10-708: Probabilistic Graphical Models, Spring 2020

21: Gaussian Processes

Lecturer: Eric Xing *Scribe: Aman Jakhar, Chang Shi, Adam Smoulder, Maxwell Wang, Ni Zhan*

1 Introduction

Assumption-based model approach vs. Probabilistic approach Assumption-based model approaches directly fit a funtion from the data without worrying about how the data is generated, while probailistic approaches provide more explicit guidance on how the data is generated and how the errors behave.

Assumption-based model approach: We guess the parametric form of a function that could fit the data

- $f(x, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} x$ [Linear function of $\boldsymbol{w}$ and x]
- $f(x, \boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}} \phi(x)$ [Linear function of $\boldsymbol{w}$] (Linear Basis Function Model)
- $f(x, \boldsymbol{w}) = g\left(\boldsymbol{w}^{\mathrm{T}} \phi(x)\right)$ [Non-linear in x and $\boldsymbol{w}$] (E.g., Neural Network)

$\phi(x)$ is a vector of basis functions. For example, if $\phi(x) = \left(1, x, x^2\right)$ and $x \in \mathbb{R}^1$, $f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2$ is a quadratic function.

Then choose an error measure $E(\boldsymbol{w})$, minimize with respect to $\boldsymbol{w}$ to estimate the parameters

$$E(\boldsymbol{w}) = \sum_{i=1}^{N} \left[f\left(x_i, \boldsymbol{w}\right) - y\left(x_i\right)\right]^2$$

Probabilistic approach: We could explicitly account for noise in our model.

$$y(x) = f(x, \boldsymbol{w}) + \epsilon(x), \text{ where } \epsilon(x) \text{ is a noise function.}$$

One commonly takes $\epsilon(x) = \mathcal{N}\left(0, \sigma^2\right)$ for i.i.d. additive Gaussian noise, in which case

$$p\left(y(x) | x, \boldsymbol{w}, \sigma^2\right) = \mathcal{N}\left(y(x); f(x, \boldsymbol{w}), \sigma^2\right) \qquad \text{Observation Model}$$
$$p\left(\boldsymbol{y} | x, \boldsymbol{w}, \sigma^2\right) = \prod_{i=1}^{N} \mathcal{N}\left(y\left(x_i\right); f\left(x_i, \boldsymbol{w}\right), \sigma^2\right) \qquad \text{Likelihood}$$

Then maximize the likelihood of the data $p\left(\boldsymbol{y} | x, \boldsymbol{w}, \sigma^2\right)$ with respect to $\sigma^2, \boldsymbol{w}$ tp estimate the parameters.

The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level σ^2 Probabilistic methods thus provide an intuitive framework for representing uncertainty, and model development. In fact, it is also possible to model how x is distributed, thus more flexibility is offered.

Regularization Both assumption-based model approach and probabilistic approach are prone to *over-fitting* for flexible $f(x, \boldsymbol{w})$: low error on the training data, high error on the test set. More complex model gives more flexibility to accomodate the irregular patterns in data, therefore increases the risk of over-fitting on data, thus regularization is introduced to penalize the complexity of the models.

Use a penalized log likelihood(or error function), such as

$$\log p(\boldsymbol{y}|X, \boldsymbol{w}) \propto \overbrace{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(f\left(x_i, \boldsymbol{w}\right) - y\left(x_i\right)^2\right)}^{\text{model fit}} \quad \overbrace{-\lambda \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}^{\text{complexity penalty}}$$

However, there are ambiguity about how we define complexity and how much we should penalize complexity. While choosing the penalty form is very ad-hoc, λ can be set using *cross-validation*.

Bayesian Inference Since these heuristics are ad-hoc, a more rational way of fitting models with the data while taking care of model complexity and penalty would be Bayesian Inference.

According to the Bayes' Rule,

$$p(a|b) = p(b|a)p(a)/p(b) \qquad (\text{posterior} = \frac{\text{likelihood} \ \times \ \text{prior}}{\text{marginal likelihood}})$$

we get $p(a|b) \propto p(b|a)p(a)$,

$$p\left(\boldsymbol{w}|\boldsymbol{y}, X, \sigma^2\right) = \frac{p\left(\boldsymbol{y}|X, \boldsymbol{w}, \sigma^2\right) p(\boldsymbol{w})}{p\left(\boldsymbol{y}|X, \sigma^2\right)}$$

Then the predictive distribution is

$$p\left(y|x_*, \boldsymbol{y}, X\right) = \int p\left(y|x_*, \boldsymbol{w}\right) p(\boldsymbol{w}|\boldsymbol{y}, X) d\boldsymbol{w}$$

Because it's an average of infinitely many models weighted by their posterior probabilities, thereis no over-fitting,and complexity is automatically calibrated. Typically we are more interested in distribution over functions than in parameters $\boldsymbol{w}$.

2 Nonparametric Bayesian Inference and Gaussian Processes

Gaussian Processes are nonparametric Bayesian inference models under particular conditions. In this section, we first review the differences between parametric and nonparametric frameworks, develop intuition of Bayesian (non)parametric inference, then define the Gaussian Process.

Parametric vs. Nonparametric models Recall that parametric models work by assuming that the present data can be explained by a fixed and finite number of parameters, with residuals being explained by noise terms. For example, simple linear regression assumes that the output is a linear combination of the inputs with Gaussian noise added. While the complexity of parametric models can greatly vary (logistic regression versus artifical neural networks), the principle of a fixed parameter set remains.

Conversely, in nonparametric models, the number of parameters typically grows with sample size. Consider kernel regression, where the expected output for a given input is a weighted average of the data outputs, where the weights are determined by distance of the given input from the data inputs. The more data that become available, the more parameters are present, the more complex the operation gets. While this increased complexity with more data can pose computational problems, nonparametric models avoid making the many assumptions enforced by parametric models.

Parametric vs. Nonparametric Bayesian Inference Parametric Bayesian inference specifically refers to modeling distributions for variables (as opposed to just variables) over a finite parameter set using Bayes Rule. Consider most examples, where parameters follow some prior distribution and the likelihood depends on these parameters.

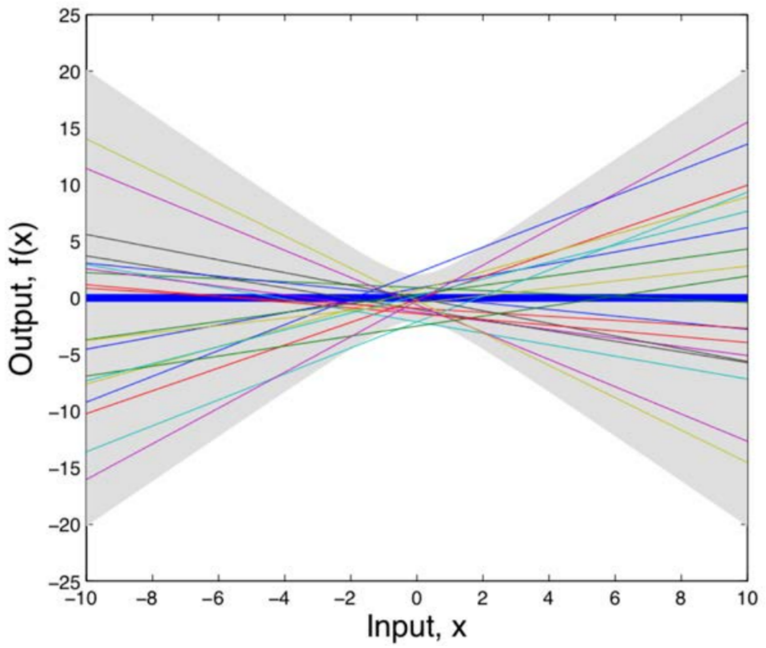
What if, however, the model has an infinite set of parameters? Bayesian nonparametrics are Bayesian models where these finite-dimensional parameters are replaced by a stochastic process. Finite datasets will ultimately only need to use a finite number of parameters, and the rest will be integrated out.

Weight-Space and Function-Space Views Consider a single linear regression model:

$$f(x) = a_0 + a_1 x$$

$$a_0, a_1 \sim N(0, 1)$$

In this case, we can see many possible lines that construct a "weight space", as seen in the figure below:



However, we are more interested in the distribution over functions that is caused by the parameter distributions (a "function-space" view). In the case of this example, we can very clearly determine this distribution. Namely, since both parameters are standard normal, we know:

$$E[f(x|a_0, a_1)] = 0 + 0x = 0$$

$$cov[f(x_b|a_0, a_1), f(x_c|a_0, a_1)] = 1 + x_b x_c$$

This means that we can model our output as a joint Gaussian distribution:

$$[f(x_1), ..., f(x_n)] \sim N(0, K)$$

$$K_{ij} = cov(f(x_i), f(x_j)) = k(x_i, x_j) = 1 + x_i x_j$$

Crucially, we are assuming x is given and not making any statements about the randomness of x. This distribution is due to the randomness in the function. This is a very specific example of a Bayesian Nonparametric approach, but we can generalize this to define Gaussian Processes.

Gaussian Process Definition A Gaussian Process is a collection of random variables, where any finite number of them have a joint Gaussian distribution. A function f is a Gaussian Process with mean function $m(x)$ and covariance kernel $k(x_i, x_j$ if:

$$[f(x_1), ..., f(x_n)] \sim N(\mu, K)$$
$$\mu_i = m(x_i)$$
$$K_{ij} = k(x_i, x_j)$$

Linear Basis Function Models A slightly more complicated example of a Gaussian Process than the one above is case of linear basis functions. Consider:

$$f(x, \mathbf{w}) = \mathbf{w}^T \phi(x)$$
$$p(\mathbf{w}) = N(0, \Sigma_w)$$

Recall that we assume x is given, and hence not random. To define our Gaussian process, determine the mean and covariance for the induced distributions over functions:

$$m(x) = E[f(x, \mathbf{w})] = E[\mathbf{w}^T]\phi(x) = 0$$
$$\begin{aligned} k(x_i, x_j) = cov(f(x_i), f(x_j)) &= E[f(x_i)f(x_j)] - E[f(x_i)]E[f(x_j)] \\ &= \phi(x_i)^T E[\mathbf{w}\mathbf{w}^T]\phi(x_j) - 0 \\ k(x_i, x_j) &= \phi(x_i)^T \Sigma_w \phi(x_j) \end{aligned}$$

3 Gaussian Processes

3.1 Gaussian Process Graphical model

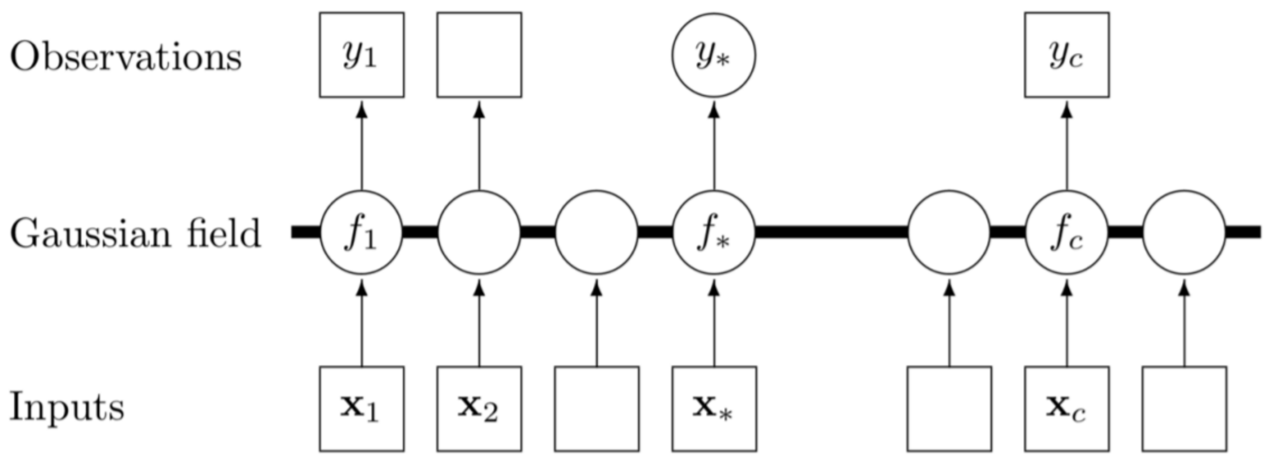


Figure: Gaussian process graphical model

In the above chart y_i represent the observations and x_i represent the inputs. The functions f_i belong to the Gaussian field. When posterior inference is done f_is act as random variables and are integrated out, which means that every prediction y_* depends on all the other inputs and observations.

3.2 Example: RBF kernel

Radial Basis Function (RBF) kernel is by far the most used and popular kernel. It expresess the intuition that the data points which are close together in some distance measure are more correlated. The kerner's hyperparameters a and l control the amplitude and wiggliness (variance) of these functions. These parameters can be learned by optimization through cross validation. If we have extremely large support of x we can approximate any function, i.e. GP with RBF kernel are universal approximators. However such a large covariance matrix presents computational problems. The RBF kernel can be formulated as:

$$k_{RBF}(x_1, x_2) = cov(f(x_1), f(x_2)) = a^2 exp(-\frac{||x_1 - x_2||^2}{2l^2})$$

The hyperparameter l controls how quickly covariance decays with L_2 distance. The impact of length scale on the covariance is shown in the below figure. When the window is very small the covariance drops very quickly.

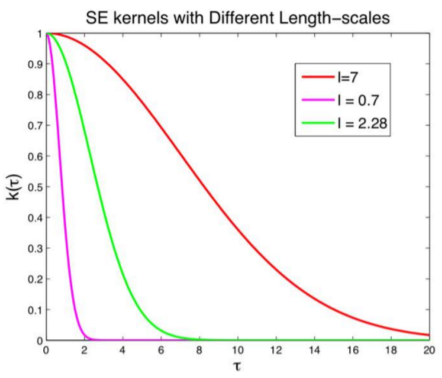


Figure: RBF kernel covariance as a function of distance with varying length scale

This RBF kernel can then be used to create prior distributions which can be used subsequently for modelling and predictions. A Gaussian prior with a mean of 0 and covariance matrix as identity matrix may be constructed as follows:

$$(X_1, X_2.....X_n) \sim \ GP(0, \Sigma)$$

Various sample of this joint distribution look like the below figure. The shaded region represents the uncertainity associated with each sample, whereas the blue line represents the mean function.

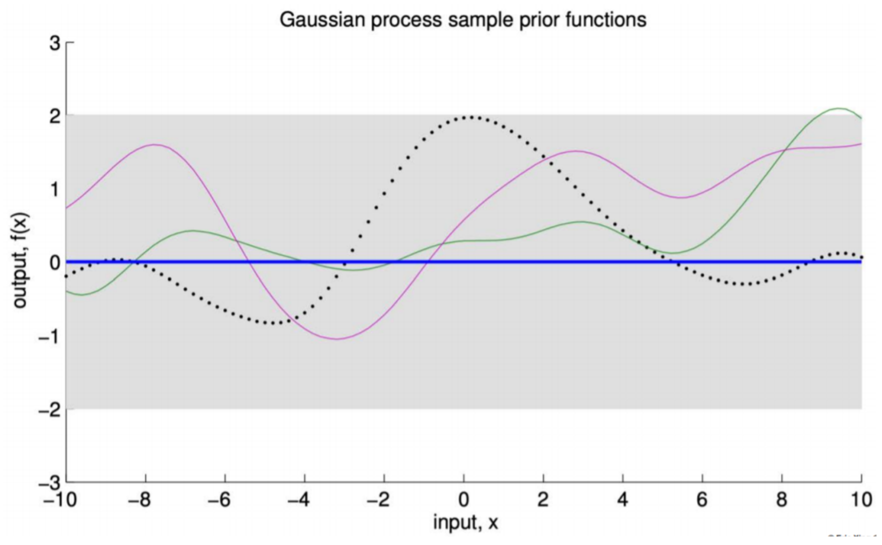


Figure: Different samples of $GP(0, \Sigma)$

3.3 Gaussian Process Inference

The process for inference for a Gaussian Process can be summarized as:

1. Observe noisy data $\mathbf{y} = (y(x_1), y(x_2)....y(x_N))^T$ at input locations X
2. Begin with the standard regression assumption:

$$y(x) = f(x) + \epsilon^2$$

   where ϵ is normally distributed

3. Place a Gaussian process distribution over noise free functions $f(x) \sim GP(0, k_\theta)$. The kernel is parameterized by θ
4. For a new observation X_* infer $p(\mathbf{f}_*|\mathbf{y}, X, X_*)$ which may be decomposed as:

$$p(\mathbf{y}_*|\mathbf{f}_*, X, X_*) = p(\mathbf{y}_*|\mathbf{f}_*)p(\mathbf{f}_*|X, X_*)$$

The conditional distribution for a multivariate Gaussian respresenting the joint ditribution of training data and some extra data point can be written as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_\theta(X,X) + \sigma^2 I & K_\theta(X, X_*) \\ K_\theta(X_*, X) & K_\theta(X_*, X_*) \end{bmatrix}\right)$$

Using this joint distribution one may extract the conditional distribution for f_* which may be written as:

$$\mathbf{f}_*|X_*, X, \mathbf{y}, \theta \sim \mathcal{N}(\bar{\mathbf{f}}_*, cov(\mathbf{f}_*))$$
$$\bar{\mathbf{f}}_* = K_\theta(X_*, X)[K_\theta(X,X) + \sigma^2 I]^{-1}\mathbf{y}$$
$$cov(\mathbf{f}_*) = K_\theta(X_*, X_*) - k_\theta(X_*, X)[K_\theta(X,X) + \sigma^2 I]^{-1} K_\theta(X, X_*)$$

This seems like an expensive calculation with an $N \times N$ matrix inversion.The below figure shows how the posterior of the GP is updated as we observe new data points. Note that variance at the data points is exactly zero.

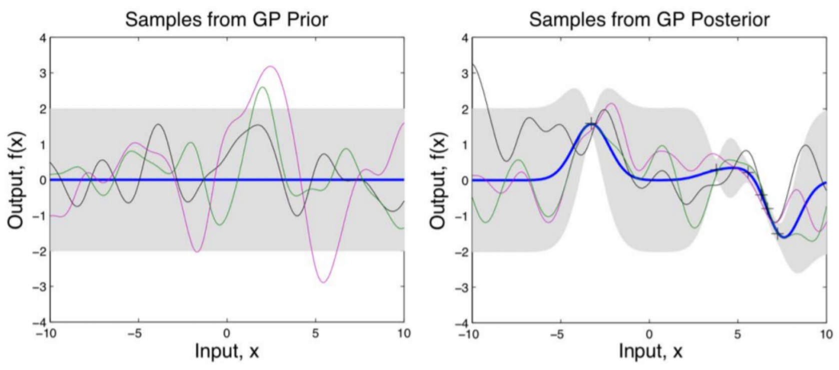


Figure: Change in posterior after introduction of data points

Increasing the length scale has effects similar to those we described in an earlier figure of the RBF kernel. Larger window size of the kernel function allows the points to influence more distant points. The figure below shows that:

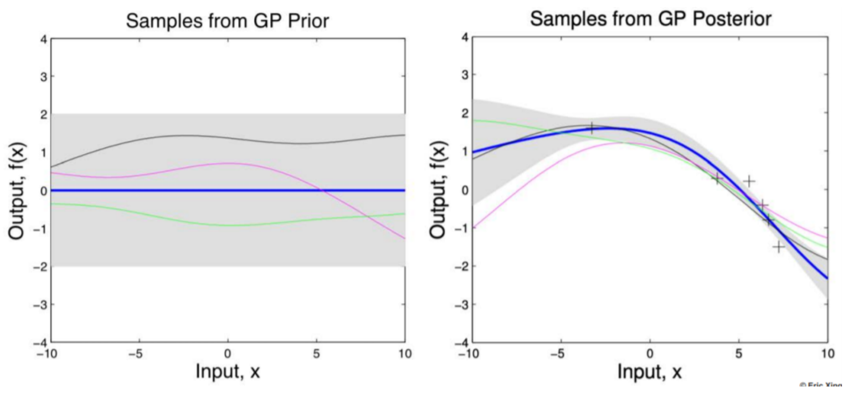


Figure: GP dependence on increasing the length scale l

We realize that we are not specifying the paramateric form of $f(x)$ and are able to utilize the entire training data which are the main advantages of the Gaussian process.

3.4 Gaussian Process Learning

As mentioned before the entire Gaussian process $f(x)$ can be integrated away to obtain the marginal likelihood as a function of kernel hyperparameters alone. This may be written as:

$$p(\mathbf{y}|\theta, X) = \int p(\mathbf{y}|\mathbf{f}, X)p(\mathbf{f}|\theta, X)d\mathbf{f}$$

The log likelihood of this may be written as:

$$logp(\mathbf{y}|\theta, X) = -\frac{1}{2}\mathbf{y}^T(K_\theta + \sigma^2\mathbf{I})^{-1} - \frac{1}{2}log|K_\theta + \sigma^2\mathbf{I}| - \frac{N}{2}log(2\pi)$$

This log likelihood may be minimized over θ to obtain the optimal value. Note the extra complexity penalty in the above term. Various other types of kernels are also used. Some of which are listed below:

1. $k_{SE}(\tau) = exp(-0.5\frac{\tau^2}{l2})$
2. $k_{MA}(\tau) = a(1 + \frac{\sqrt{3}\tau}{l})exp(-\frac{\sqrt{3}}{l})$
3. $k_{RQ}(\tau) = (1 + \frac{\tau^2}{2\alpha l^2})^{-\alpha}$
4. $k_{PE}(\tau) = exp(-2\frac{\sin^2(\pi\tau\omega)}{l^2})$

4 Deep Kernel Learning

We can use GP in conjunction with other techniques. GP itself is a particular way of transforming an input to y. GP gives uncertainty and transforms to a whole distribution, and the function is implicit and has rich representation. We can treat GP as a regular transformer and use it inside an existing transformer framework. For example, in deep kernel learning, we insert a GP layer into a neural network [1], Fig. 1. The GP layer uses a RBF kernel, which achieves the effect of having an infinite dimension NN layer. In NN theory, wider dimensional layers are more expressive for latent representations and allow more influence and correspondences between dimensions. (Here we are referring to dimension of a single data point.) We do not need to plug in an infinite dimension layer but we achieve this effect with the GP layer.

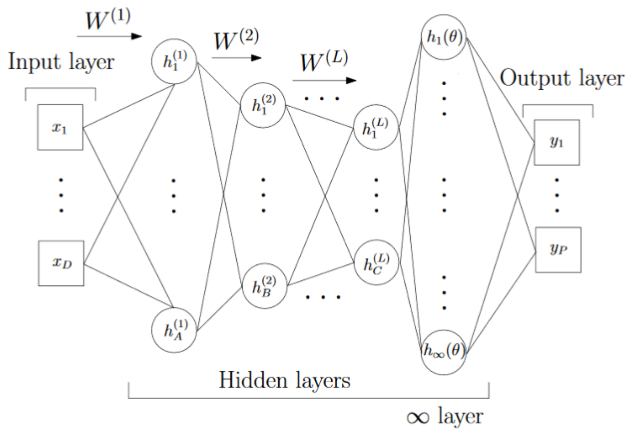


Figure 1: NN with GP layer

We jointly learn the parameters using back-propagation and chain rule. We take derivative of loss function w.r.t. NN parameters and hyperparameters inside GP.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial K_\gamma}\frac{\partial K_\gamma}{\partial \theta}, \quad \frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial K_\gamma}\frac{\partial K_\gamma}{\partial g(\mathbf{x},\mathbf{w})}\frac{\partial g(\mathbf{x},\mathbf{w})}{\partial \mathbf{w}}$$

where θ are parameters of base kernel, $\mathbf{w}$ are weights of network, and L is log marginal likelihood of the GP.

The deep kernel has advantages in regression. In a comparison between GP, deep NN, and deep NN with infinity layer, across multiple datasets, many of the best performing models (in terms of error and variance) were deep kernel learning models. (In Table 1 in paper [1], bold numbers were considered best performance.)

DKL on Sequential Data We can combine GP with sequential model such as RNN or LSTM. In a regular sequential model, we have fixed length input that generates one output. When inputing forward through the sequence, we must drop the earlier part of sequence because the model takes fixed length input. The outputs do not use information from the entire sequence. We can add a GP layer that uses latent representations from h as inputs and maps to the output, as shown in Fig. 2. With the added GP layer, we make use of all inputs up to that point in the sequence. The predictions are derived using conditional GP with mean and covariance as previously described. We get benefits from both sequential NN and GP (which gives correlation and uncertainty). Details in paper [2].

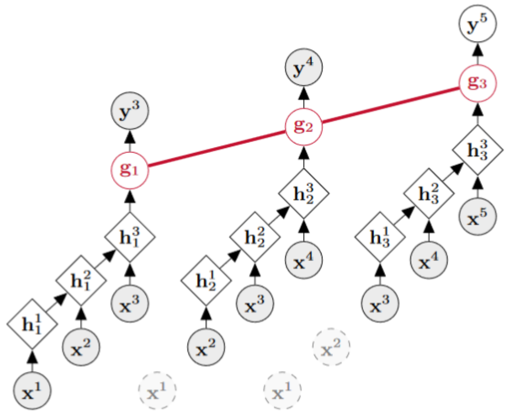


Figure 2: LSTM with GP layer

An example application is autonomous driving.

- We want to predict the lane location in the front, shown in Fig. 3. There is a visible difference in prediction quality/accuracy, with GP-LSTM having higher accuracy. The GP-LSTM also gives error bound (uncertainty) on prediction, not just point prediction.
- Predict lead vehicle distance in Fig. 4. Comparing LSTM and GP-LSTM, the GP-LSTM is higher accuracy and provides uncertainty. The uncertainty is useful because we can take additional actions based on uncertainty. When uncertainty is large, we may want to stay away from the driver.

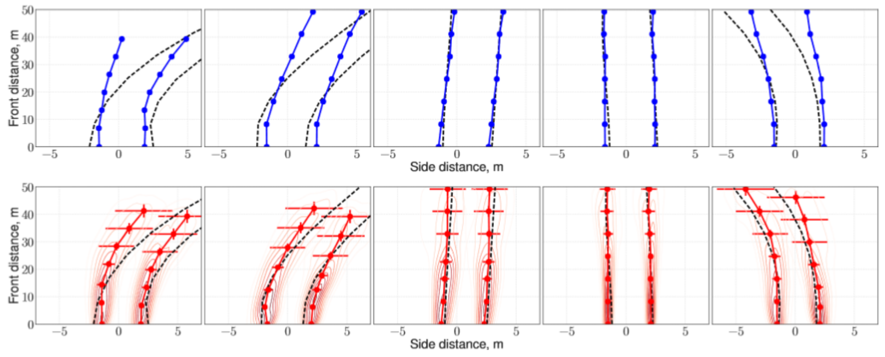


Figure 3: Lane prediction: Blue is standard LSTM, and red is GP-LSTM. Dots indicate prediction.

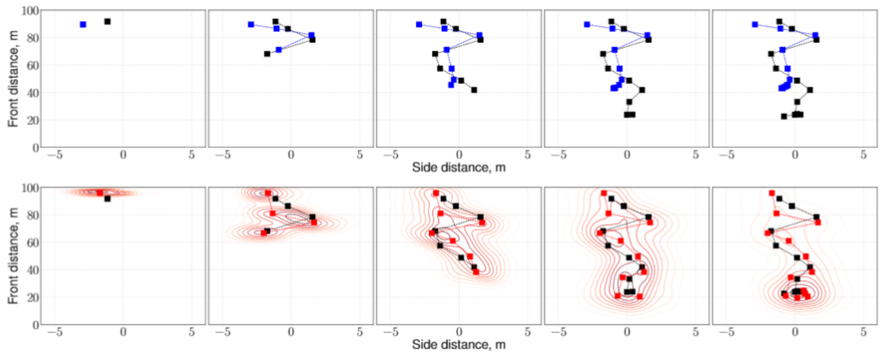


Figure 4: Lead vehicle prediction: Blue is standard LSTM, and red is GP-LSTM. GP-LSTM curves represent uncertainty.

5 Scalability Issues and Solutions

As previously mentioned, Gaussian Processes are a form of nonparametric models, meaning that complexity will increase with the number of data. In this section, we identify scalability issues and identify possible methods of circumventing these.

The Scalability Issue There are bottlenecks in both inference and learning for Gaussian Processes. In inference, the limiting step comes due to the $n \times n$ kernel matrix inversion involved in predicting both the mean and covariance, namely $(K_\theta(X,X)+\sigma^2 I)^{-1}$. This requires $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ storage. Similarly for learning, the same complexity is achieved due to the necessity of calculating $log|K_\theta(X,X)+\sigma^2 I|$ for the marginal likelihoods needed to learn θ. After this calculation, the predictive mean and variance cost $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ per test point.

There are three different families of approaches one can take to avoid the explosion of operations with data:

- Approximate nonparametric kernels in a finite basis "dual space". For m basis functions, this only requires $\mathcal{O}(m^2 n)$ computations and $\mathcal{O}(m)$ storage. E.g. SSGP, Random Kitchen Sinks, Fastfood, A la Carte
- Exploit existing structure in K to exactly (and quickly) solve linear systems and log-determinants. E.g. Toeplitz Method, Kronecker Method. This can reduce time complexity to $\mathcal{O}(1)$ mean and variance predictions and $\mathcal{O}(n)$ for inference and learning in the case of Massively Scalable Gaussian Processes (See Wilson, Dann, Nickisch (2015), Thoughts on Massively Scalable Gaussian Processes).
- Inducing point based sparse approximations. E.g. SoR, FITC, KISS-GP

We will focus on the latter of these.

Inducing Point Methods The goal of inducing point methods is to approximate the Gaussian Process through m inducing points, $\bar{\mathbf{f}}$. These points are used to obtain a Sparse Pseudo-input Gaussian process (SPGP) prior, defined as:

$$p(\bar{\mathbf{f}}) = \int d\bar{\mathbf{f}} \prod_n p(f_n|\bar{\mathbf{f}})p(\bar{\mathbf{f}})$$

Inverting the covariance for a SPGP requires $\mathcal{O}(m^2 n)$ operations.

Typically, selecting inducing points is as simple as making a grid spanned by m selected points. However, higher-dimensional grids would require many inducing points, potentially hurting the efficiency gained by using this method.

Finally, recent efforts have been put forth to parallelize learning and inference for Gaussian Processes such that they can be performed using multiple GPUs. For information, see Wang et al. (2019). Exact Gaussian Processes on a Million Data Points (arXiv:1903.08114).

References

1. Wilson, A. G., Hu, Z., Salakhutdinov, R. & Xing, E. P. Deep Kernel Learning. *CoRR.* arXiv: `1511.02222 [cs.LG]`. `http://arxiv.org/abs/1511.02222v1` (2015).
2. Al-Shedivat, M., Wilson, A. G., Saatchi, Y., Hu, Z. & Xing, E. P. Learning Scalable Deep Kernels With Recurrent Structure. *CoRR.* arXiv: `1610.08936 [cs.LG]`. `http://arxiv.org/abs/1610.08936v3` (2016).