Advanced Machine Learning

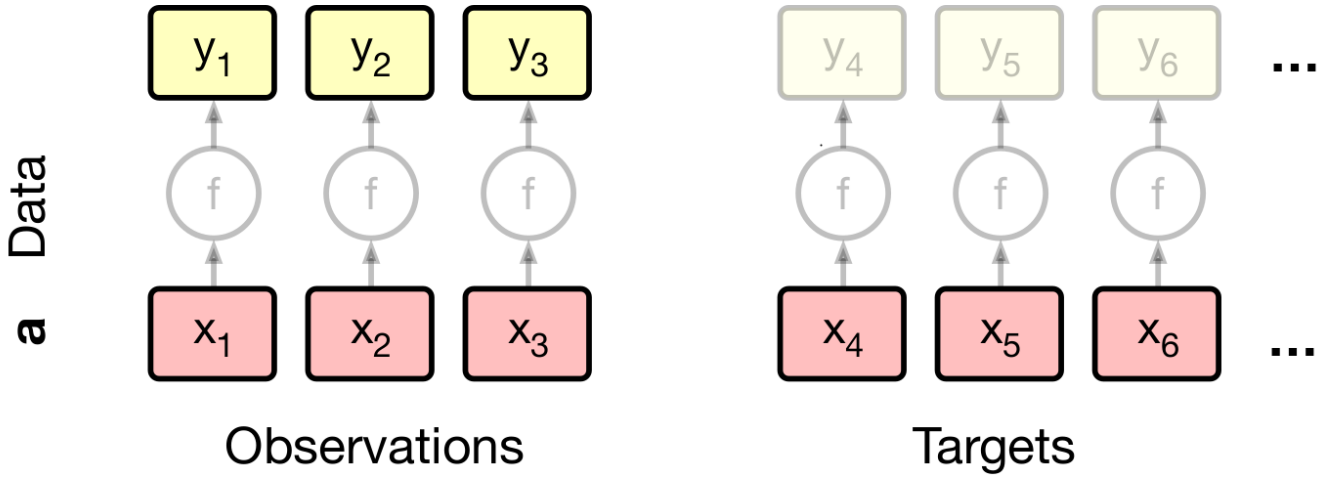
Gaussian & Neural Processes

Paper: Marta Garnelo et al., *Conditional neural processes*,
in International Conference on Machine Learning 2018. [Code]

Gilles Louppe
g.louppe@uliege.be

LIÈGE université

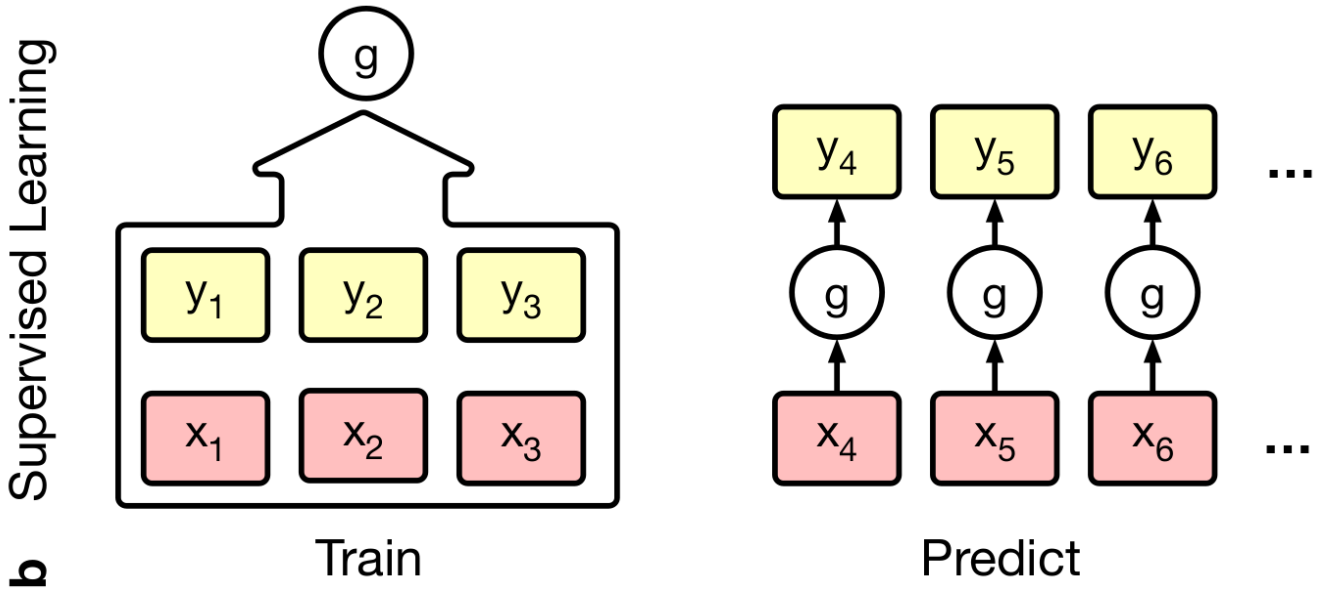
Supervised learning



Supervised learning = function approximation given a finite set of observations.

- Data $\{(x_i, y_i)\}_{i=0}^{n-1}$, with inputs $x_i \in \mathcal{X}$ and outputs $y_i \in \mathcal{Y}$.
- Assume outputs represent evaluations $y_i = f(x_i)$ of some unknown function $f : \mathcal{X} \to \mathcal{Y}$, which may be fixed or a realization of some random function.

—
Credits: Marta Garnelo et al., arXiv:1807.01613.



The traditional approach to SL is to define a parametric function $g : \mathcal{X} \to \mathcal{Y}$ for each new task and spend the bulk of the computation on a costly fitting phase.

- Prior information about f is specified via the architecture of g, the loss function, or the training details.
- Since the extent of prior knowledge that can be expressed in this way is limited, the amount of training data required is usually large.

—
Credits: Marta Garnelo et al., arXiv:1807.01613.

An alternative is to specify a distribution over functions g, also known as stochastic processes.

- The prior knowledge about f is captured in the distributional assumptions of the process prior.
- Learning corresponds to Bayesian inference over the functional space conditioned on the observed values.

Gaussian processes are an example of this approach to supervised learning.

Gaussian processes

Multivariate Gaussian distribution

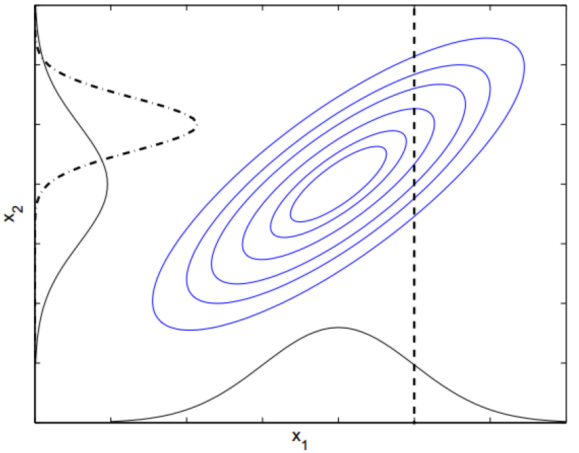
Let $\mathbf{f}$ be a random variable that follows a multivariate Gaussian distribution $p(\mathbf{f})$.

Let us partition its dimensions into two sets A and B, such that:

$$\underbrace{f_1, \dots, f_i}_{\mathbf{f}_A}, \underbrace{f_{i+1}, \dots, f_N}_{\mathbf{f}_B} \sim \mathcal{N}(\mu, K)$$

$$\mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix} \in \mathbb{R}^N$$

$$K = \begin{bmatrix} K_{AA} & K_{AB} \\ K_{BA} & K_{BB} \end{bmatrix} \in \mathbb{R}^{N \times N}$$



Marginal and conditional distributions

- The marginal distribution $p(\mathbf{f}_A)$ of the multivariate Gaussian $p(\mathbf{f})$ is a multivariate Gaussian such that

$$\mathbf{f}_A \sim \mathcal{N}(\mu_A, K_{AA}).$$

- The conditional distribution $p(\mathbf{f}_A|\mathbf{f}_B)$ is a multivariate Gaussian such that

$$\mathbf{f}_A|\mathbf{f}_B \sim \mathcal{N}(\mu_A + K_{AB}K_{BB}^{-1}(\mathbf{f}_B - \mu_B), K_{AA} - K_{AB}K_{BB}^{-1}K_{BA}).$$

Gaussian processes

Let us define a stochastic process as a random function

$$f : \mathcal{X} \to \mathcal{Y},$$

such that for each finite sequence $\mathbf{x}_{1:N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, with $\mathbf{x}_i \in \mathcal{X}$, we define the marginal joint distribution over function values

$$Y_{1:N} := (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)).$$

When these joint distributions are all defined as multivariate Gaussians, the resulting stochastic process is called a Gaussian process.

... or put differently:

A Gaussian process is a (potentially infinite) collection of random variables, such that the joint distribution of any finite number them is multivariate Gaussian.

... or even more simply, a Gaussian process is a

HUGE*

multivariate Gaussian distribution.

* Its dimension is the number of data points, possibly infinite.

Gaussian distribution

vs.

Gaussian process

$$\mathbf{x} \sim \mathcal{N}(\mu, K)$$

- Distribution over vectors.
- Fully specified by a mean and a covariance matrix.
- The position of the random variable in the vector plays the role of the index.

$$f \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$$

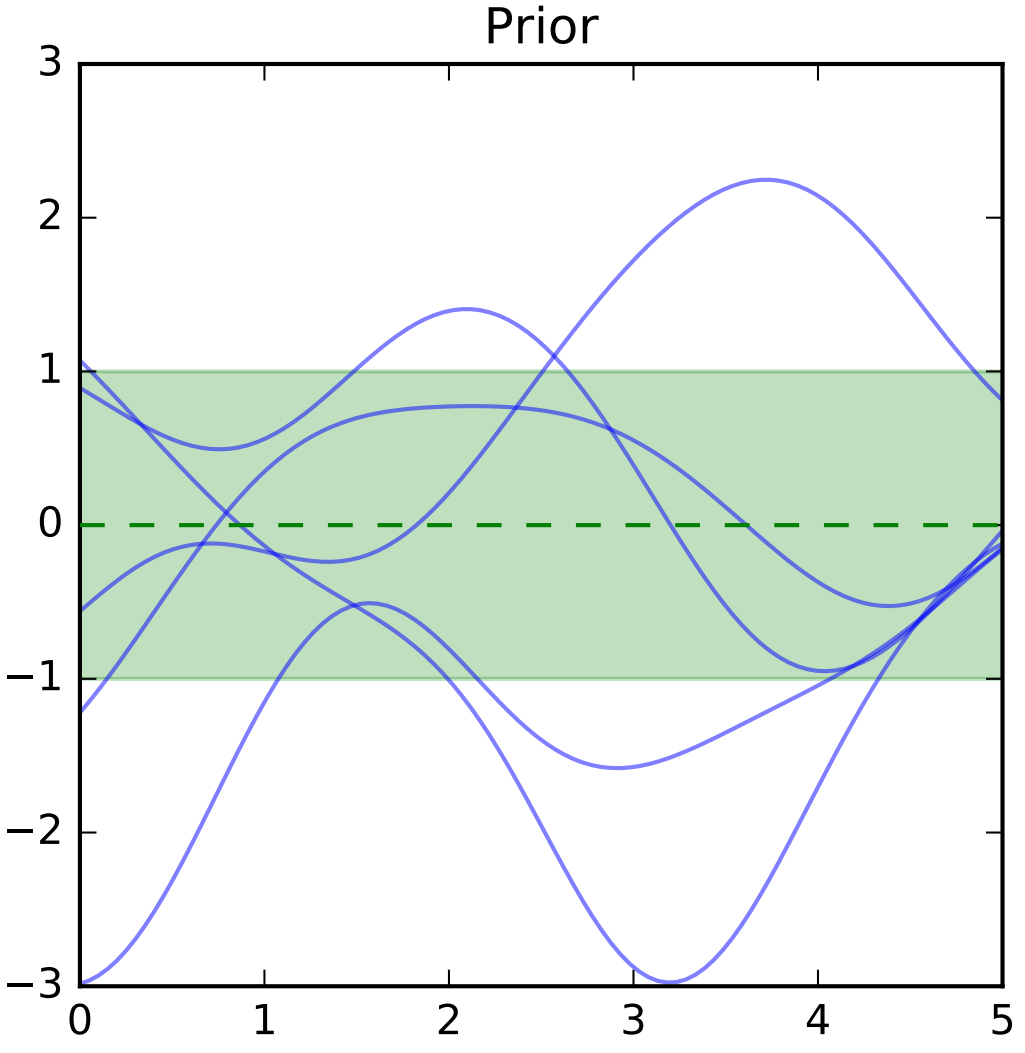
- Distribution over functions.
- Fully specified by a mean function m and a covariance function k.
- The argument of the random function plays the role of the index.

Prior

For $m(\cdot) = 0$ and for any set $A = \mathbf{x}_1, \ldots, \mathbf{x}_M$ of target points, we compute the covariance matrix K_{AA}, which defines a joint distribution $p(\mathbf{f}_A)$ over function values at those points:

$$\begin{bmatrix} f(\mathbf{x}_1) \\ \ldots \\ f(\mathbf{x}_M) \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, K_{AA})$$

That is, we are marginalizing over the random variables not included in the target points.


Prior
3
2
1
0
−1
−2
−3
0
1
2
3
4
5

Posterior

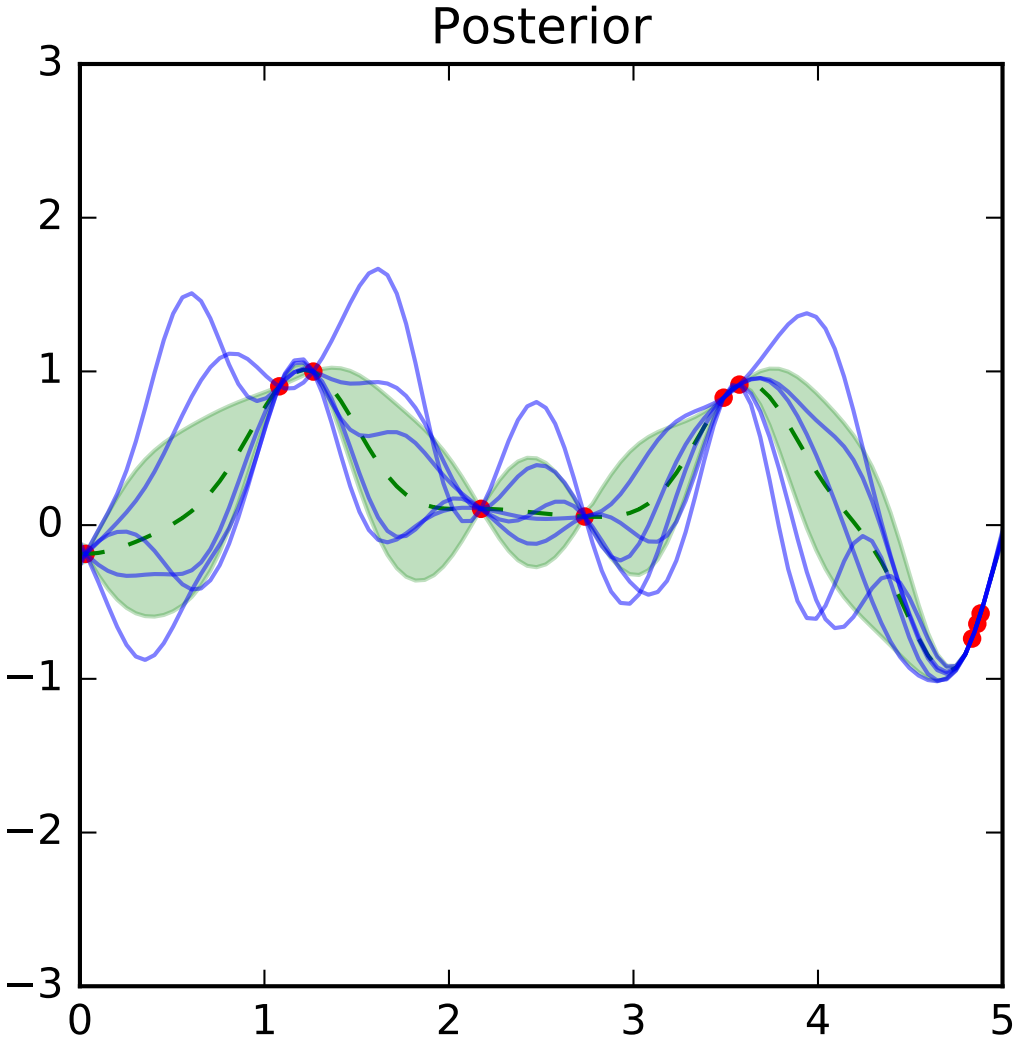
Given observations $\mathcal{D} = (B = \mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{f}_B = f(\mathbf{x_1}), \ldots, f(\mathbf{x_N}))$ and target points $A = \mathbf{x}_1, \ldots, \mathbf{x}_M$, we similarly derive the joint distribution

$$\begin{bmatrix} \mathbf{f}_A \\ \mathbf{f}_B \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} K_{AA} & K_{AB} \\ K_{BA} & K_{BB} \end{bmatrix})$$

on which we can condition $\mathbf{f}_B$ on the known values from $\mathcal{D}$, resulting in the posterior distribution $p(\mathbf{f}_A|\mathcal{D})$:

$$\mathbf{f}_A \sim \mathcal{N}(K_{AB}K_{BB}^{-1}\mathbf{f}_B, K_{AA} - K_{AB}K_{BB}^{-1}K_{BA})$$

Note that due to the inversion of K_{BB}, the complexity for (exact) posterior inference is $O(N^3)$.


Posterior
3
2
1
0
−1
−2
−3
0
1
2
3
4
5

Kernels

The kernel or covariance function $k(\cdot, \cdot)$ encodes the covariance between pairs of random variables $f(\mathbf{x}_i)$, $f(\mathbf{x}_j)$. It must be positive semi-definite and symmetric.

Popular examples include:

- The squared exponential function (RBF)
- The Matern kernel
- The linear kernel
- The polynomial kernel
- The white noise kernel

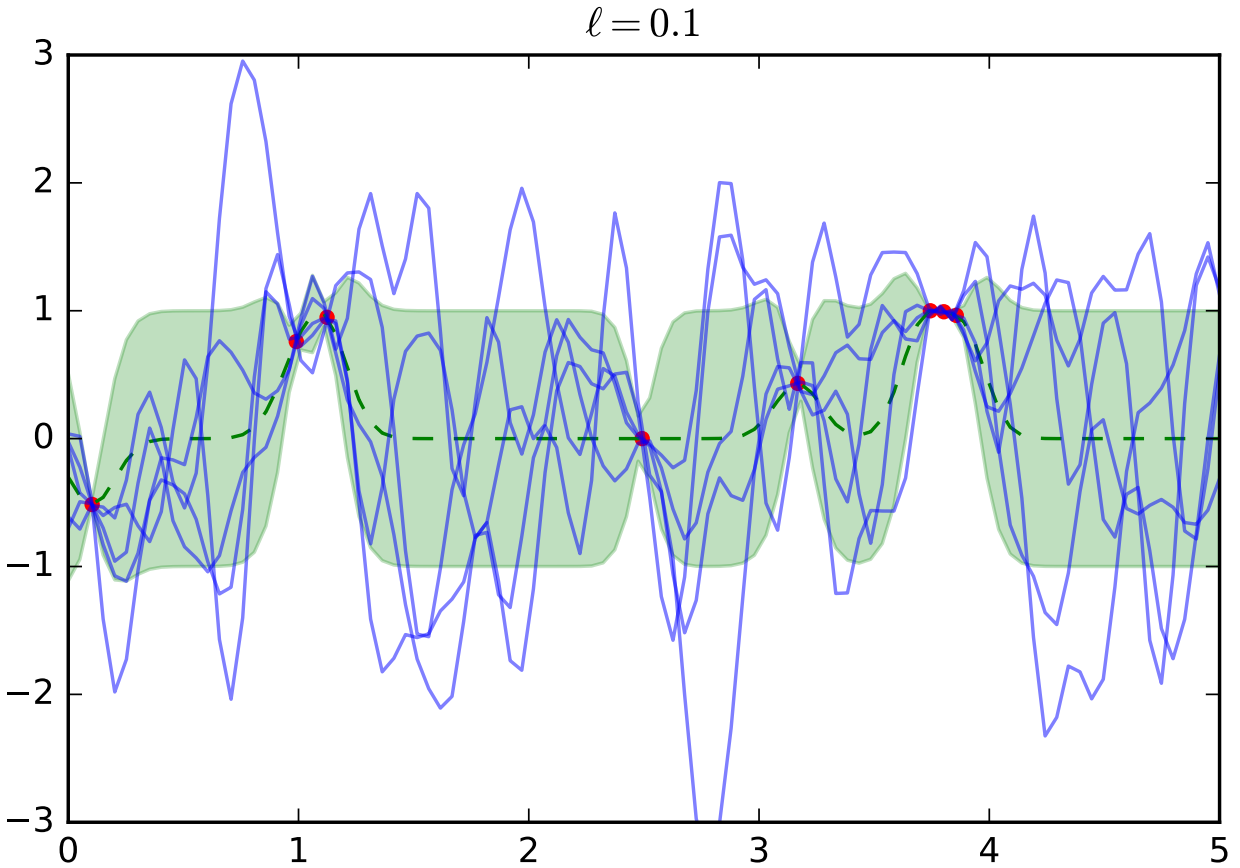
Kernels can be composed together to describe complex interactions. They are a way to incorporate prior knowledge about the class of functions.

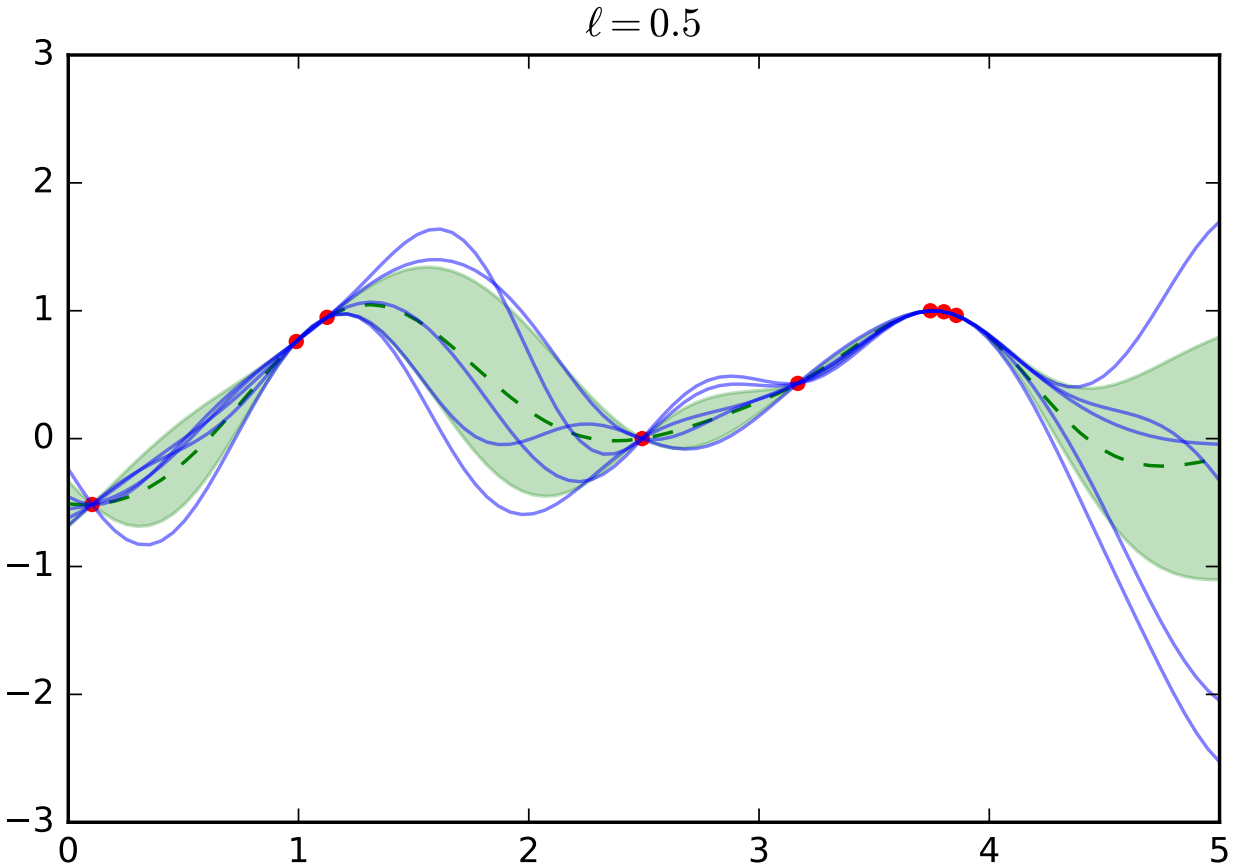
Squared exponential function

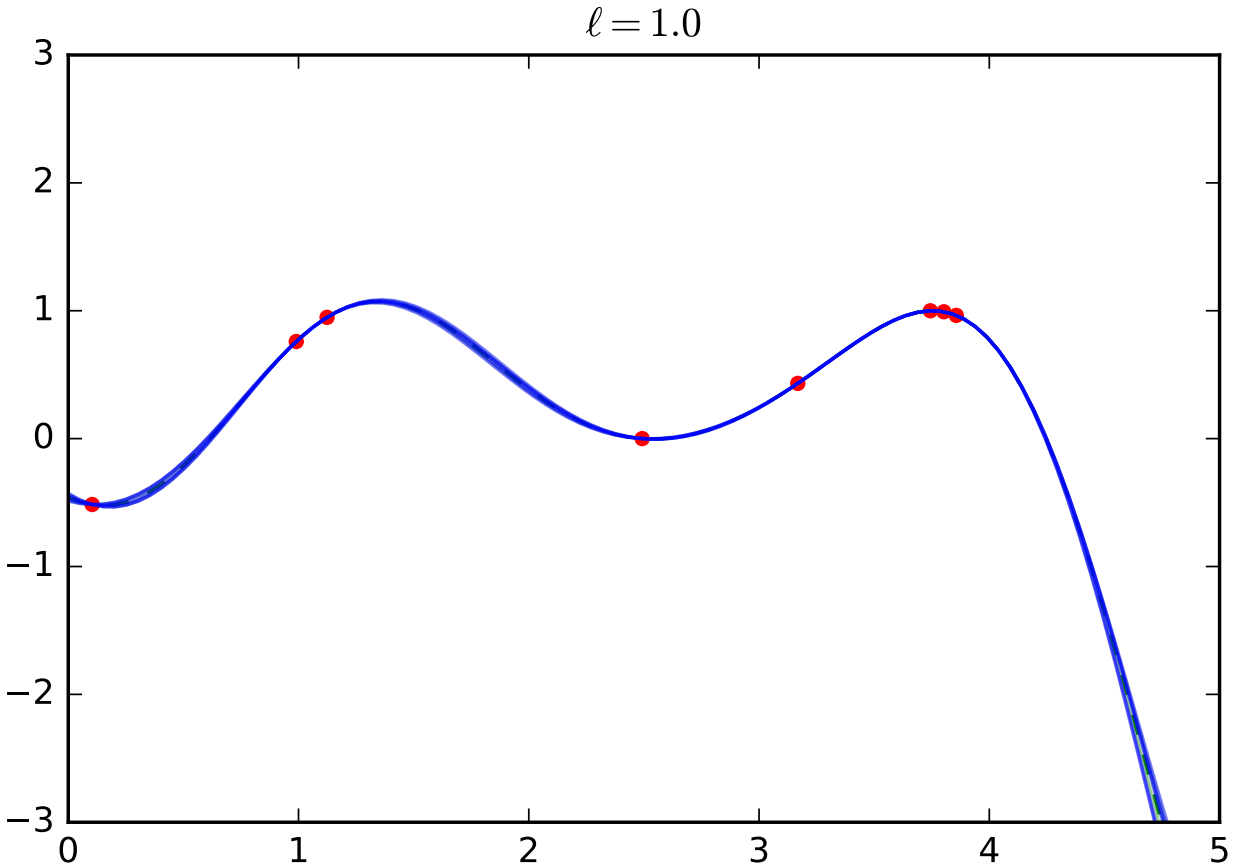
$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\ell^2})$$

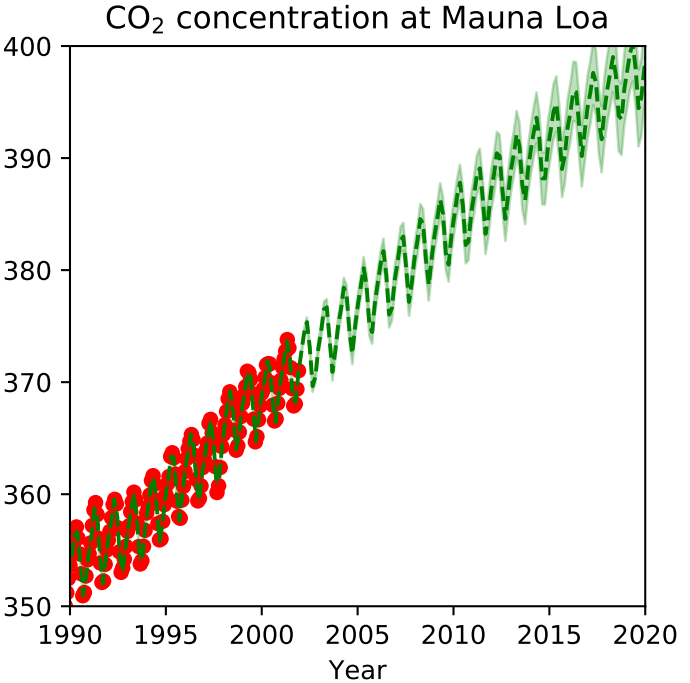
Hyper-parameters:

- The length scale ℓ describes the smoothness of the function.
- The output variance σ^2 determines the average distance of the function away from its mean.


$\ell = 0.1$
3
2
1
0
−1
−2
−3
0
1
2
3
4
5


$\ell = 0.5$
3
2
1
0
−1
−2
−3
0
1
2
3
4
5


$\ell = 1.0$
3
2
1
0
−1
−2
−3
0
1
2
3
4
5



$$
\begin{aligned}
k = & \ 66^2 \mathrm{RBF}(\ell = 67) \\
& + 2.4^2 \mathrm{RBF}(\ell = 90) \times \mathrm{ExpSineSquared}(\ell = 1.3) \\
& + 0.66^2 \mathrm{RationalQuadratic}(\ell = 2, \alpha = 0.78) \\
& + 0.18^2 \mathrm{RBF}(\ell = 0.134) \\
& + \mathrm{WhiteKernel}()
\end{aligned}
$$

Hyper-parameters

- So far we have assumed that the Gaussian process prior $p(\mathbf{f})$ was specified a priori.
- However, this distribution itself has parameters. E.g., ℓ and σ when using the squared exponential function.
- Let θ denote the vector of hyper-parameters. How do we set θ?

Maximum marginal likelihood

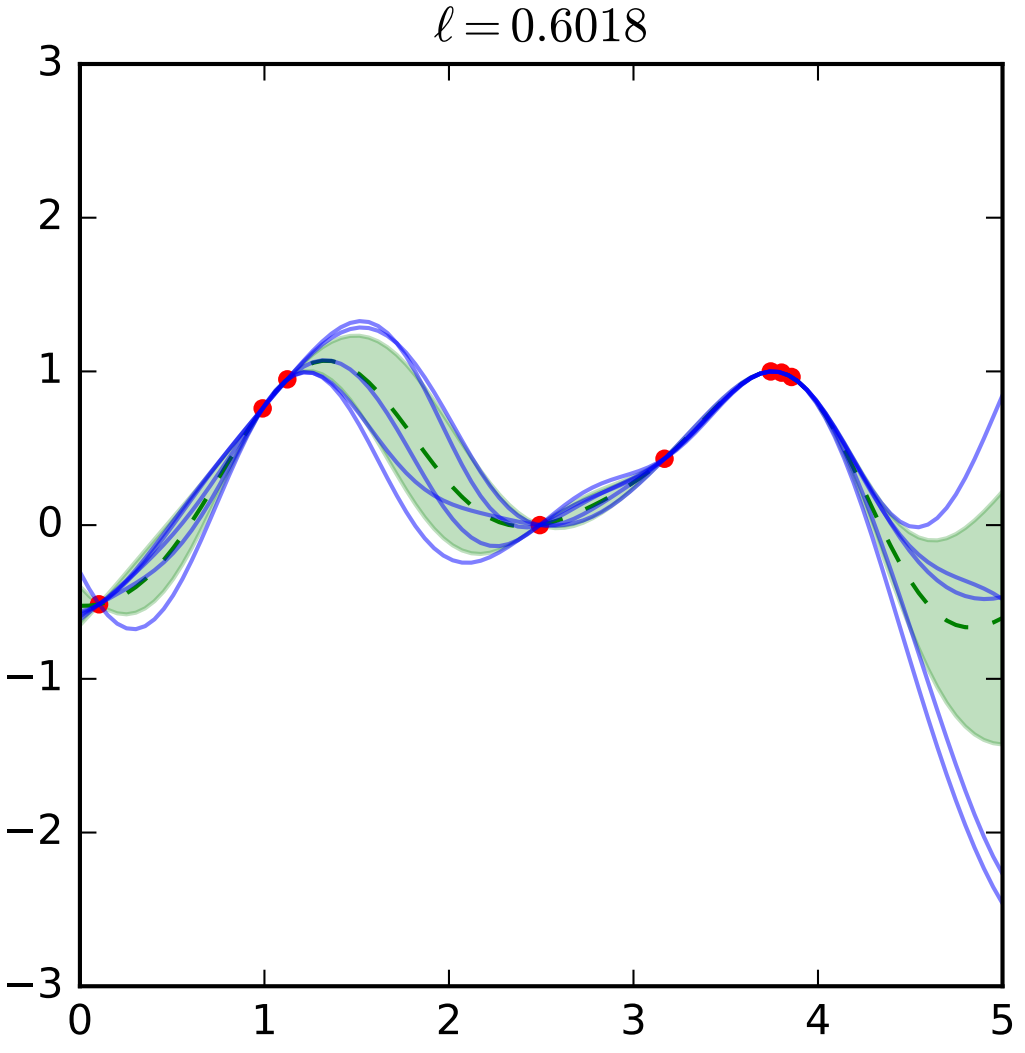
Given observations $\mathcal{D} = (B = \mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{f}_B = f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N))$, we have the prior

$$\begin{bmatrix} f(\mathbf{x}_1) \\ \ldots \\ f(\mathbf{x}_N) \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, K_{BB;\theta})$$

at the observation points.

Let us select θ to maximize the likelihood $p(\mathbf{f}_B; \theta)$ of the observations under that prior:

$$\begin{aligned} \theta^* &= \arg\min_\theta -\log p(\mathbf{f}_B; \theta) \\ &= \arg\min_\theta -\frac{1}{2}\log\det K_{BB;\theta} - \frac{1}{2}\mathbf{f}_B^T K_{BB;\theta}^{-1}\mathbf{f}_B + c \end{aligned}$$


$\ell = 0.6018$
3
2
1
0
−1
−2
−3
0
1
2
3
4
5

Noisy observations

So far we assumed noise-free observations $\mathbf{f}_B = f(\mathbf{x}_B)$.

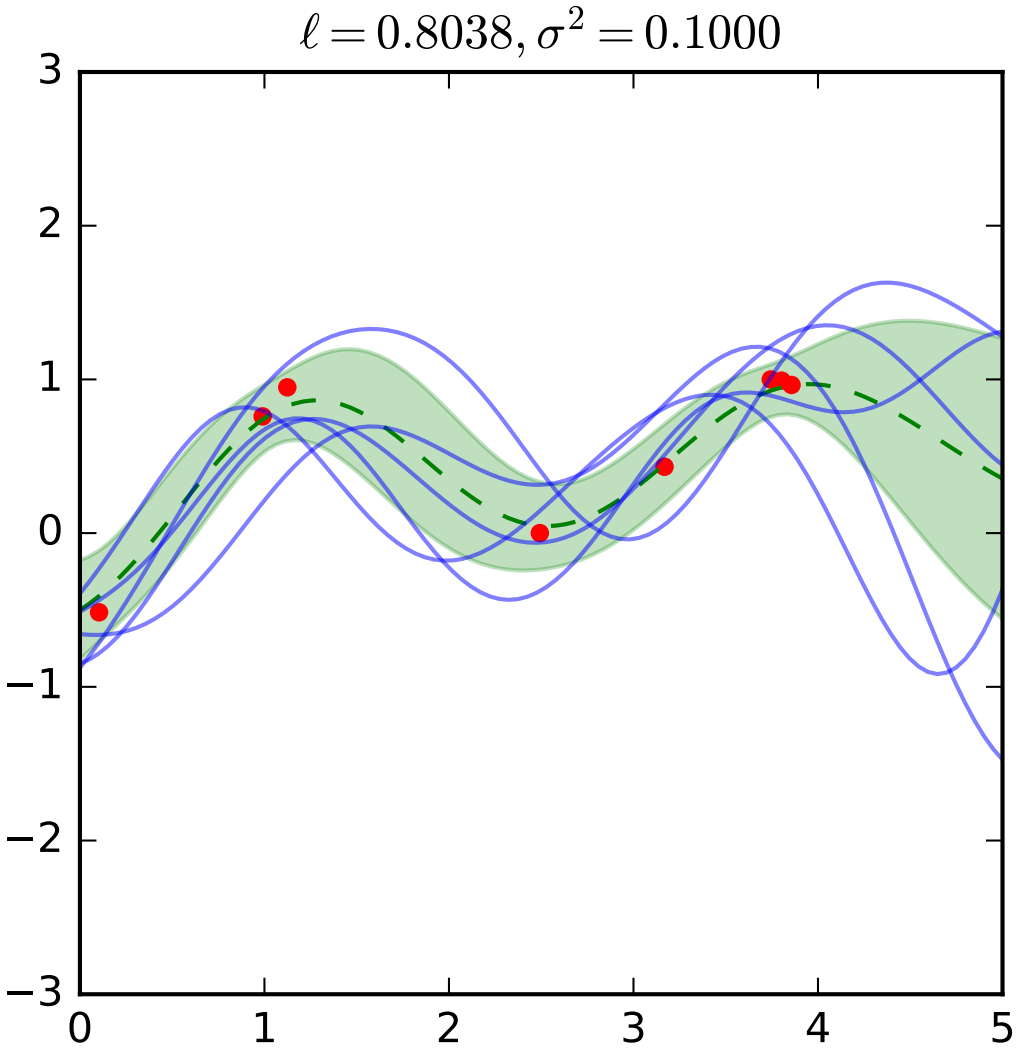
In more realistic situations, it is typical to instead consider noisy observations $\mathbf{y}_B = f(\mathbf{x}_B) + \epsilon$.

Assuming iid Gaussian noise ϵ with variance σ_N^2, the joint distribution of noisy observations of f at observation points and of the true values of f at target points is:

$$\begin{bmatrix} \mathbf{f}_A \\ \mathbf{y}_B \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_{AA} & K_{AB} \\ K_{BA} & K_{BB} + \sigma_N^2 I \end{bmatrix}\right),$$

which results in the posterior

$$\mathbf{f}_A \sim \mathcal{N}\left(K_{AB}[K_{BB} + \sigma_N^2 I]^{-1}\mathbf{f}_B, K_{AA} - K_{AB}[K_{BB} + \sigma_N^2 I]^{-1}K_{BA}\right).$$


$\ell = 0.8038, \sigma^2 = 0.1000$
3
2
1
0
−1
−2
−3
0
1
2
3
4
5

Summary

- Gaussian processes $=$ multivariate Gaussian in infinite dimension.
- Prior knowledge about f is captured in the distributional assumptions about the process prior. For example, assumptions on the smoothness of f is captured a priori via a parametric kernel function.
- Learning corresponds to posterior inference over the functional space conditioned on observations, and g is taken to be a random function distributed according to the predictive posterior distribution.
- Exact inference does not scale to many observations ($O(N^3)$).

Conditional neural processes

How do I read a paper?

1. Read title, abstract and conclusions (what is this about?).
2. Hop to the main figures.
3. If that looks interesting, I dive into the technical details:
    - Understand the method
    - Read through the experiments (do they support the claims?)

Abstract

Deep neural networks excel at function approximation, yet they are typically trained from scratch for each new function. On the other hand, Bayesian methods, such as Gaussian Processes (GPs), exploit prior knowledge to quickly infer the shape of a new function at test time. Yet GPs are computationally expensive, and it can be hard to design appropriate priors. In this paper we propose a family of neural models, Conditional Neural Processes (CNPs), that combine the benefits of both. CNPs are inspired by the flexibility of stochastic processes such as GPs, but are structured as neural networks and trained via gradient descent. CNPs make accurate predictions after observing only a handful of training data points, yet scale to complex functions and large datasets. We demonstrate the performance and versatility of the approach on a range of canonical machine learning tasks, including regression, classification and image completion.

5. Discussion

In this paper we have introduced Conditional Neural Processes, a model that is both flexible at test time and has the capacity to extract prior knowledge from training data. We have demonstrated its ability to perform a variety of tasks including regression, classification and image completion. We compared CNPs to Gaussian Processes on one hand, and deep learning methods on the other, and also discussed the relation to meta-learning and few-shot learning.

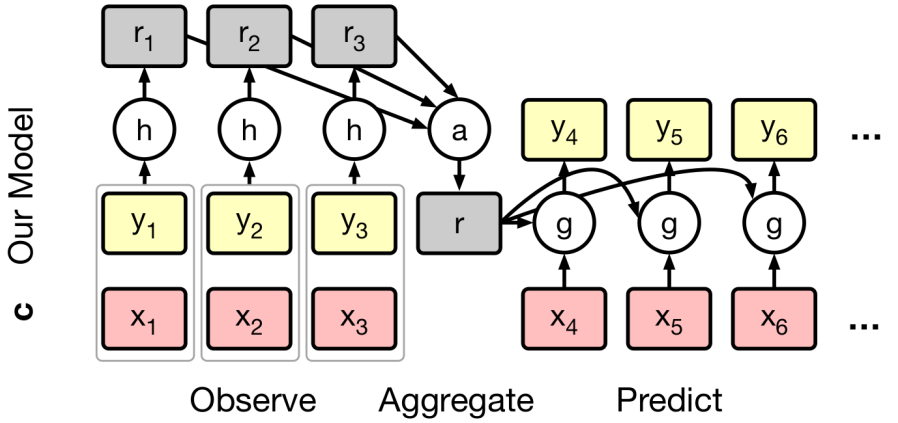
It is important to note that the specific CNP implementations described here are just simple proofs-of-concept and can be substantially extended, e.g. by including more elaborate architectures in line with modern deep learning advances.

To summarize, this work can be seen as a step towards learning high-level abstractions, one of the grand challenges of contemporary machine learning. Functions learned by most

conventional deep learning models are tied to a specific, constrained statistical context at any stage of training. A trained CNP is more general, in that it encapsulates the high-level statistics of a family of functions. As such it constitutes a high-level abstraction that can be reused for multiple tasks. In future work we will explore how far these models can help in tackling the many key machine learning problems that seem to hinge on abstraction, such as transfer learning, meta-learning, and data efficiency.

Conditional neural process

A Conditional Neural Process (CNP) is a conditional distribution over functions trained to model the empirical conditional distributions of functions $f \sim P$.



A Conditional Neural Process

- embeds each observation through a neural encoder h, that takes pairs of (x_i, y_i) context values and produces a representation $r_i = h((x_i, y_i))$ for each of the pairs;
- aggregates these embeddings r_i into a further embedding r of fixed dimension with a symmetric aggregator a,
- and produces through a neural decoder g distributions for $f(x_T)$ at the target points x_T, conditioned on the embedding r.

—

Credits: Marta Garnelo et al., arXiv:1807.01613.

Specifically, given a set of observations O, a CNP defines a conditional stochastic process Q_θ that defines distributions over $f(x)$ for targets $x \in T$. Permutation invariance w.r.t. T is enforced by assuming a factored structure such that

$$Q_\theta(f(T)|O,T) = \prod_{x_j \in T} Q_\theta(f(x_j)|O,x_j).$$

Then,

$$\begin{aligned} r_i &= h_\theta(x_i, y_i) && \forall (x_i, y_i) \in O \\ r &= (r_1 + ... + r_n)/n \\ \phi_j &= g_\theta(x_j, r) && \forall x_j \in T \end{aligned}$$

where ϕ_j are parameters for $Q_\theta(f(x_j)|O,x_j) = Q_\theta(f(x_j)|\phi_j)$.

- For regression tasks, ϕ_j parameterizes the mean and the variance $\phi_j = (\mu_j, \sigma_j^2)$ of a Gaussian distribution.
- For classification tasks, ϕ_j parameterizes the logits of the class probabilities p_c over the classes of a categorical distribution.

Exchangeability

A CNP is permutation invariant in observations and targets.

- For observations, permutation invariance is guaranteed through the symmetric aggregator.
- For targets, permutation invariance is guaranteed through the factored structure of Q_θ.

Consistency

If we marginalise out a part of the sequence the resulting marginal distribution is the same as that defined on the original sequence.

More precisely, if $1 \leq m \leq n$, then

$$Q_\theta(\{y_i\}_{i=0}^{m-1}|O) = \int Q_\theta(\{y_i\}_{i=0}^{n-1}|O) d_{y_{m:n-1}}.$$

For regression, this property is guaranteed by construction, as Q_θ is defined as a Gaussian.

As stated by the Kolmogorov Extension Theorem, exchangeability and consistency are sufficient conditions to a define a stochastic process.

Therefore, CNPs are stochastic processes, just like GPs.

Time complexity

A CNP is scalable, achieving running time complexity of $O(n + m)$, for making m predictions with n observations.

Training

Let $f \sim P, O = \{(x_i, y_i)\}_{i=0}^{n-1}, N \sim \mathcal{U}[0, ..., n-1]$ and $O_N = \{(x_i, y_i)\}_{i=0}^{N} \subset O$.

The training objective is then given by

$$\mathcal{L}(\theta) = -\mathbb{E}_{f \sim P}\left[\mathbb{E}_N\left[\log Q_\theta(\{y_i\}_{i=0}^{n-1} | O_N, \{x_i\}_{i=0}^{n-1}\right]\right].$$

That is, Q_θ is trained by asking it to predict O conditioned on a randomly chosen subset of O.

This approach shifts the burden of imposing prior knowledge from an analytic prior to empirical data. This has the advantage of liberating a practitioner from having to specify an analytic form for the prior, which is ultimately intended to summarize their empirical experience. Still, we emphasize that the Q_θ are not necessarily a consistent set of conditionals for all observation sets, and the training routine does not guarantee that. In summary,

Experiment 1: 1D regression

4.1. Function Regression

As a first experiment we test CNP on the classical 1D regression task that is used as a common baseline for GPs. We generate two different datasets that consist of functions generated from a GP with an exponential kernel. In the first dataset we use a kernel with fixed parameters, and in the second dataset the function switches at some random point on the real line between two functions each sampled with different kernel parameters.

At every training step we sample a curve from the GP, select a subset of n points (x_i, y_i) as observations, and a subset of points (x_t, y_t) as target points. Using the model described in Figure 1, the observed points are encoded using a three layer MLP encoder h with a 128 dimensional output representation r_i. The representations are aggregated into a single representation $r = \frac{1}{n}\sum r_i$ which is concatenated to x_t and passed to a decoder g consisting of a five layer MLP. The decoder outputs a Gaussian mean and variance for the target outputs $\hat{y}_t$. We train the model to maximize the log-likelihood of the target points using the Adam optimizer (Kingma & Ba, 2014).

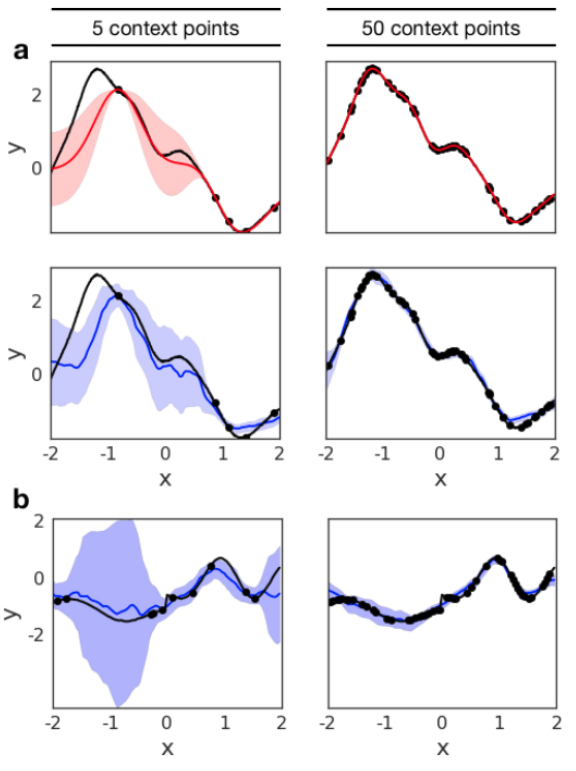


Figure 2. **1-D Regression**. Regression results on a 1-D curve (black line) using 5 (left column) and 50 (right column) context points (black dots). The first two rows show the predicted mean and variance for the regression of a single underlying kernel for GPs (red) and CNPs (blue). The bottom row shows the predictions of CNPs for a curve with switching kernel parameters.

Experiment 2: Image completion

4.2. Image Completion

We consider image completion as a regression task over functions in either $f : [0, 1]^2 \rightarrow [0, 1]$ for grayscale images, or $f : [0, 1]^2 \rightarrow [0, 1]^3$ for RGB images. The input x is the 2D pixel coordinates normalized to $[0, 1]^2$, and the output y is either the grayscale intensity or a vector of the RGB intensities of the corresponding pixel. For this completion task we use exactly the same model architecture as for 1D function regression (with the exception of making the last layer 3-dimensional for RGB).

We test CNP on two different data sets: the MNIST handwritten digit database (LeCun et al., 1998) and large-scale CelebFaces Attributes (CelebA) dataset (Liu et al., 2015). The model and training procedure are the same for both: at each step we select an image from the dataset and pick a subset of the pixels as observations. Conditioned on these, the model is trained to predict the values of all the pixels in the image (including the ones it has been conditioned on). Like in 1D regression, the model outputs a Gaussian mean and variance for each pixel and is optimized with respect to the log-likelihood of the ground-truth image. It is important to point out that we choose images as our dataset because they constitute a complex 2-D function that is easy to evaluate visually, not to compare to generative models benchmarks.

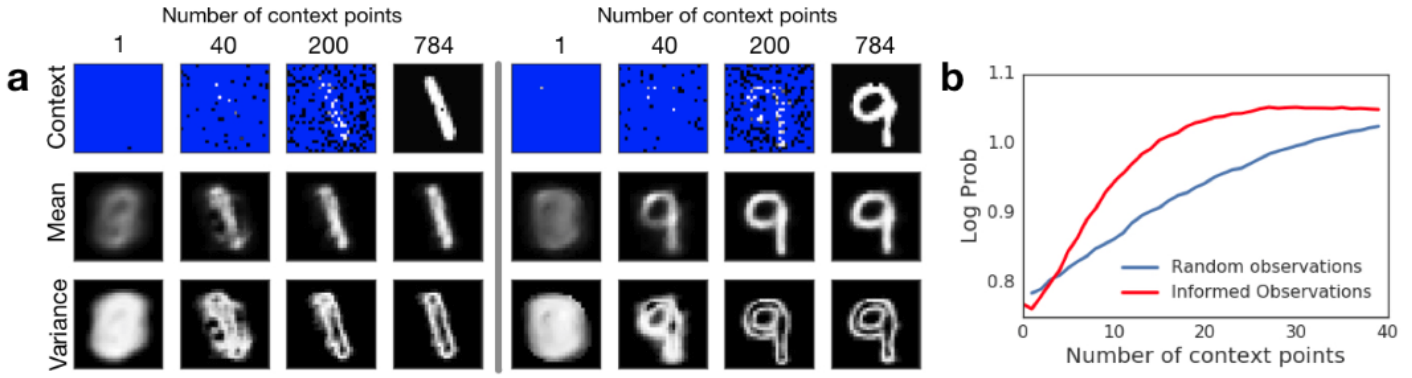


Figure 3. **Pixel-wise image regression on MNIST.** Left: Two examples of image regression with varying numbers of observations. We provide the model with 1, 40, 200 and 728 context points (top row) and query the entire image. The resulting mean (middle row) and variance (bottom row) at each pixel position is shown for each of the context images. Right: model accuracy with increasing number of observations that are either chosen at random (blue) or by selecting the pixel with the highest variance (red).

Since this implementation of CNP returns factored outputs, the best prediction it can produce given limited context information is to average over all possible predictions that agree with the context. An alternative to this is to add latent variables in the model such that they can be sampled conditioned on the context to produce predictions with high probability in the data distribution. We consider this model later in section 4.2.3.

!!!

Experiment 3: One-shot learning

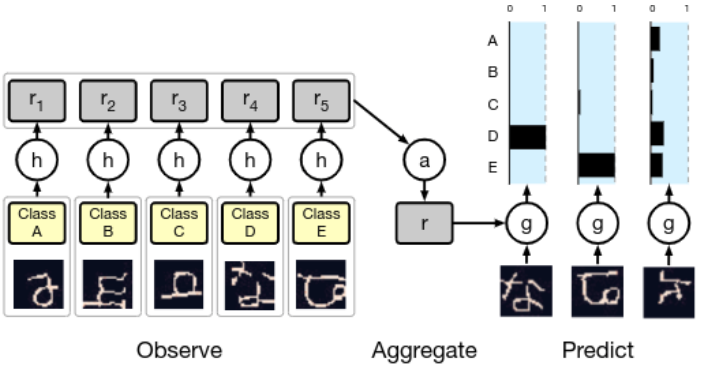


Figure 7. **One-shot Omniglot classification**. At test time the model is presented with a labelled example for each class, and outputs the classification probabilities of a new unlabelled example. The model uncertainty grows when the new example comes from an un-observed class.

	5-way Acc		20-way Acc		Runtime
	1-shot	5-shot	1-shot	5-shot	
MANN	82.8%	94.9%	-	-	$\mathcal{O}(nm)$
MN	**98.1%**	**98.9%**	**93.8%**	**98.5%**	$\mathcal{O}(nm)$
CNP	95.3%	98.5%	89.9%	96.8%	$\mathcal{O}(n+m)$

Table 2. **Classification results on Omniglot**. Results on the same task for MANN (Santoro et al., 2016), and matching networks (MN) (Vinyals et al., 2016) and CNP.

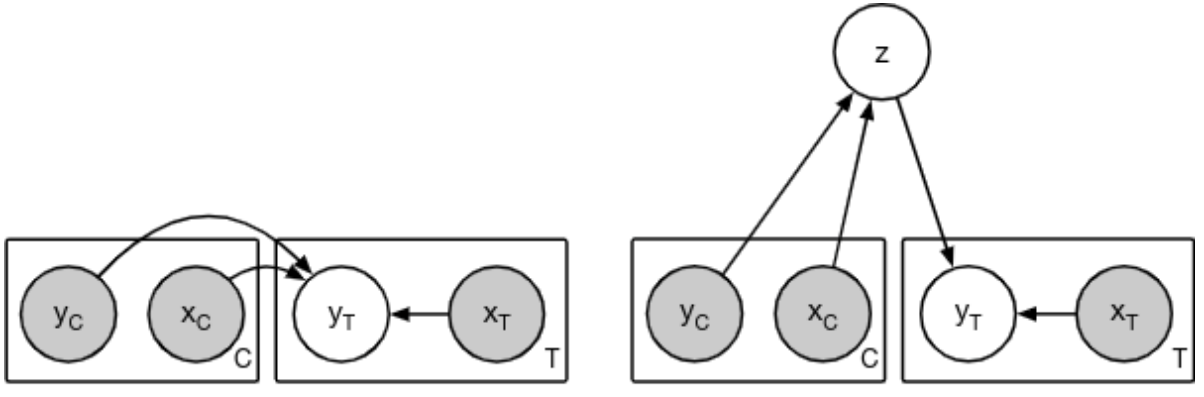
What do you think?

Do the experiments support the claims?

Follow-ups

Neural Processes (Garnelo et al, 2018)

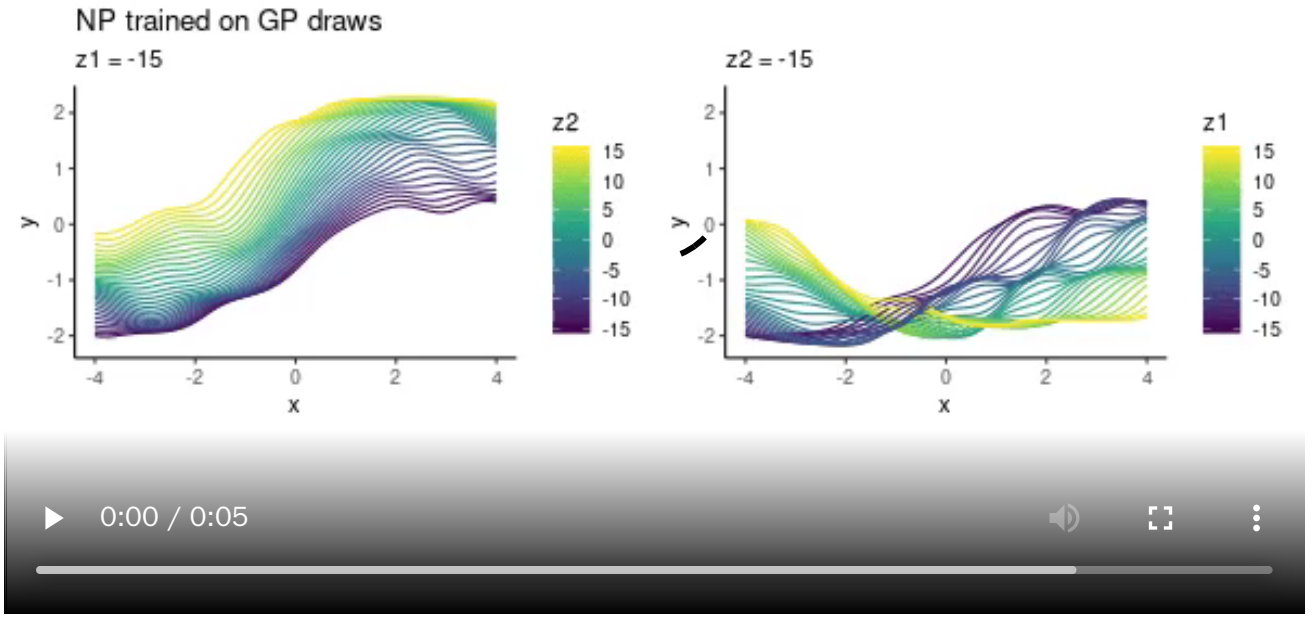
a large part of the motivation behind neural processes, but lack a latent variable that allows for global sampling (see Figure 2c for a diagram of the model). As a result, CNPs are unable to produce different function samples for the same context data, which can be important if modelling this uncertainty is desirable. It is worth mentioning that the orig-



(c) Conditional neural process (d) Neural process

—

Credits: Marta Garnelo et al., arXiv:1807.01622.



Credits: Kaspar Martens, 2018.

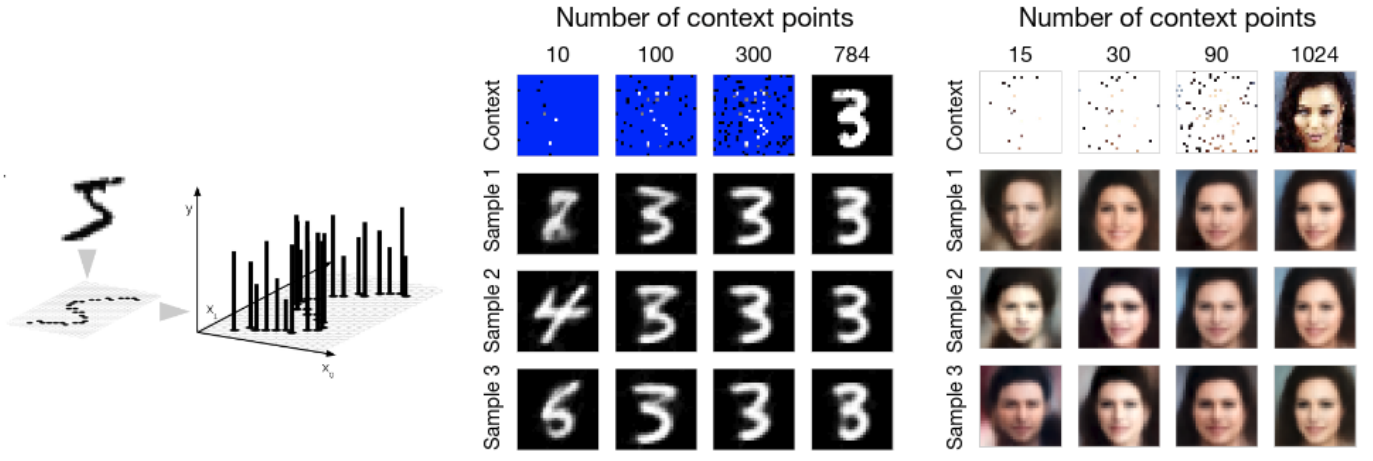


Figure 4. **Pixel-wise regression on MNIST and CelebA** The diagram on the left visualises how pixel-wise image completion can be framed as a 2-D regression task where f(pixel coordinates) = pixel brightness. The figures to the right of the diagram show the results on image completion for MNIST and CelebA. The images on the top correspond to the context points provided to the model. For better clarity the unobserved pixels have been coloured blue for the MNIST images and white for CelebA. Each of the rows corresponds to a different sample given the context points. As the number of context points increases the predicted pixels get closer to the underlying ones and the variance across samples decreases.

Attentive neural processes (Kim et al, 2019)

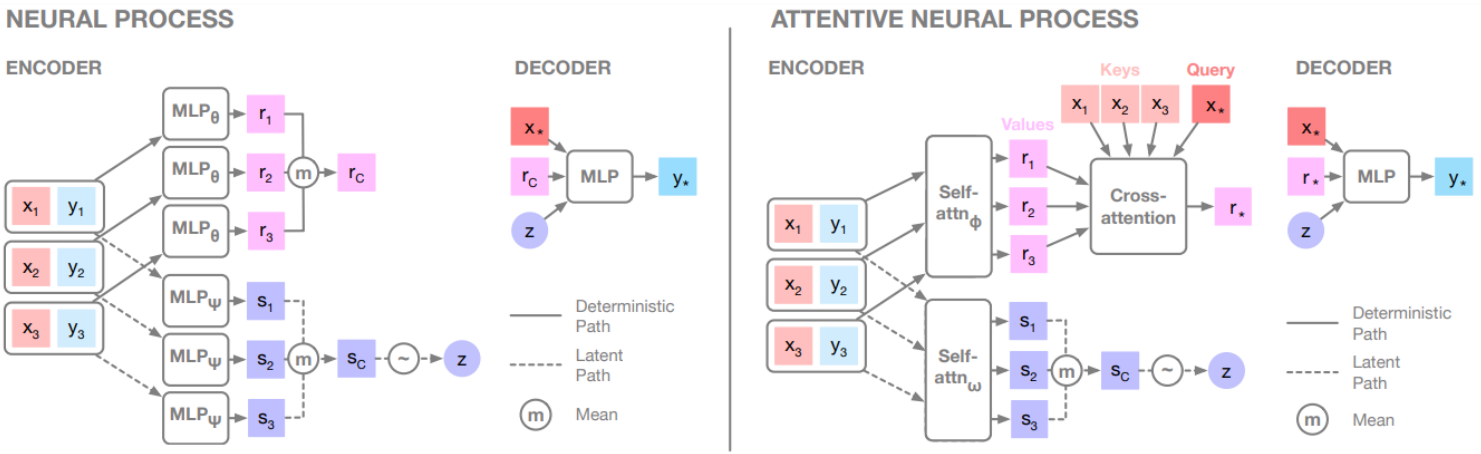


Figure 2: Model architecture for the NP (left) and Attentive NP (right)

Figure 2 describes how attention is incorporated into NP to give the Attentive NP (ANP). In summary, *self-attention* is applied to the context points to compute representations of each $(\boldsymbol{x}, \boldsymbol{y})$ pair, and the target input attends to these context representations (*cross-attention*) to predict the target output. In detail, the representation of each context pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i \in C}$ before the mean-aggregation step is computed by a self-attention mechanism, in both the deterministic and latent path. The intuition for the self-attention is to model interactions between the context points. For example, if many context points overlap, then the query need not attend to all of these points, but only give high weight to one or a few. The self-attention will help obtain richer representations of the context points that encode these types of relations between the context points. We model higher order interactions by simply stacking the self-attention, as is done in Vaswani et al. (2017).

—

Credits: Kim et al., arXiv:1901.05761.

Generative Query Networks (Eslami et al, 2018)

Finally, NPs and CNPs themselves can be seen as generalizations of recently published generative query networks (GQN) which apply a similar training procedure to predict new viewpoints in 3D scenes given some context observations (Eslami et al., 2018). Consistent GQN (CGQN) is an extension of GQN that focuses on generating consistent samples and is thus also closely related to NPs (Kumar et al., 2018).

Learning to See - Ali Eslami

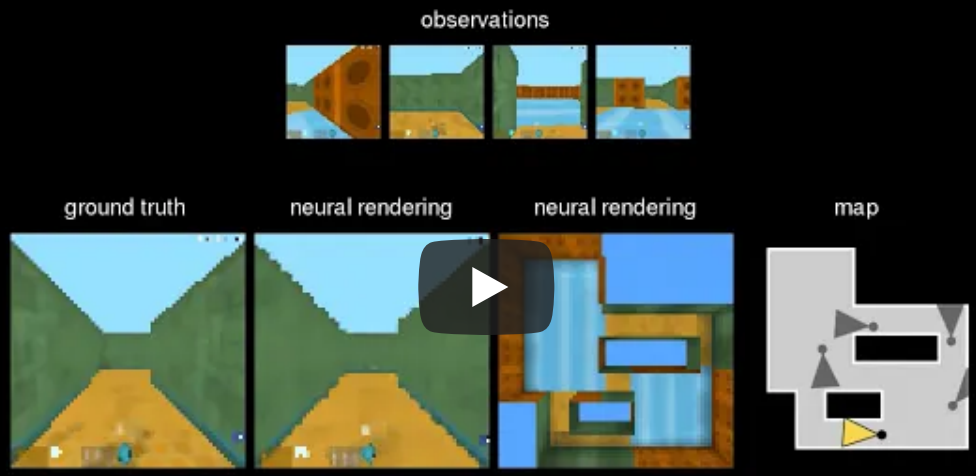
Later bekijk...

Delen

Neural Scene Representation and Rendering

Later bekijk...

Delen



The end.