Deep Gaussian Processes

During our class sessions, two phrases struck a chord with me: that gaussian processes are "distributions over functions" and that a neural networks are "stacks of function that approximate functions" (both probably uttered by Felipe during a pre-class poll debrief). Taken together, I had a vague sense of a link between the two techniques, that happened to very much be the case in "Deep Neural Networks as Gaussian Processes" (Lee et al., 2018).)

In this project, I will focus on two goals: (1) Replicating the Deep Neural Network Gaussian Process from scratch in numpy, and (2) Implementing an MCMC method for hyperparameter optimization. I'm early in my ML journey, and one of the skills I want to develop is to synthesize, reproduce and communicate research insights in a precise way. As a result, I don't reproduce detailed proofs, but I do lean into visualizations and code snippets. As an addition to the report, I have attached an annotated bibliography of the references I used in this project.

To align with grading rubrics, I will also use a small timeseries dataset (Canadian Lynx Pelts), and benchmark my NNGP implementation against standard techniques.

Problem Specification

A Gaussian process ($\mathcal{GP}$) is a non-parametric modelling technique where each point $\{x_1, x_2 \dots x_N\}$ is assigned a random variable $\{f(x_1), f(x_2) \dots f(x_n)\}$, and the joint distribution of these functions is a Gaussian.

$$z_i^l(x) = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l \, x_j^l(x) \quad , \quad x_j^l(x) = \phi(z_j^{l-1}(x))$$

The key insight of the paper is that each activation layer $z_i^l(x)$ computes a zero-mean GP $z_i^l \sim \mathcal{GP}(0, K^l)$, or, that the covariance at layer l is given by the relation,

$$K^l(x, x') = \sigma_b^2 + \sigma_w^2 F_\phi(K^{l-1}(x, x'), K^{l-1}(x, x), K^{l-1}(x', x'))$$

where σ_w^2 and σ_b^2 are the noise terms associated with the weights and biases, F_ϕ is a function depending on the non-linearity ϕ. Note that this relation is recursive: the computation of the kernel at the current layer K^l depends on three terms from the previous layer K^{l-1}. The base case for the initial layer of the network is,

$$K^0(x, x') = \sigma_b^2 + \sigma_w^2(\frac{x.x'}{d_{in}})$$

In Appendix B, the paper provides an analytic form for the kernel function F_ϕ under ReLU non-linearity. This analytic form is called the arccosine kernel and is computed using,

$$K^l(x, x') = \sigma_b^2 + \frac{\sigma_w^2}{2\pi}\sqrt{K^{l-1}(x,x)K^{l-1}(x',x')}(\sin\theta_{x,x'}^{l-1} + (\pi - \theta_{x,x'}^{l-1})\cos\theta_{x,x'}^{l-1})$$

$$\theta_{x,x'}^l = \cos^{-1}(\frac{K^l(x,x')}{\sqrt{K^l(x,x)K^l(x',x')}})$$

Bayesian Training and Optimization

For test points x^*, observed data D, we have a multivariate Gaussian with $z^*, \mathbf{z}|x^*, \mathbf{x} \sim N(0, \mathbf{K})$ and its block covariance matrix,

$$\mathbf{K} = \begin{bmatrix} K_{D,D} & K_{x^*,D}^T \\ K_{x^*,D} & K_{x^*,k^*} \end{bmatrix}$$

The block matrix augments the training points with the test points x^*.To make predictions with a new test point x^*, we use the predictive distribution $z^*|D, x^* \sim N(\bar{\mu}, \bar{K})$, with

$$\bar{\mu} = K_{x^*,D}(K_{D,D} + \sigma_e^2\mathbb{I}_n)^{-1}\mathbf{y}$$

This suggests that GPs are simply linear weightings of past observed outputs or the target vector $\mathbf{y}$. σ_e^2 is a noise term we assume, in this project, to be a fixed hyperparameter with value 1. Computing this inverse of $(K_{D,D} + \sigma_e^2\mathbb{I}_n)^{-1}$ makes inference computationally expensive. Unlike with regular neural networks with

deterministic predictions, we can reuse this inverse to recover uncertainty estimates over *each* prediction with,

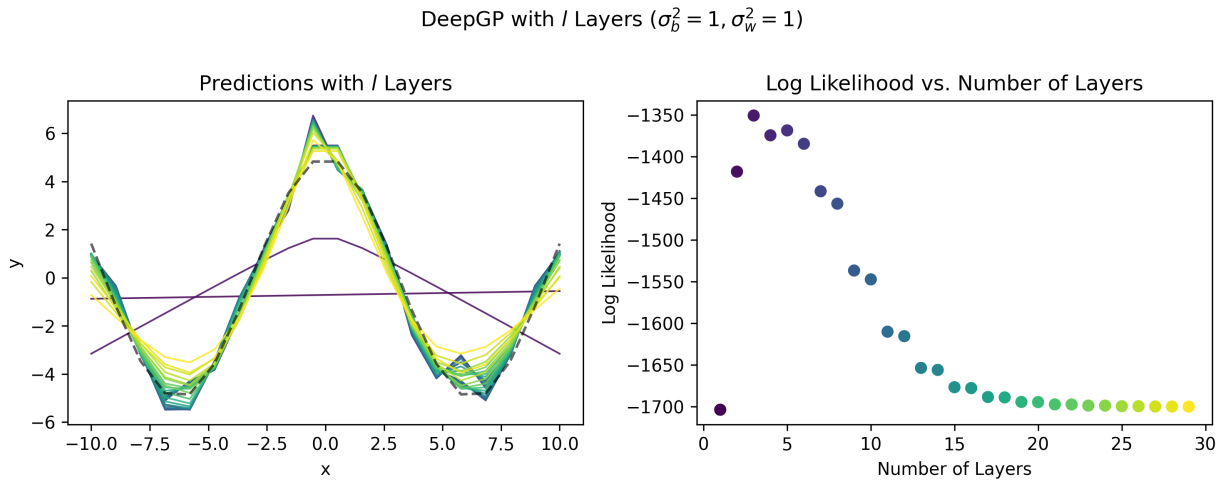
$$\bar{K} = K_{x^*,x^*} - K_{x*,D}(K_{D,D} + \sigma^2 \mathbb{I}_n)^{-1} K_{x^*,D}^T$$

While there are 4 possible hyperparameters: the network depth, nonlinearity, and weight and bias variances, we focus only on the latter two for simplicity (i.e. gradient ascent can be plotted in 3D!). We assume that there are 16 layers and ReLu nonlinearity (for which we have described the analytic form of the kernel above).

To optimize selected hyperparameters $\theta = (\sigma_w^2, \sigma_b^2)$, we can maximize the log-marginal likelihood for a zero-mean GP,

$$\log p(y|X,\theta) = -\frac{1}{2} y^T (K + \sigma_n^2 I) y - \frac{1}{2} \log |K + \sigma_n^2 I| - \frac{n}{2} \log 2\pi$$

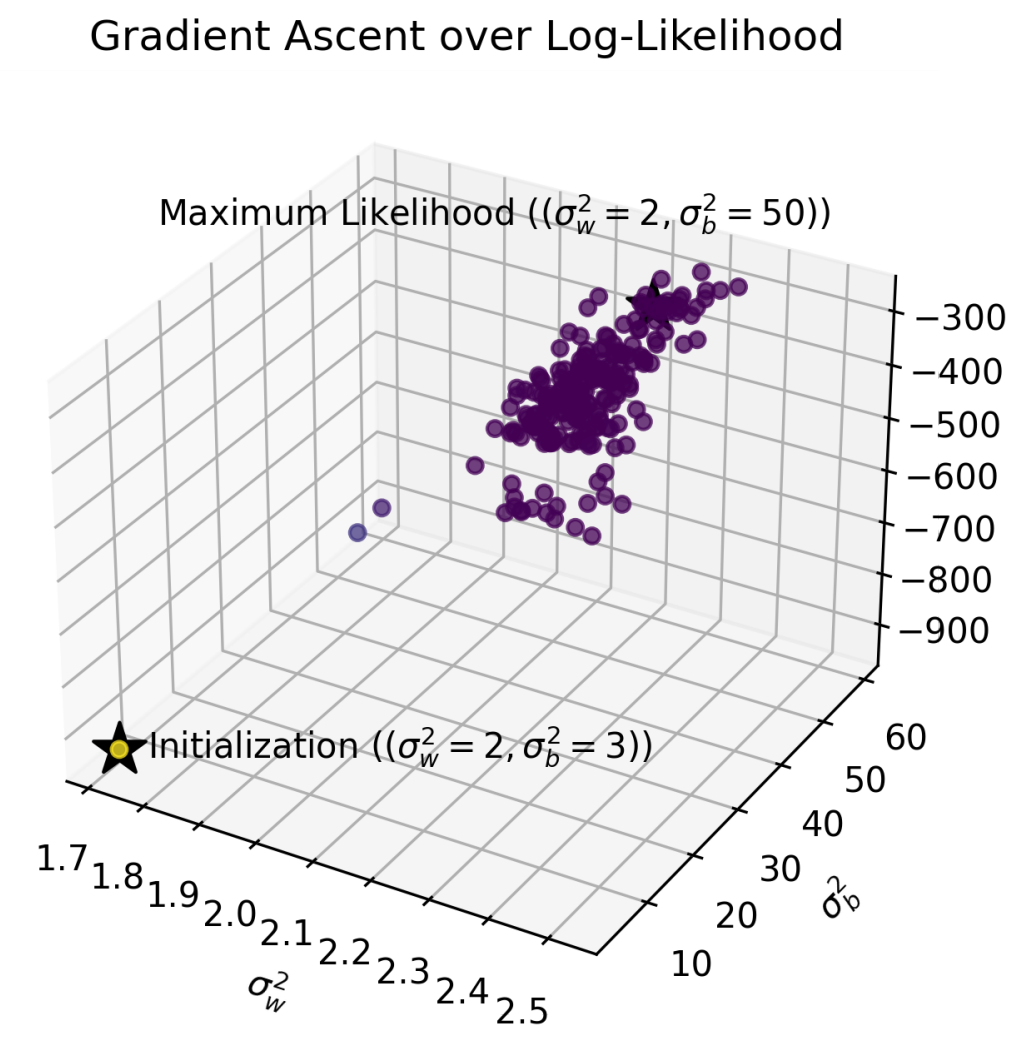
where y is the training target vector/class labels, K is the covariance matrix extracted from the NNGP, I is the identity matrix and σ_n^2 is a noise term added ensure K is non-singular. This is analytically intractable, and has to be estimated numerically with gradient ascent.



Fitting a GP to 5 points from a noisy cosine function f ~ N(5cos(0.5x), 1). For a pair of fixed hyperparameters, increasing the number of hidden layers increases the flexibility of the fit, upto a limit.

To perform gradient ascent, we use a Metropolis-Hastings sampler. In the log-parameter space, a bivariate normal proposal distribution is sampled to yield a new proposal for

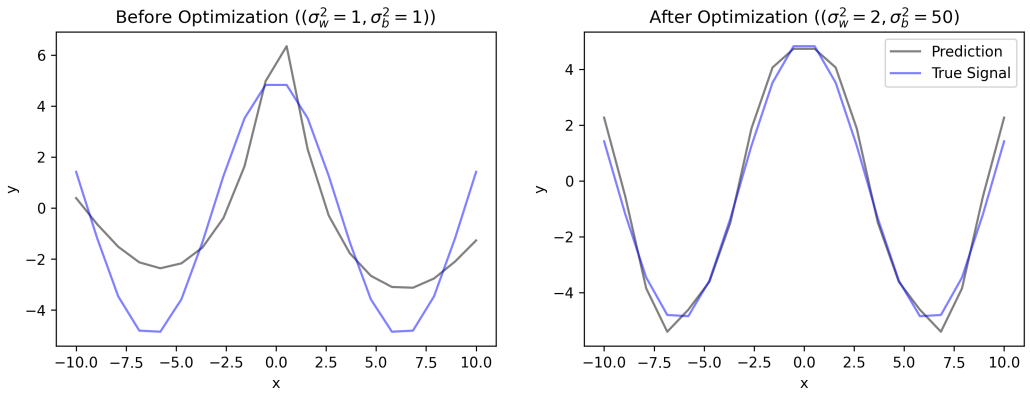
each respective hyperparameter: $\log \sigma_{w*}^2 = \log \sigma_w^2 + N(0, \log(1.5))$ and $\log \sigma_{b*}^2 = \log \sigma_b^2 + N(0, \log(1.1))$.



For the same noisy cosine function, an initialization of (2,3) ascends over time to (2,50) and maximize the log-likelihood.

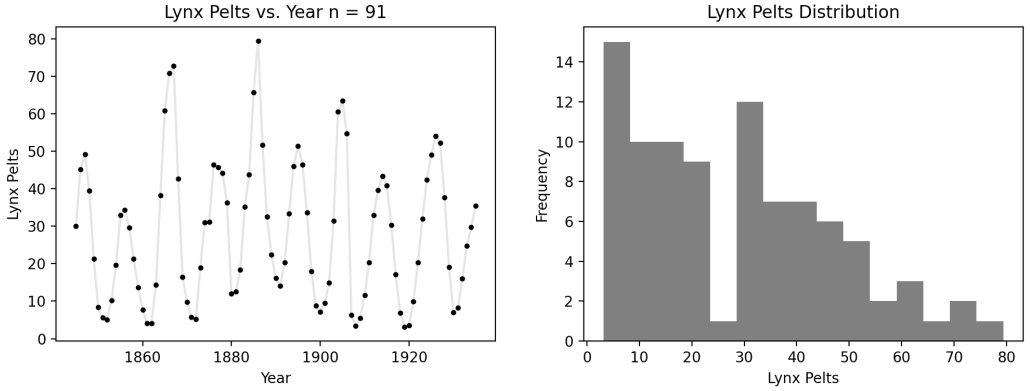
A new NNGP instance is trained on the proposed hyperparameters $\log \theta^* = (\log \sigma_{w*}^2, \log \sigma_{b*}^2)$ and the new log-marginal likelihood $\log p(y|X, \theta^*)$ is computed. If

the new log-marginal likelihood exceeds the previous log-marginal likelihood $\log p(y|X,\theta)$, the parameters are updated with the proposed hyperparameters. If it does not exceed the previous log-marginal likelihood, we accept the proposal with probability $\frac{\log p(y|X,\theta^*)}{\log p(y|X,\theta)}$. Together with the proposal distribution, these stochastic components help prevent the log-marginal likelihood from converging to a local maxima, provided that the bandwith of the proposal distribution is sufficiently wide and the sampler is run for enough iterations.



After updating the hyperparameters to the pair corresponding to the maximum log-likelihood, the fit is substantially improved.

Comparison with Other Techniques



Left: the univariate timeseries of Lynx pelts (the full dataset also contains the number of snowshoe hare pelts over the same timeframe, and is used to test predator-prey models such as the Lokta-Volterra model). The relationship is periodic, with inconsistent amplitude, making it difficult to model parametrically without some insight into the underlying behaviour.

The *Lynx-Hare Pelt* dataset is a timeseries of the number of pelts collected in Canada between 1821 and 1934 by trappers in the Hudson's Bay Company, which was heavily involved in the fur trade. Its small size ($n = 95$) lends well to a GP application.

$$\text{MSE} = y - \hat{y} = y - \bar{\mu}$$

Unlike other LPMs, the Gaussian Process' uncertainty bars gives us more tools to reason and infer about Lynx Pelts.

References

1. Canadian Lynx and Snowshoe Hare Dataset
2. Deep Neural Networks as Gaussian Processes (Lee et al, 2018)
3.