1) with
    longitudinal kernel in Eq. (1) by minimizing the
    ELBO in Eq. (3)    // See description in
    Module 1 for details.
3 Train the State-space Mean Function (Module 2) with
    the output latent representation $e(X)$ from Module 1
    // See description in Module 2 for
    details.
4 Fixed the parameters of Module 2 and use it as mean
    function for ICDKGP.
5 Train ICDKGP (Module 3) with its encoder network
    initialized by the pre-trained encoder network in
    Module 1    // Objective function of
    Module 3 is given by Eq. (10).
```

Experimental Data Setup

Generating Simulated Data. We construct simulated longitudinal data sets that exhibit *i.e.,* longitudinal correlation (LC) and multilevel correlation (MC) as follows: The outcome is generated using $\boldsymbol{y} = f(X) + \boldsymbol{\epsilon}$ where $f(X)$ is a non-linear transformation based on the observed covariate matrix X and the residual $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \Sigma)$. To simulate longitudinal correlation, we simply set Σ to a block diagonal matrix. For each individual, we use a first-order auto-regressive correlation structure (AR(1)) with decaying factor fixed at 0.9. To simulate a data set that exhibits multilevel correlation, we first split the individuals into C clusters. We then define the cluster correlation matrix by setting the correlation associated to data points in the same cluster to 1. Finally, we compute the multilevel correlation by summing up the longitudinal correlation and cluster correlation. To simulate non-smooth target function across the observations per individual, we split the observations for each individual into 2 clusters and set the correlation associated to data points in the same cluster to 1. Following (Cheng et al. 2019; Timonen et al. 2019), we simulate 40 individuals, 20 observations, and 30 covariates for each individual. To simulate correlation among the covariates, we first generate 10 base features independently from $[0, 1)$ uniform distribution, then the covariate matrix X is computed using an encoder network with architecture $10 - 100 - Tanh - Dropout(0.7) - BatchNorm - 30 - Tanh$. It therefore results in 30 covariates that are conditionally independent given encoder network and base features. We hold out both the base features and the encoder network to all comparing methods, thus leading to a covaraite matrix with non-linear

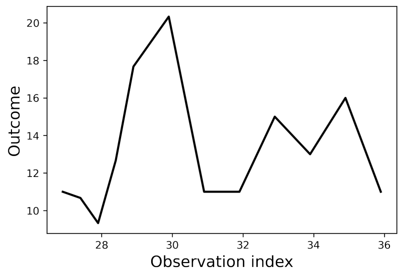


(a) Smooth observations

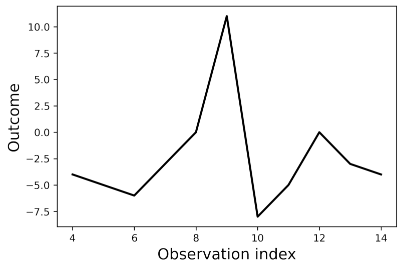


(b) Non-smooth observations

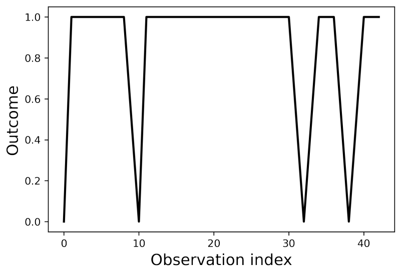
Figure 1: Simulated examples of outcomes from a single individual.



(a) TADPOLE



(b) SWAN



(c) GSS

Figure 2: Real world examples of non-smooth outcome transitions over time from the observations for a single individual.

correlation that is unknown to all methods. To generate $\boldsymbol{y}$, we use another nonlinear transformation $f(X)$, which is defined by a network with structure $30 - 100 - Tanh - 1$. In our experiment, We vary the number of clusters C from $[2, 5]$. Examples of simulated outcomes from a single individual is given in Fig. 1.

Pre-processing on SWAN data. Since CESD score is not contained from the original SWAN data, we manually compute the score based on its definition (Radloff 1977). To form the outcome label, we define an adjusted CESD score by $y = CESD - 15$, thus $y \geq 0$ indicates depression. We center $\boldsymbol{y}$ with $\boldsymbol{y} = \boldsymbol{y} - \text{mean}(\boldsymbol{y})$. After computing the label, we exclude all columns that are directly associated to computing the CESD score. We convert the categorical features using one-hot encoding and perform standard scaling on the continuous features.

Pre-processing on GSS data. Since the original data set contains repeated columns for the same survey question, we keep only one column for each survey question. We re-format all the answer codes associated to 'unknown' and 'missing' to 'unknown'. The outcome label is derived from the field 'General Happiness', we code the value 'pretty happen' and 'very happy' to 1 and the others to -1. As the other covaraites, We convert the categorical features using one-hot encoding and perform standard scaling on the continuous features.

Pre-processing on TADPOLE data. There are three data sets in the original files. We first combine the three data sets and remove the repeated data points. Then, we convert the categorical features using one-hot encoding and perform standard scaling on the continuous features. The outcome label is defined by the value of 'ADAS13'. Similarly, we center the label with $\boldsymbol{y} = \boldsymbol{y} - \text{mean}(\boldsymbol{y})$.

Examples of non-smooth outcome transition for each of the real-world data can be found in Fig. 2.

References

Cheng, L.; Ramchandran, S.; Vatanen, T.; Lietzén, N.; Lahesmaa, R.; Vehtari, A.; and Lähdesmäki, H. 2019. An additive Gaussian process regression model for interpretable non-parametric analysis of longitudinal data. *Nature communications*, 10(1): 1798.

Gardner, J.; Pleiss, G.; Weinberger, K. Q.; Bindel, D.; and Wilson, A. G. 2018. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 7576–7586.

Jiang, Z.; Zheng, Y.; Tan, H.; Tang, B.; and Zhou, H. 2017. Variational deep embedding: an unsupervised and generative approach to clustering. In *Proc IJCAI*, 1965–1972.

Liang, J.; Wu, Y.; Xu, D.; and Honavar, V. G. 2021. Longitudinal deep kernel Gaussian process regression. In *Proc. AAAI*, volume 35, 8556–8564.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS 32*, 8024–8035. Curran Associates, Inc.

Radloff, L. S. 1977. The CES-D scale: A self-report depression scale for research in the general population. *Applied psychological measurement*, 1(3): 385–401.

Timonen, J.; Mannerström, H.; Vehtari, A.; and Lähdesmäki, H. 2019. An interpretable probabilistic method for heterogeneous longitudinal studies. *arXiv preprint arXiv:1912.03549*.

Xie, J.; Girshick, R.; and Farhadi, A. 2016. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, 478–487. PMLR.