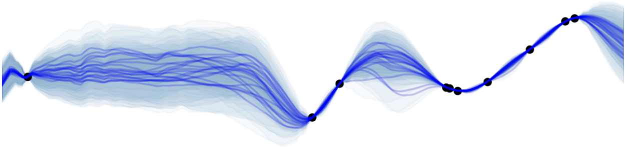
Neural Processes Reading Group

Andrew Foong, Sebastian Ober, Stratis Markou

2nd December 2020

UNIVERSITY OF CAMBRIDGE

Introduction



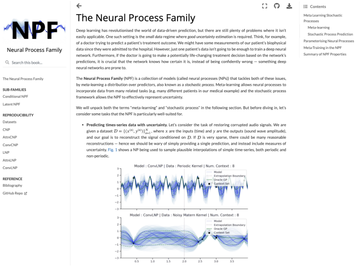
What are Neural Processes (NPs)?

- They are:
  1. a **meta-learning** framework.
  2. modelling **stochastic processes**.
  3. using **neural networks**.
- The hope: benefits of GPs and deep learning together.
- Introduced in Garnelo et al. [2018a].
- Many variants/extensions since proposed.

Outline

This reading group in conceptual order.

1. Introduction: meta-learning stochastic processes. (Andrew)
2. Conditional Neural Processes. (Sebastian)
3. (Latent) Neural Processes. (Stratis)

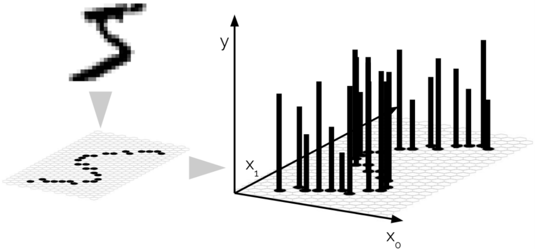


- Layout follows `https://yanndubs.github.io/Neural-Process-Family`, made by Yann Dubois, Jonathan Gordon and Andrew Foong.
- Code for many NPs.

Problem Set-up

Task: prediction **under uncertainty** in the **small-data regime**

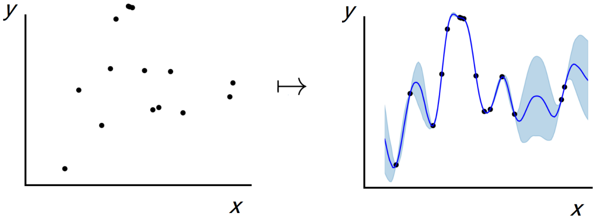
- Examples:
  - Predicting time-series.
  - Image completion. View images as functions from the 2D plane $\mathbb{R}^2 \to \mathbb{R}$.



- Could be difficult to design a GP kernel for this kind of data — can we learn this structure?

Meta-Learning

Meta-learning is **learning to learn**.



context set

prediction with uncertainty

- View learning as a **map** from data sets to predictives.
- Given observed *context set* $D_{\mathcal{C}} = \{(x^{(c)}, y^{(c)})\}_{c=1}^{C}$.
- Make predictions at a *target set* $\mathrm{x}_{\mathcal{T}} = \{x^{(t)}\}_{t=1}^{T}$.
- NPs use neural networks to **directly parameterise** map $D_{\mathcal{C}} \mapsto p(\mathrm{y}_{\mathcal{T}} | \mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}})$.

General Challenges in Designing NPs

Two challenges:

1. Standard NNs eat fixed-length *vectors*. NPs eat *entire datasets*:
   - Datasets can be of **varied sizes**.
   - The map $D_{\mathcal{C}} \mapsto p(\mathrm{y}_{\mathcal{T}}|\mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}})$ should be **invariant to permutations** of $D_{\mathcal{C}}$.
2. For *any* target set $\mathrm{x}_{\mathcal{T}}$, the NP must return $p(\mathrm{y}_{\mathcal{T}}|\mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}})$.
   - Are these predictives be **consistent** for varying $\mathrm{x}_{\mathcal{T}}$?
   - I.e. do NPs define a valid **stochastic process**?

We'll discuss **challenge 1** first.

First Challenge: Machine Learning on Sets

Deep learning on sets is well-studied, e.g. Zaheer et al. [2017].

- Key result is a **representation theorem**:

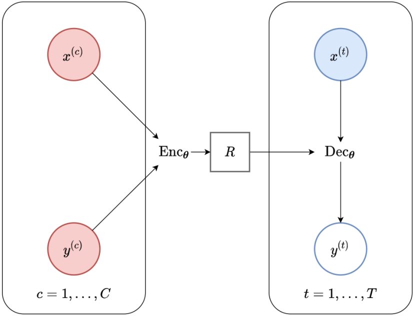
Theorem 1 (Zaheer et al. [2017], Wagstaff et al. [2019]).
Let $M \in \mathbb{N}$, and let $f : [0,1]^M \to \mathbb{R}$ be a continuous, permutation-invariant *function. Then there exist continuous maps $\phi : [0,1] \to \mathbb{R}^M$ and $\rho : \mathbb{R}^M \to \mathbb{R}$ such that for all $X \in [0,1]^M$,*

$$f(X) = \rho\left(\sum_{i=1}^{M} \phi(X_m)\right)$$

- Implement ϕ and ρ as NNs.
- Think of X as a data set and $\mathbb{R}^M$ as a representation space.
- Each data point X_m is mapped to $\phi(X_m)$, then summed.
- Known as a **deep sets** or **sum decomposition**.

Computational Graph

Common NP computational graph:



- The **encoder** Enc_θ maps each datapoint to a representation.
- Representations are **aggregated** to form R.
- The **decoder** Dec_θ maps R along with target input $x^{(t)}$ to predictions.

Many concrete instantiations during Seb and Stratis' talks.

Second Challenge: Consistency of Predictives

NPs map $D_{\mathcal{C}} \mapsto p(\mathrm{y}_{\mathcal{T}}|\mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}})$ for *any* $\mathrm{x}_{\mathcal{T}}$.

- What could go wrong with an arbitrary mapping?
- Consider 1D regression, with $\mathrm{x}_{\mathcal{T}} = \{1\}, \mathrm{x}'_{\mathcal{T}} = \{1, 2\}$ for a fixed $D_{\mathcal{C}}$.
- We obtain $p(y_1|\{1\}, D_{\mathcal{C}})$ and $p(y_1, y_2|\{1, 2\}, D_{\mathcal{C}})$.
- Must satisfy a **consistency condition**:

$$\int p(y_1, y_2|\{1, 2\}, D_{\mathcal{C}})\,\mathrm{d}y_2 = p(y_1|\{1\}, D_{\mathcal{C}}).$$

- If not, predictions change arbitrarily depending on which points are in the target set!

Kolmogorov Extension Theorem guarantees that:

- If predictives consistent under **marginalisation** and **permutation**,
- they are indeed marginals of a stochastic process (random function).

The Two NP Sub-families

Two main flavours:

1. **Conditional** Neural Process Family assumes the predictive is **factorised** conditioned on the representation $R(D_{\mathcal{C}})$.

$$p(\mathrm{y}_{\mathcal{T}}|\mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}}) = \prod_{t=1}^{T} p(y_t|x^{(t)}, R(D_{\mathcal{C}})).$$

2. **(Latent)** Neural Process Family uses the representation $R(D_{\mathcal{C}})$ to define a **latent variable** $\mathrm{z} \sim p(\mathrm{z}|R)$.
   - The predictive is **factorised conditioned on** z:

$$p(\mathrm{y}_{\mathcal{T}}|\mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}}) = \int \prod_{t=1}^{T} p(y_t|x^{(t)}, \mathrm{z}) p(\mathrm{z}|R(D_{\mathcal{C}}))\,\mathrm{dz}.$$

   - Allows for **dependencies**, unlike Conditional NPs!

Episodic Training

How to train NPs?

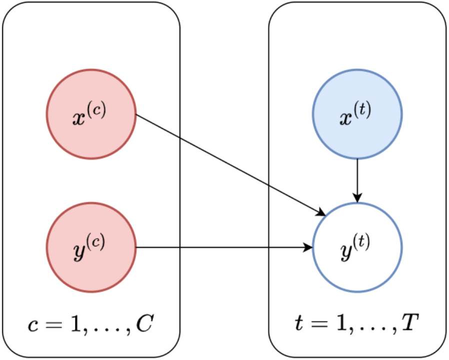
1. Sample dataset D from **a large collection** $\{D_i\}_{i=1}^{N_{\text{tasks}}}$.
2. Randomly split into context and target sets: $D = D_{\mathcal{C}} \cup D_{\mathcal{T}}$.
3. Pass $D_{\mathcal{C}}$ through the NP to obtain the predictive $p(\mathrm{y}_{\mathcal{T}} | \mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}})$.
4. Compute objective $\mathcal{L}$ which measures predictive performance on the target set.
5. Compute $\nabla_\theta \mathcal{L}$ to optimise parameters of the NP.

That's it! Now we'll take a closer look at the **Conditional** Neural Process family.

The Conditional Neural Processes Family

As we mentioned earlier, members of the Conditional Neural Process Family (CNPF) assume the following factorisation for the predictive:

$$p(\mathrm{y}_{\mathcal{T}} | \mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}}) = \prod_{t=1}^{T} p(y^{(t)} | x^{(t)}, R(D_{\mathcal{C}})).$$



The CNPF factorisation implies consistency

We briefly show how this factorisation implies the consistency required for stochastic processes:

1. **Permutation:** Let π be any permutation of $\{1, \ldots, T\}$. Then

$$\prod_{t=1}^{T} p(y^{(t)}|x^{(t)}, R(D_{\mathcal{C}})) = \prod_{t=1}^{T} p(y^{\pi(t)}|x^{\pi(t)}, R(D_{\mathcal{C}}))$$
$$= p(y^{\pi(1)}, \ldots, y^{\pi(T)}|x^{\pi(1)}, \ldots, x^{\pi(T)}, D_{\mathcal{C}})$$

2. **Marginalisation:** Let $A \subset \{1, \ldots, T\}$ and A^c be its complement. Then

$$\int p(\mathrm{y}_A, \mathrm{y}_{A^c}|\mathrm{x}_A, \mathrm{x}_{A^c}, D_{\mathcal{C}}) d\mathrm{y}_{A^c} = \int p(\mathrm{y}_A|\mathrm{x}_A, D_{\mathcal{C}}) p(\mathrm{y}_{A^c}|\mathrm{x}_{A^c}, D_{\mathcal{C}}) d\mathrm{y}_{A^c}$$
$$= p(\mathrm{y}_A|\mathrm{x}_A, D_{\mathcal{C}})$$

Therefore, CNPF members do indeed satisfy both conditions necessary to be stochastic processes!

Decoders and the Maximum Likelihood Objective

We have already discussed the encoder in neural processes, which encodes a context set $D_{\mathcal{C}}$ into a global representation $R(D_{\mathcal{C}})$. We now discuss the **decoder** in the CNPF:

1. The decoder Dec_θ takes the representation $R(D_{\mathcal{C}})$ and a target input $x^{(t)}$, and maps them to parameters of the predictive distribution
2. For all the CNP types we consider, we assume a Gaussian predictive:

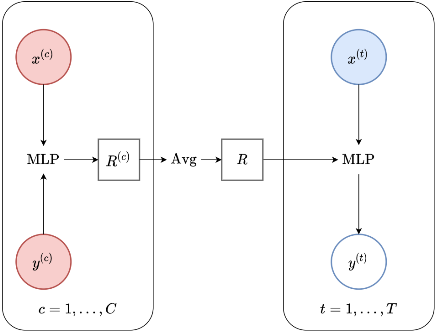
$$\begin{aligned} p(y^{(t)}|x^{(t)}, R(D_{\mathcal{C}})) &= \mathcal{N}(y^{(t)}; \mu_t, \sigma_t^2) \\ (\mu_t, \sigma_t^2) &= \mathrm{Dec}_\theta(R(D_{\mathcal{C}}), x^{(t)}) \end{aligned}$$

In CNPs, there are no random variables we need to perform inference over - we can optimize using **maximum likelihood**!

$$\mathcal{L} = \log p(\mathrm{y}_{\mathcal{T}}|\mathrm{x}_{\mathcal{T}}, D_{\mathcal{C}})$$

Conditional Neural Processes

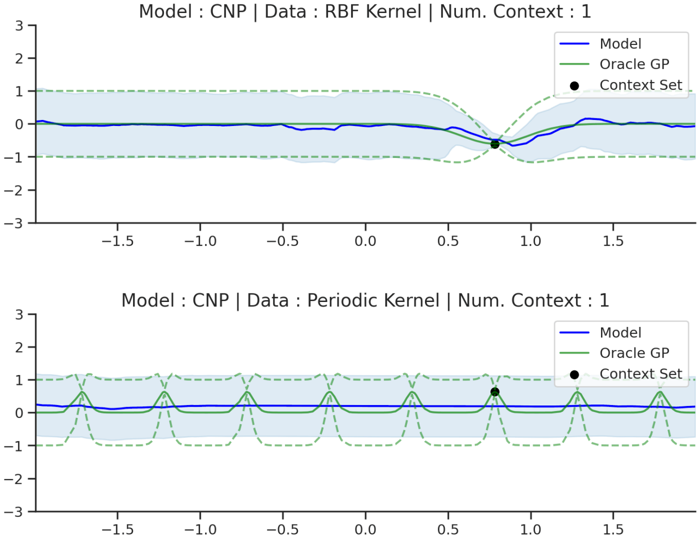
We start with the simplest member, known just as the CNP [Garnelo et al., 2018a].



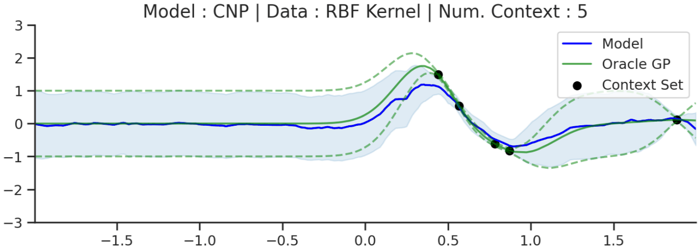
- **Encoder:** $R(D_{\mathcal{C}}) = \mathrm{Enc}_\theta(D_{\mathcal{C}}) = \frac{1}{C}\sum_{c=1}^{C} \mathrm{MLP}([x^{(c)}, y^{(c)}])$
- **Decoder:** $(\mu_t, \sigma_t^2) = \mathrm{Dec}_\theta(R(D_{\mathcal{C}}), x^{(t)}) = \mathrm{MLP}([R(D_{\mathcal{C}}), x^{(t)}])$

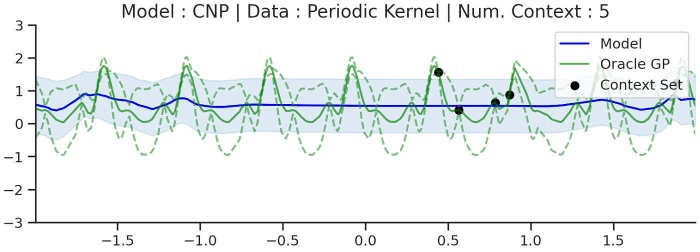
For wide enough MLPs, this should be able to predict any mean μ_t and variance σ_t^2!

Conditional Neural Processes (cont'd)

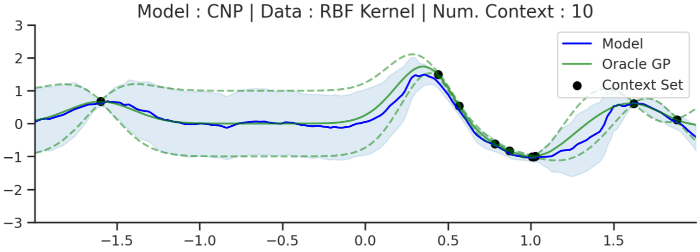


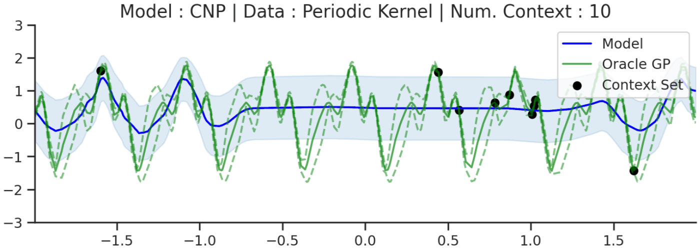
Conditional Neural Processes (cont'd)



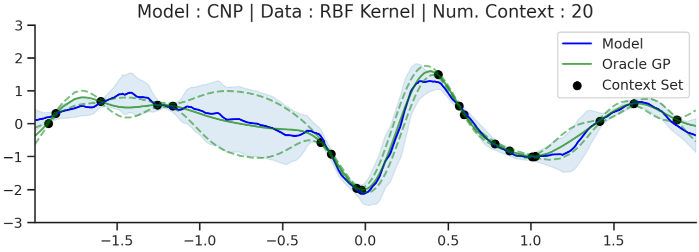


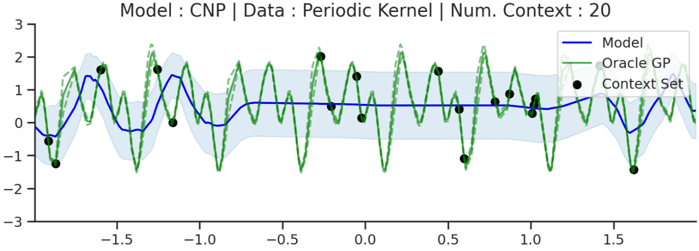
Conditional Neural Processes (cont'd)



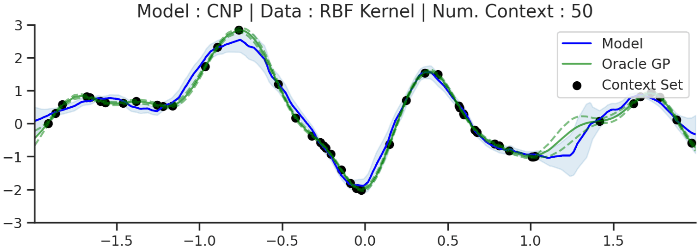


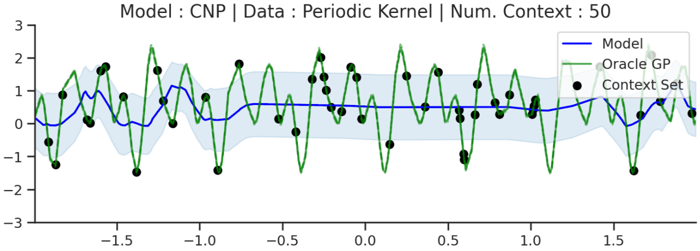
Conditional Neural Processes (cont'd)



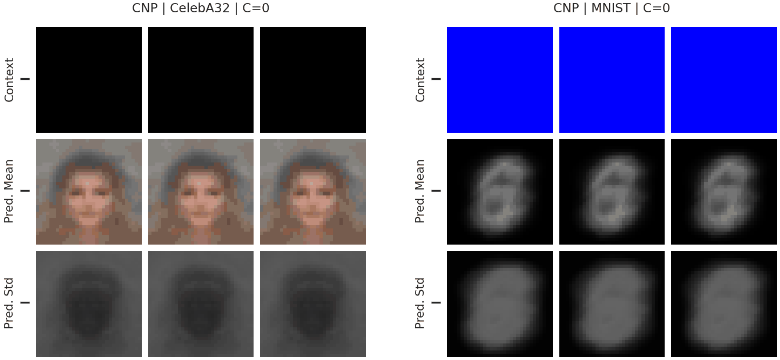


Conditional Neural Processes (cont'd)

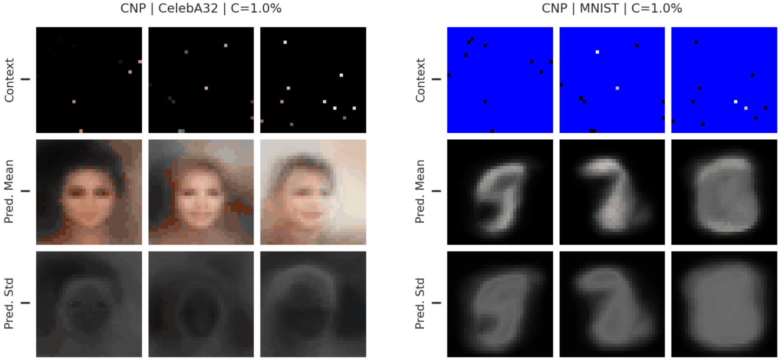




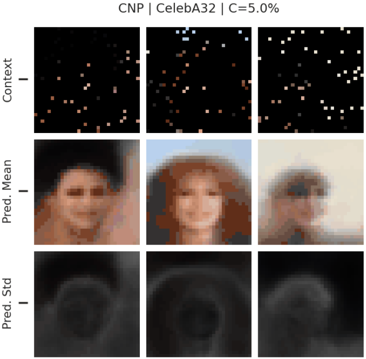
Conditional Neural Processes (cont'd)

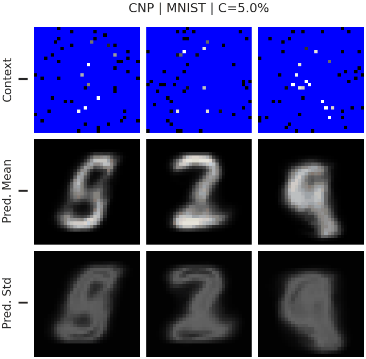


Conditional Neural Processes (cont'd)

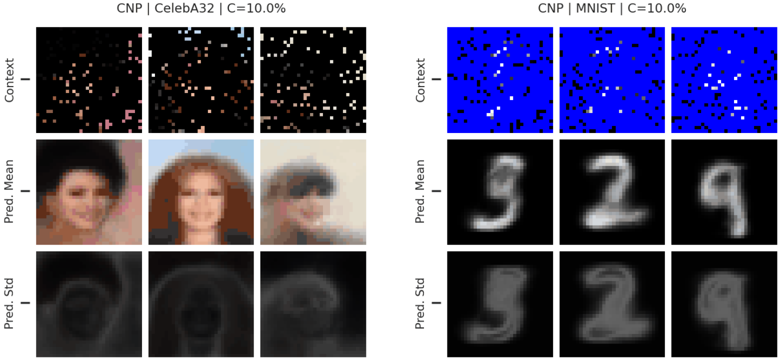


Conditional Neural Processes (cont'd)

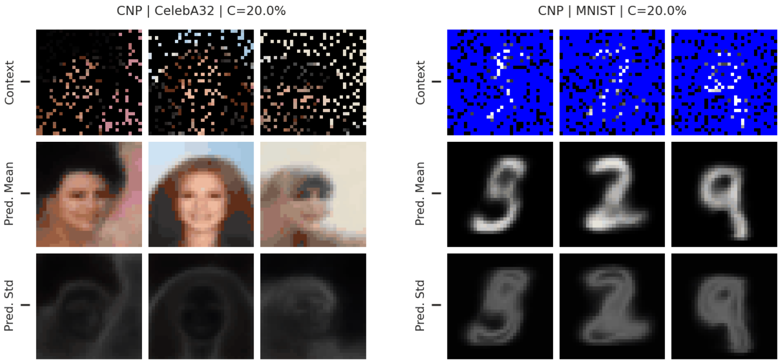




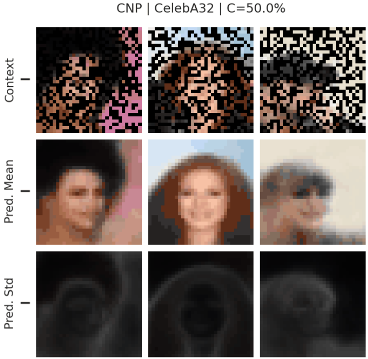
Conditional Neural Processes (cont'd)

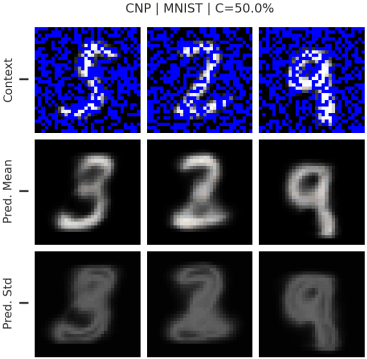


Conditional Neural Processes (cont'd)

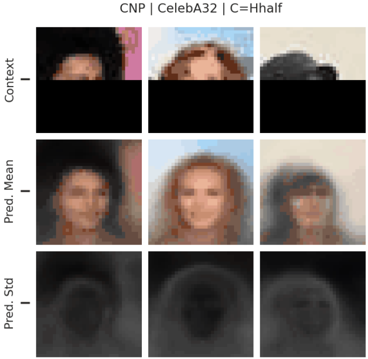


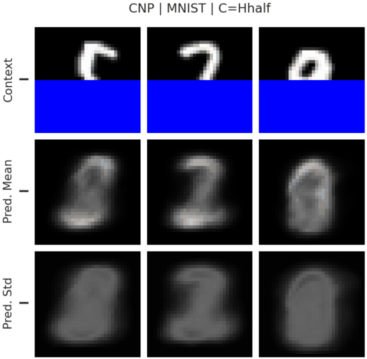
Conditional Neural Processes (cont'd)





Conditional Neural Processes (cont'd)





Attentive CNPs

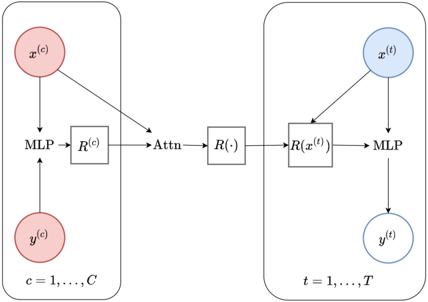
We've seen that even though the results are quite impressive, the standard CNP has a tendency to underfit.

1. This may be due to the fact that the representation $R(D_\mathcal{C})$ is the same for each target input, $x^{(t)}$
2. Instead, we may want to focus on context points closer to the target input, and give less weight to those further away
3. A great way of achieving this is **attention**: learn a weighting $w_\theta(x^{(c)}, x^{(t)})$ for each context-target point pair to be used in the encoding

$$R(D_\mathcal{C}, \cdot) = \mathrm{Enc}_\theta(D_\mathcal{C}) = \sum_{c=1}^{C} w_\theta(x^{(c)}, \cdot)\mathrm{MLP}([x^{(c)}, y^{(c)}])$$

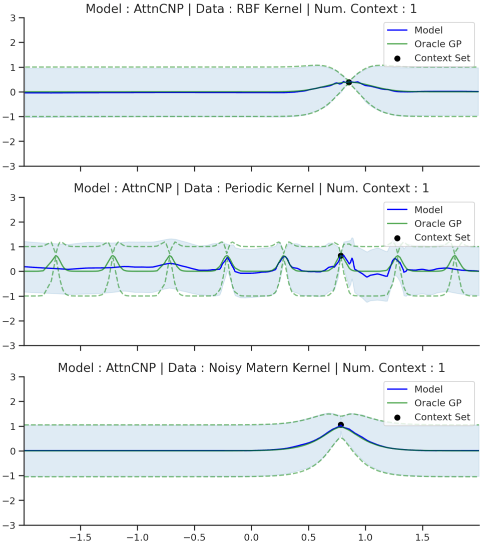
Attentive CNPs (cont'd)

This motivates the **Attentive CNP (AttnCNP)** [Kim et al., 2019].

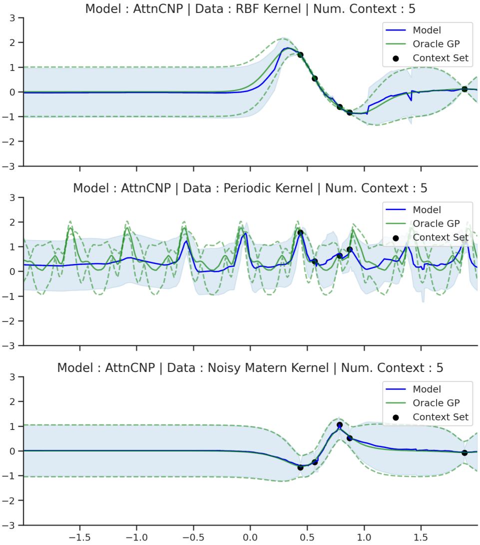


- **Encoder:** $R(D_{\mathcal{C}}, \cdot) = \mathrm{Enc}_\theta(D_{\mathcal{C}}) = \sum_{c=1}^{C} w_\theta(x^{(c)}, \cdot)\mathrm{MLP}([x^{(c)}, y^{(c)}])$
- **Decoder:** $(\mu_t, \sigma_t^2) = \mathrm{Dec}_\theta(R(D_{\mathcal{C}}), x^{(t)}) = \mathrm{MLP}([R(D_{\mathcal{C}}, x^{(t)}), x^{(t)}])$

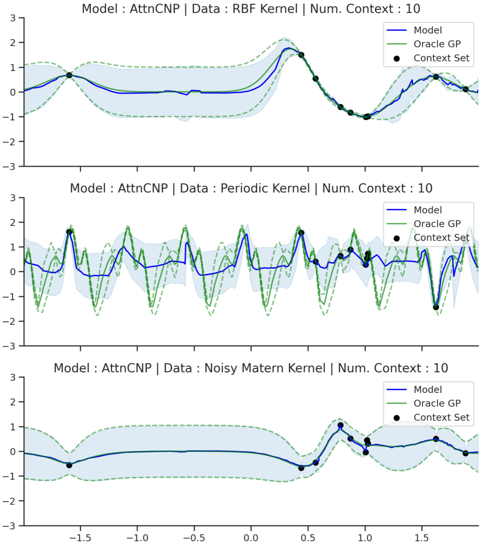
Attentive CNPs (cont'd)



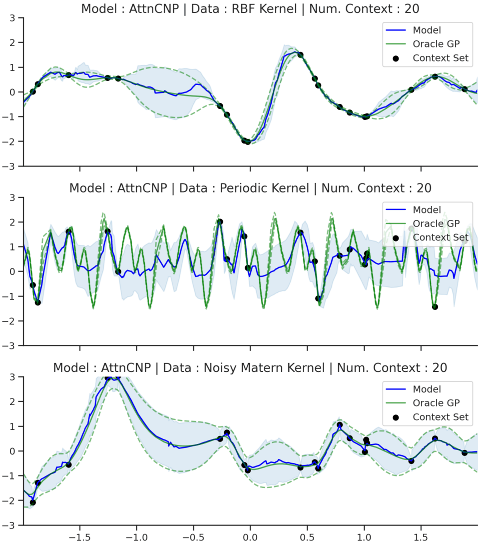
Attentive CNPs (cont'd)



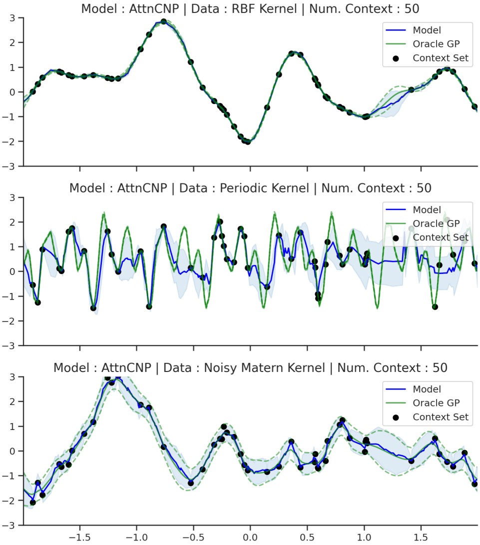
Attentive CNPs (cont'd)



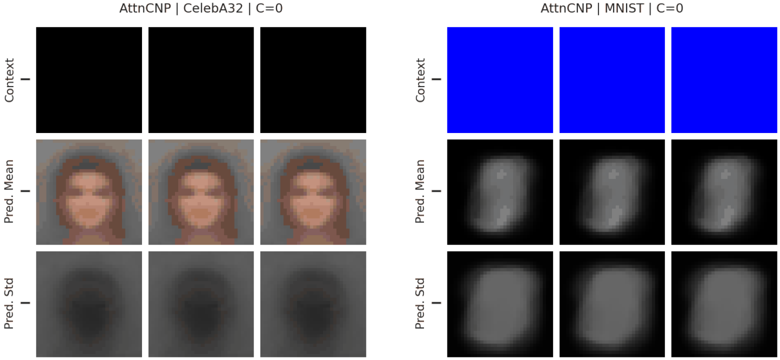
Attentive CNPs (cont'd)



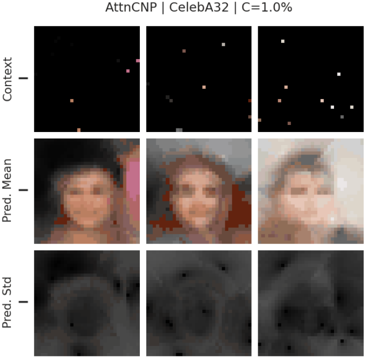
Attentive CNPs (cont'd)

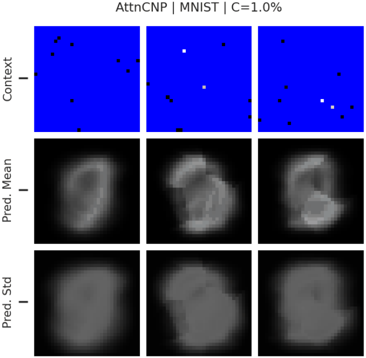


Attentive CNPs (cont'd)

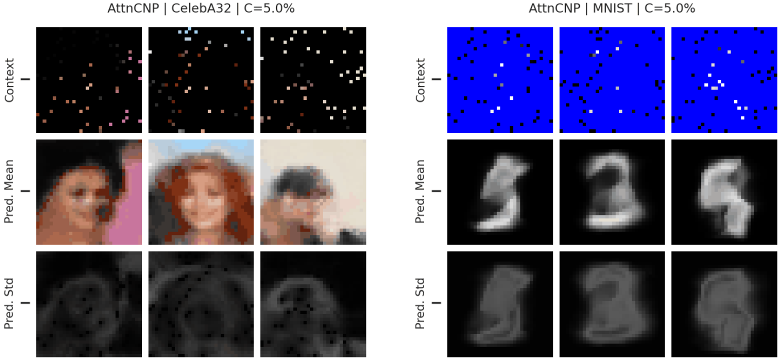


Attentive CNPs (cont'd)

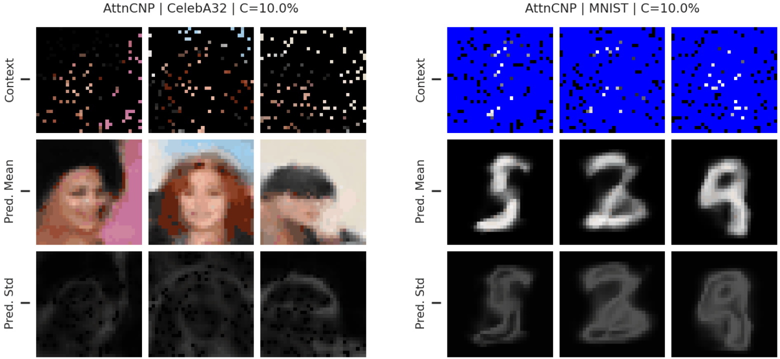




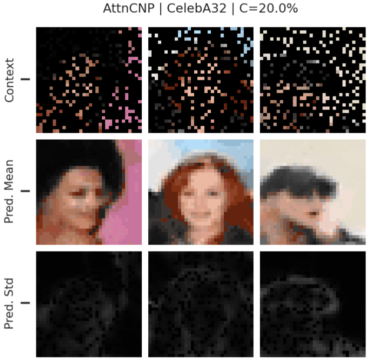
Attentive CNPs (cont'd)

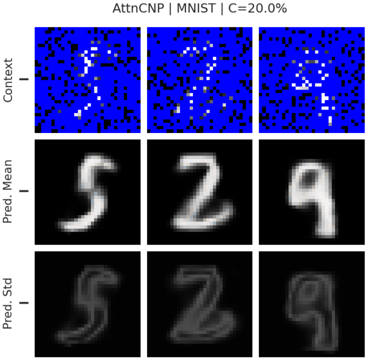


Attentive CNPs (cont'd)

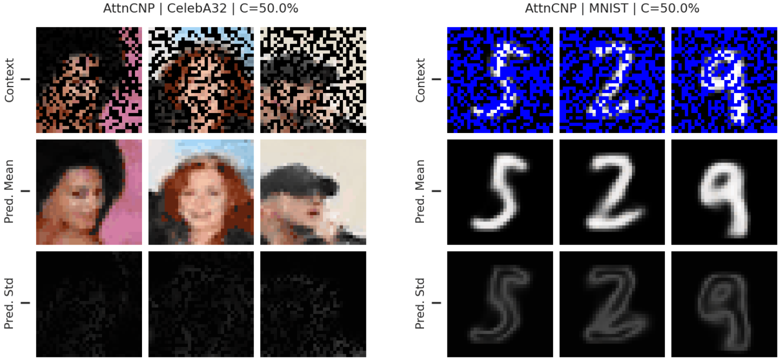


Attentive CNPs (cont'd)

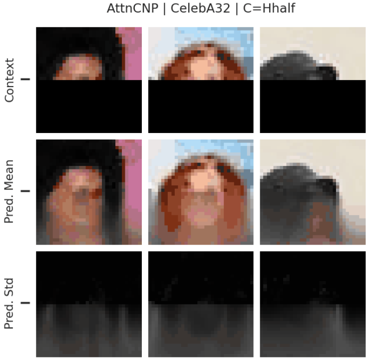


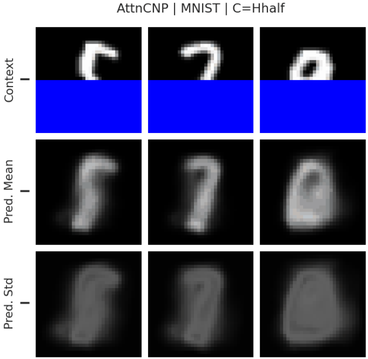


Attentive CNPs (cont'd)



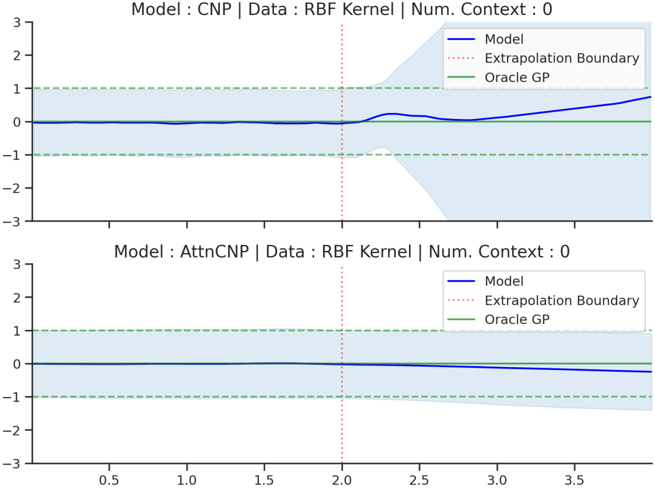
Attentive CNPs (cont'd)



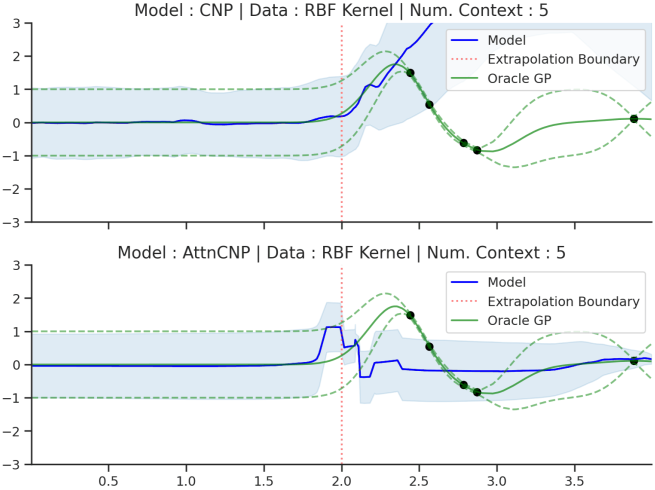


Generalisation and Extrapolation

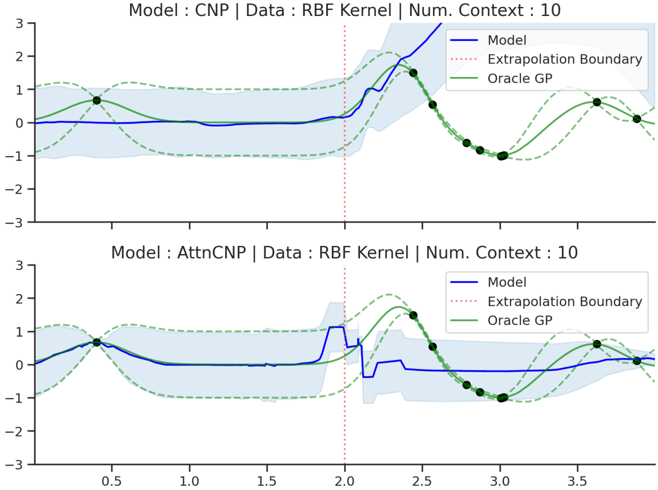
We now look at the ability of CNPs to generalise outside of the region in which they were trained.



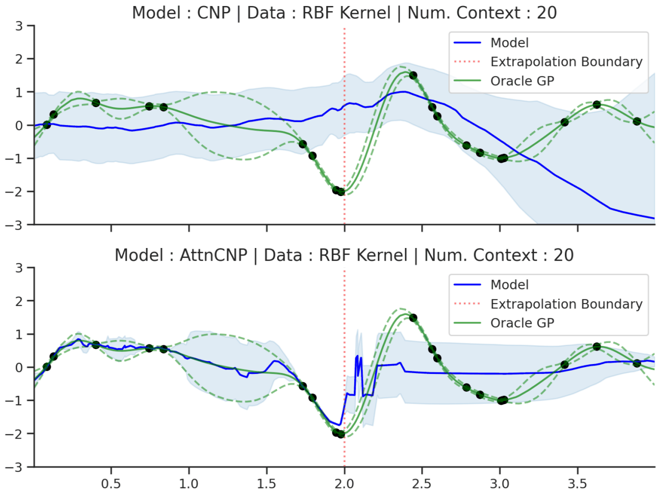
Generalisation and Extrapolation (cont'd)



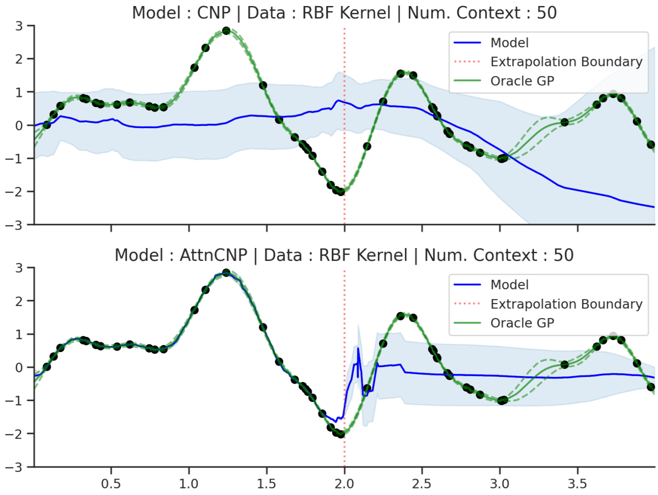
Generalisation and Extrapolation (cont'd)



Generalisation and Extrapolation (cont'd)

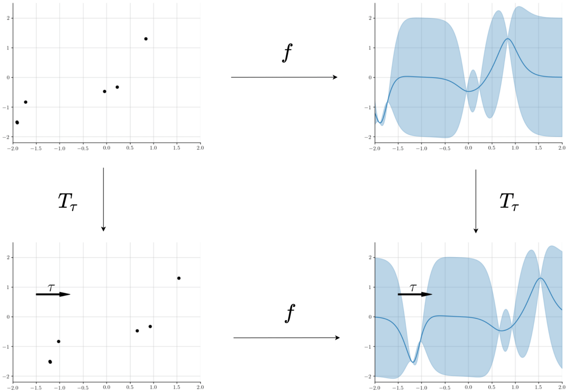


Generalisation and Extrapolation (cont'd)



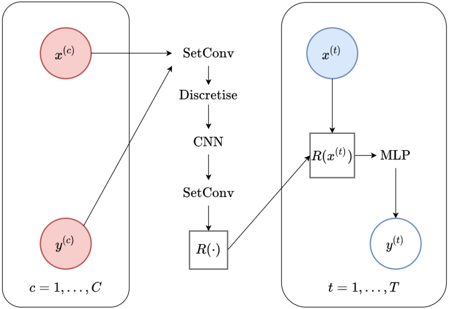
Translation Equivariance

We want predictions to depend on *relative* positions of context points, not *absolute* positions. This can be achieved with **translation equivariance**.



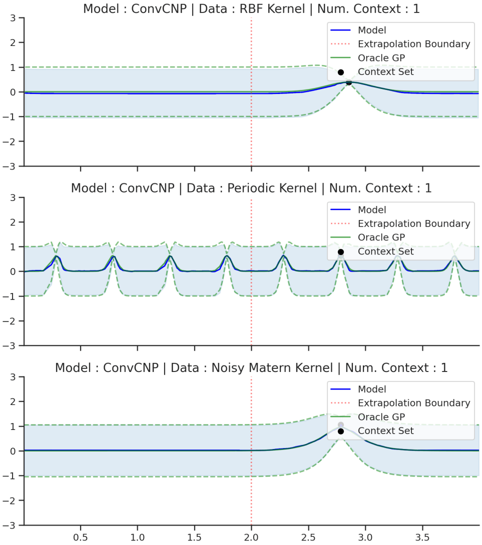
Convolutional CNP (ConvCNP)

One model that encodes translation equivariance is the **convolutional CNP** (ConvCNP) [Gordon et al., 2019], using a special set operation called SetConv and a CNN, motivated by a ConvDeepSets result.

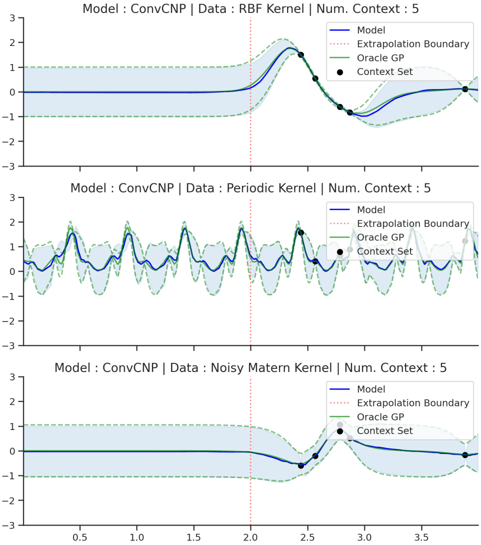


- **Encoder:** $R(D_{\mathcal{C}}, \cdot) = \mathrm{Enc}_\theta(D_{\mathcal{C}}) = \mathrm{SetConv}(\mathrm{CNN}(\{\mathrm{SetConv}(D_{\mathcal{C}})(x^{(u)})\}_{u=1}^{U}))(\cdot)$
- **Decoder:** $(\mu_t, \sigma_t^2) = \mathrm{Dec}_\theta(R(D_{\mathcal{C}}), x^{(t)}) = \mathrm{MLP}(R(D_{\mathcal{C}}, x^{(t)}))$

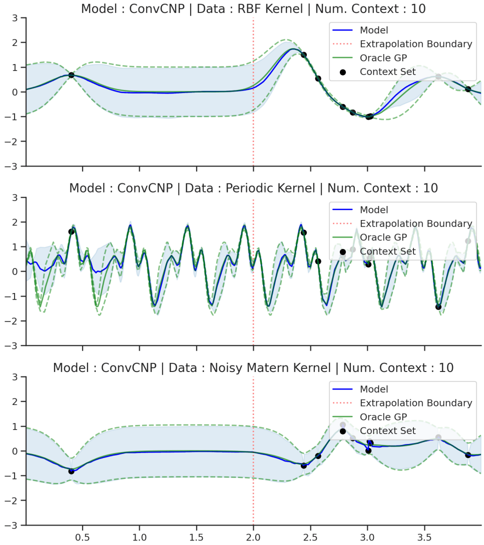
ConvCNP (cont'd)



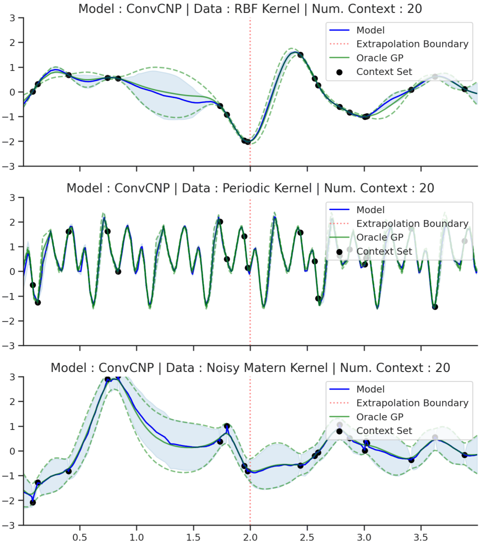
ConvCNP (cont'd)



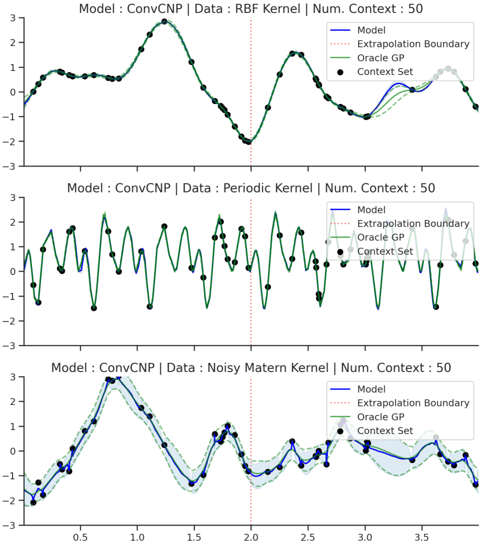
ConvCNP (cont'd)



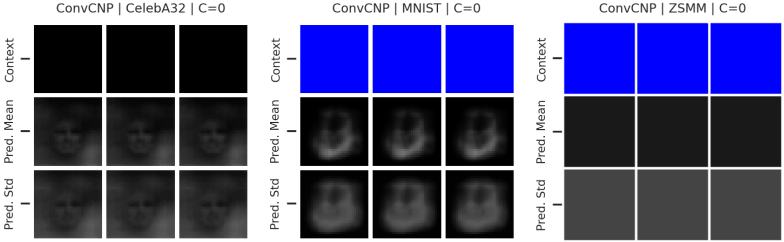
ConvCNP (cont'd)



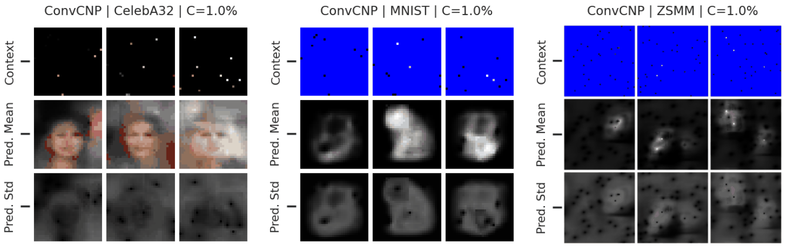
ConvCNP (cont'd)



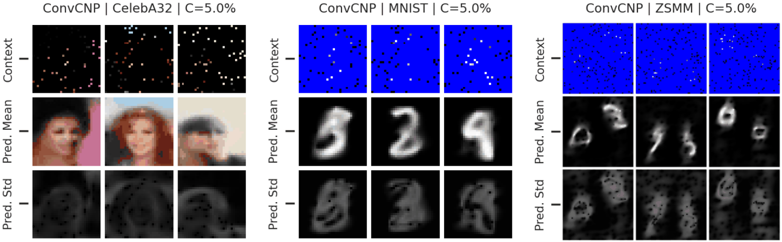
ConvCNP (cont'd)



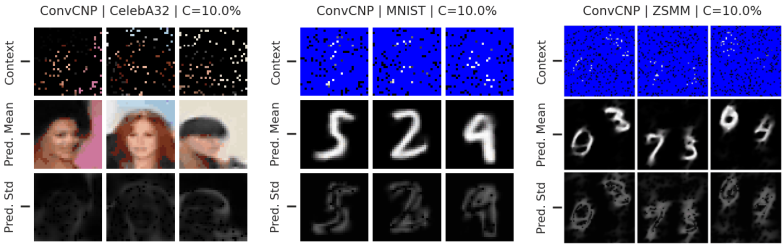
ConvCNP (cont'd)



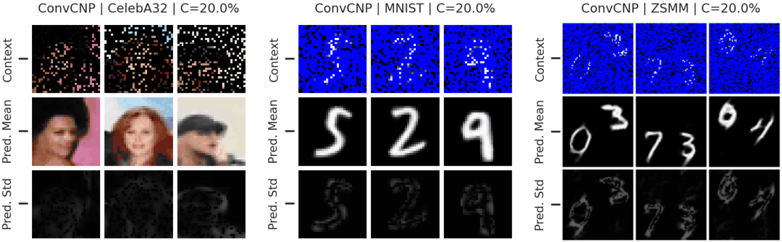
ConvCNP (cont'd)



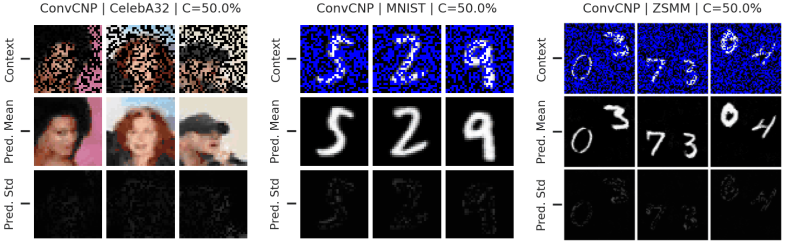
ConvCNP (cont'd)



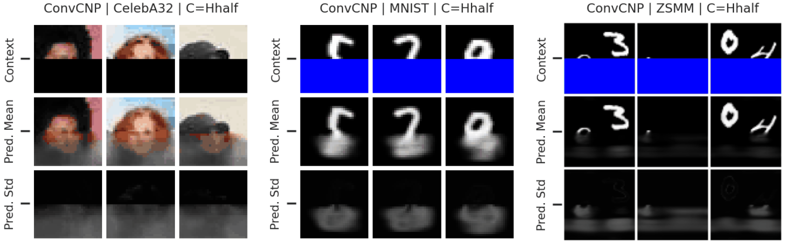
ConvCNP (cont'd)



ConvCNP (cont'd)

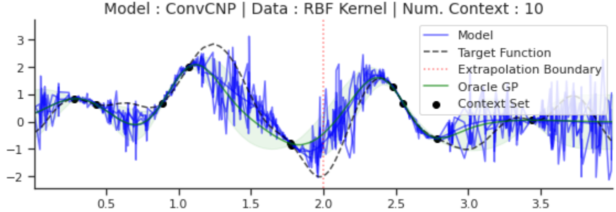


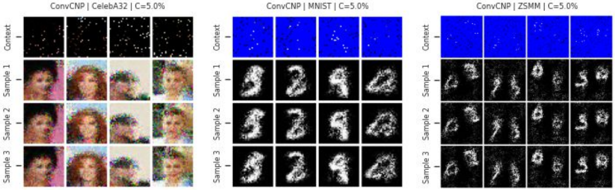
ConvCNP (cont'd)



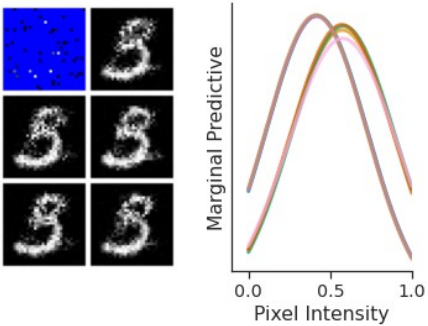
Issues with the CNPF

One of the main issues in the CNPF is caused by the **factorisation** assumption on the predictive - we cannot draw coherent function samples, as could be done with a GP.





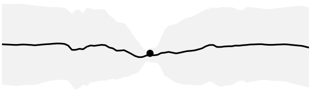
Issues with the CNPF (cont'd)



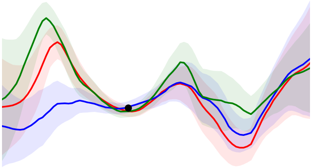
Another issue is the **Gaussianity** assumption - this does not allow for multimodality in the predictive.

Latent Neural Processes

CNPs do not distinguish **noise** and uncertainty due to **finite** $D_{\mathcal{C}}$.



(a) CNP



(b) What we would like to have

CNPs push function uncertainty to the noise layer!

Why care about function uncertainty?

❶ **Bayesian Optimisation** (Thompson sampling)

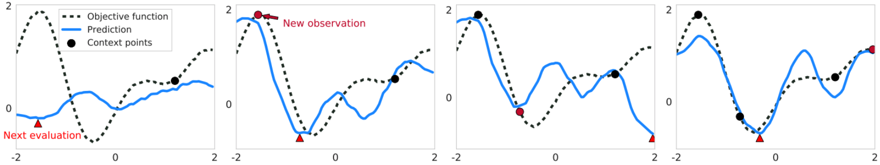


Figure 2: Bayes. opt. with Thompson sampling [Garnelo et al., 2018b].

❷ **Reinforcement Learning** (Contextual bandits)

❸ Sample **plausible functions** for downstream estimation

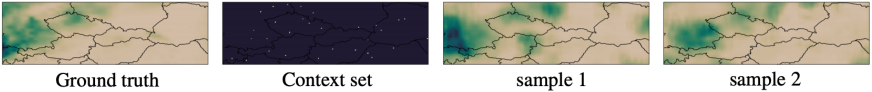
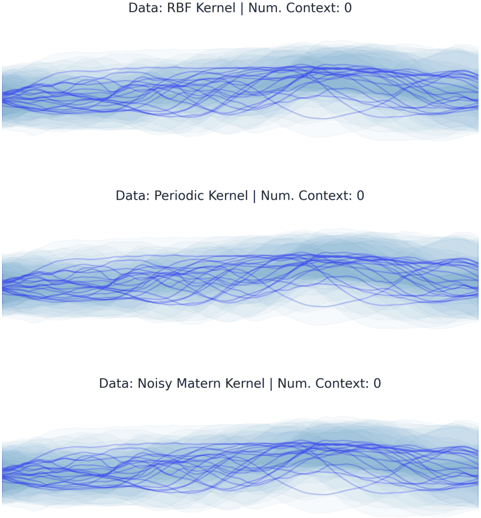


Figure 3: Precipitation over Europe, edited from Foong et al. [2020].

Why care about function uncertainty?

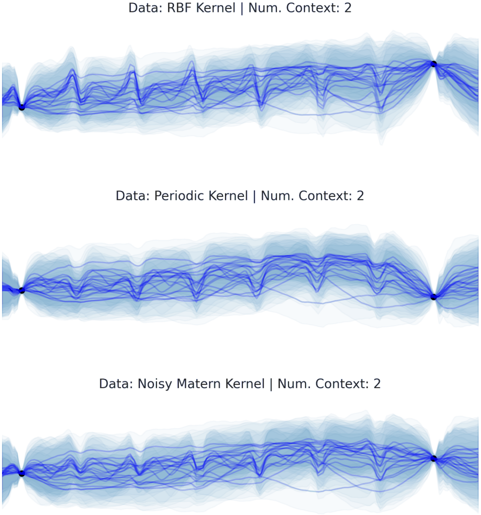
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

Why care about function uncertainty?

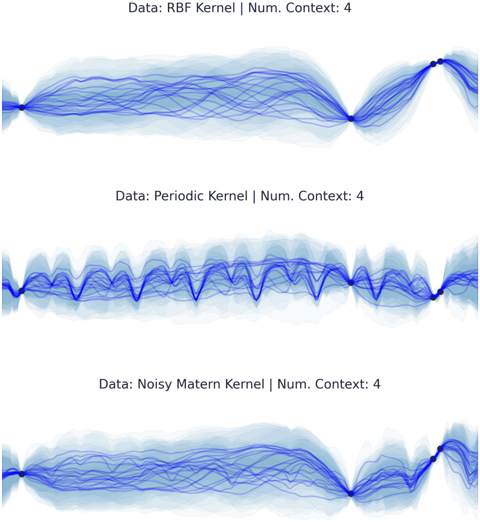
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

Why care about function uncertainty?

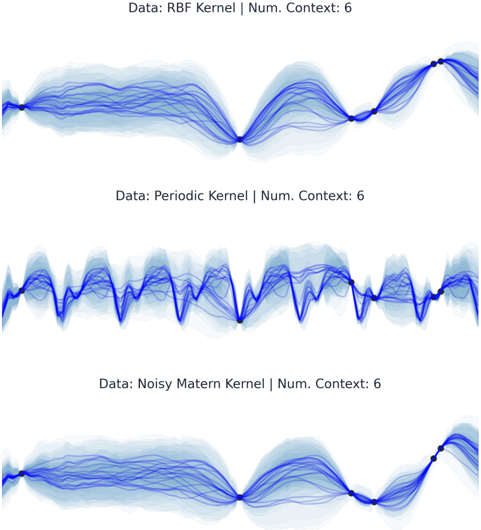
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

Why care about function uncertainty?

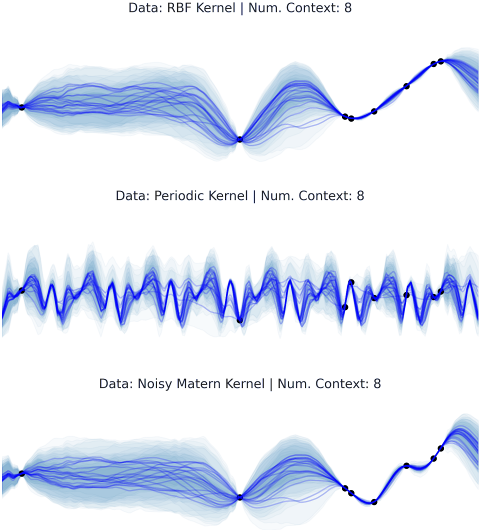
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

Why care about function uncertainty?

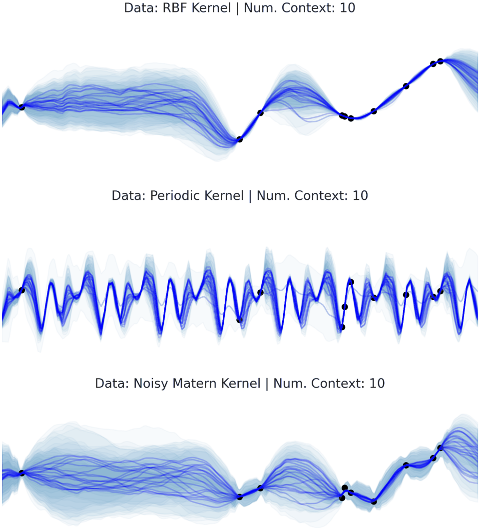
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

Why care about function uncertainty?

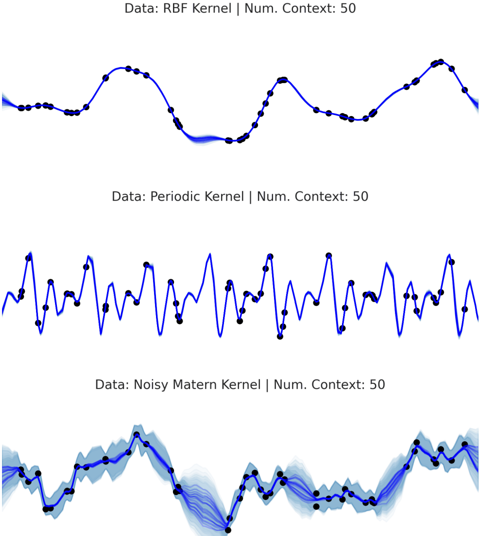
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

Why care about function uncertainty?

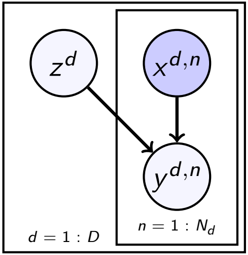
❸ Sample **plausible functions** for downstream estimation



Model identification with neural processes [Dubois et al., 2020].

The Latent Neural Process model

We **would like to** use this model



Learn an approximate posterior by VI:

$$p(D_{\mathcal{C}\cup\mathcal{T}}) \geq \mathbb{E}_{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\left[\log p(D_{\mathcal{C}\cup\mathcal{T}}|z) + \log \frac{p(z)}{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\right]$$

Garnelo et al. [2018b] introduce, this objective **but do not use it**.

Anecdotal evidence that it causes **under-fitting**. Instead they consider

$$p(D_{\mathcal{T}}|D_{\mathcal{C}}) \geq \mathbb{E}_{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\left[\log p(D_{\mathcal{T}}|z) + \log \frac{p(z|D_{\mathcal{C}})}{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\right]$$

the marginal likelihood of the target **conditioned on the context**.

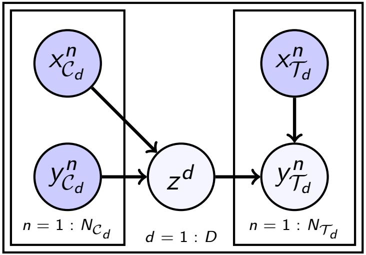
The Latent Neural Process model

$$p(D_{\mathcal{T}}|D_{\mathcal{C}}) \geq \mathbb{E}_{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\left[\log p(D_{\mathcal{T}}|z) + \log \frac{p(z|D_{\mathcal{C}})}{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\right]$$

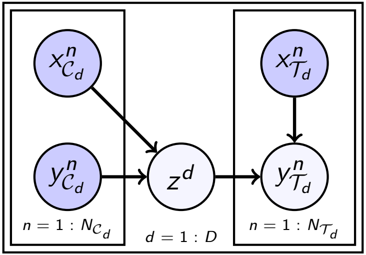
But $p(z|D_{\mathcal{C}})$ is intractable. Garnelo approximate $p(z|D_{\mathcal{C}}) \approx q(z|D_{\mathcal{C}})$

$$\mathcal{L} = \mathbb{E}_{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\left[\log p(D_{\mathcal{T}}|z) + \log \frac{q(z|D_{\mathcal{C}})}{q(z|D_{\mathcal{C}\cup\mathcal{T}})}\right]$$

Not a lower bound anymore. Can be regarded as **defining** the model



The Latent Neural Process model



Defines the conditional prior as $q(z|D_{\mathcal{C}})$.

Chooses the varational posterior $q(z|D_{\mathcal{C}\cup\mathcal{T}})$.

But this **does not correspond to a single consistent Bayesian model**.

Performs VI over a **family of models** (one for each possible dataset).

Not a single consistent Bayesian model

Given context data $\mathcal{D}_\mathcal{C}$, conditional prior **defined as**

$$p(z|\mathcal{D}_\mathcal{C}) := q(z|\mathcal{D}_\mathcal{C})$$

New datum $x^{(n+1)}, y^{(n+1)}$ arrives

$$\mathcal{D}_{\mathcal{C}'} = \mathcal{D}_\mathcal{C} \cup \{(x^{(n+1)}, y^{(n+1)})\}$$

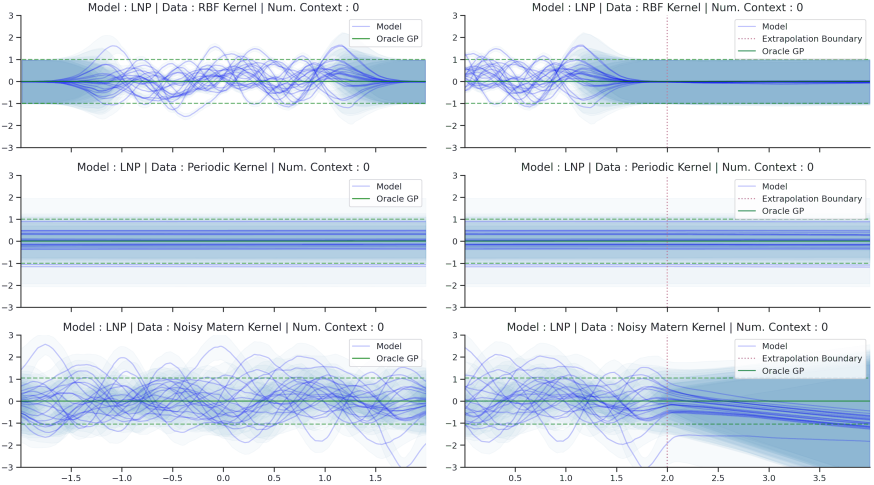
Should update the prior by Bayes

$$p(z|\mathcal{D}_{\mathcal{C}'}) = \frac{p(y^{(n+1)}|x^{(n+1)}, z)q(z|\mathcal{D}_\mathcal{C})}{Z}$$

Instead LNP **defines a separate model for** $\mathcal{D}_{\mathcal{C}'}$

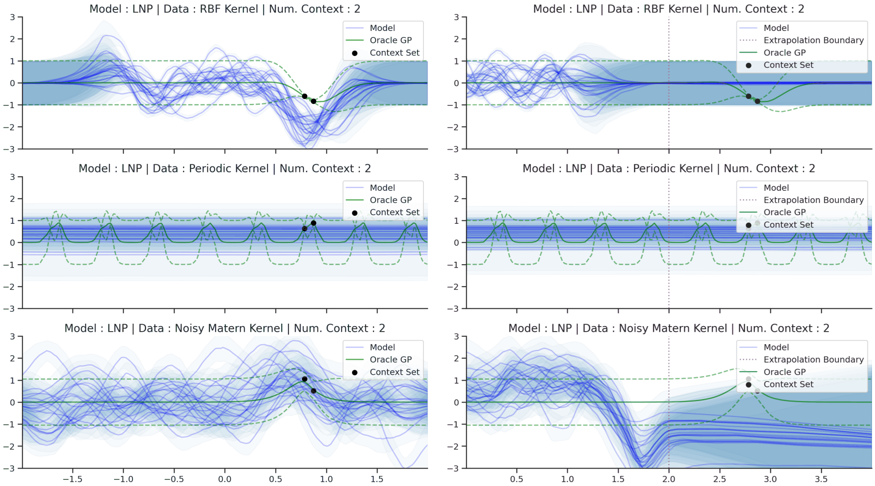
$$p(z|\mathcal{D}_{\mathcal{C}'}) := q(z|\mathcal{D}_{\mathcal{C}'})$$

Latent Neural Process data fits



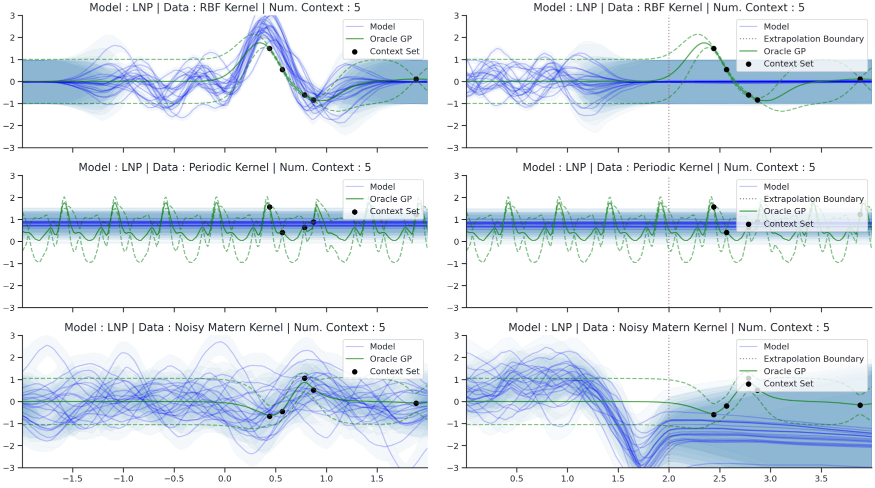
LNP fitted to data, from Dubois et al. [2020].

Latent Neural Process data fits



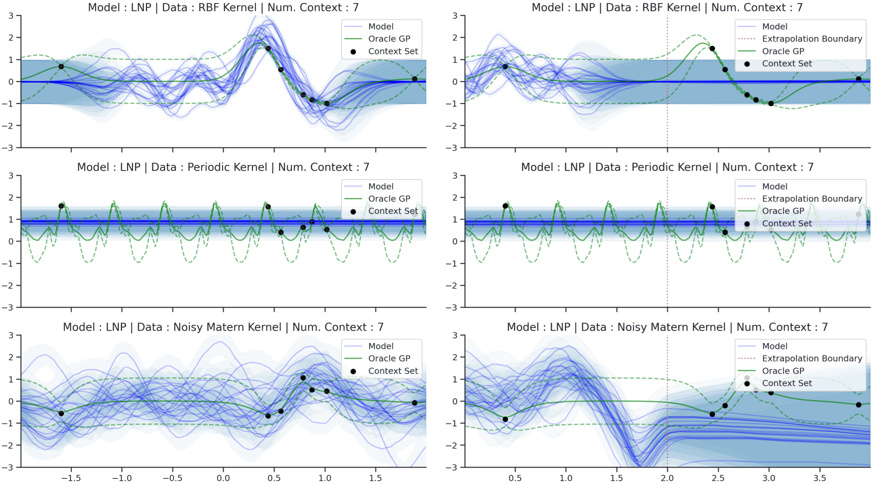
LNP fitted to data, from Dubois et al. [2020].

Latent Neural Process data fits



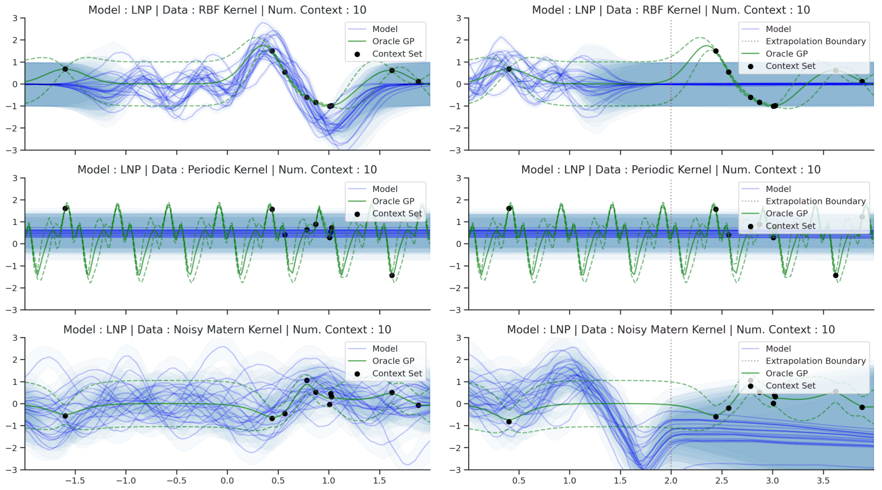
LNP fitted to data, from Dubois et al. [2020].

Latent Neural Process data fits



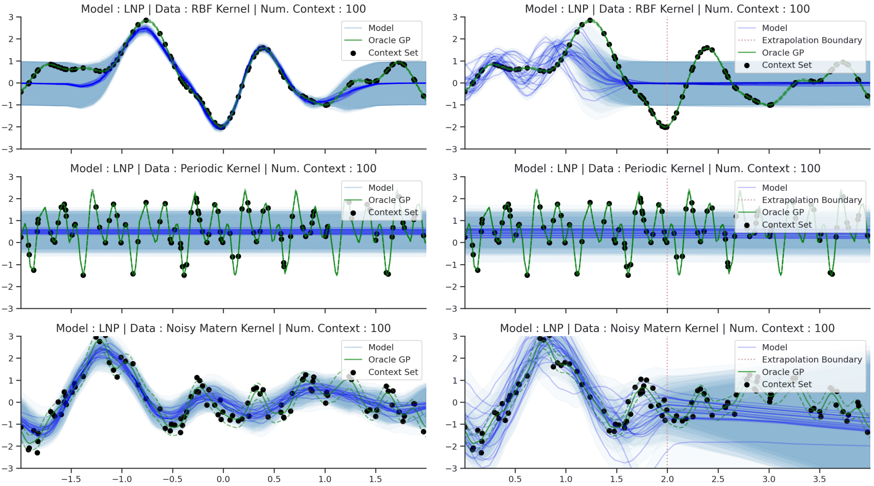
LNP fitted to data, from Dubois et al. [2020].

Latent Neural Process data fits



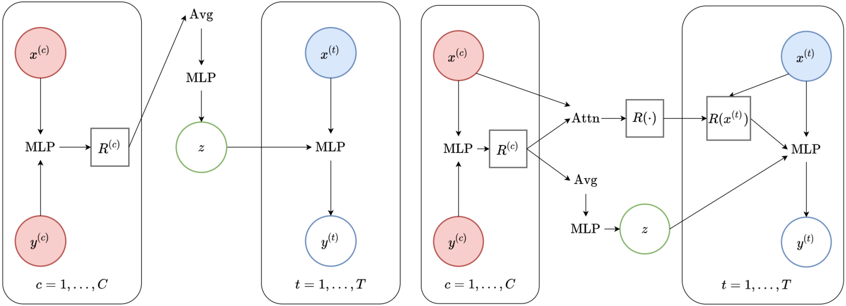
LNP fitted to data, from Dubois et al. [2020].

Latent Neural Process data fits



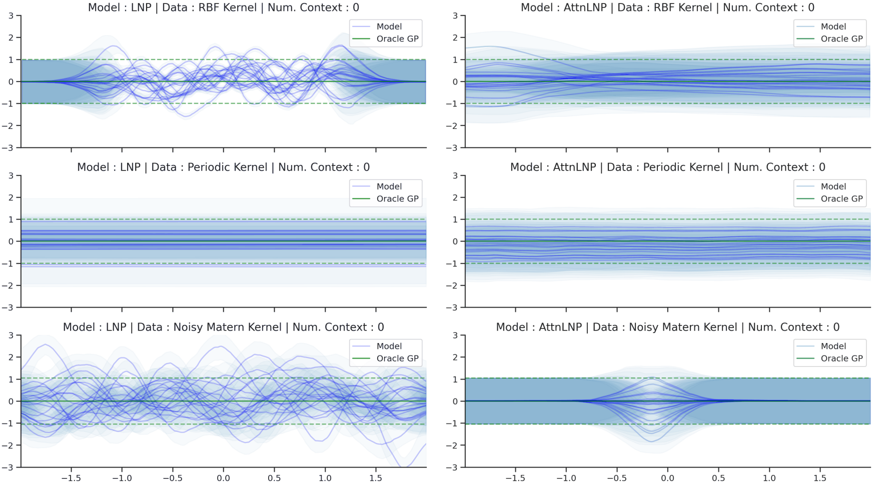
LNP fitted to data, from Dubois et al. [2020].

Attentive Latent Neural Process



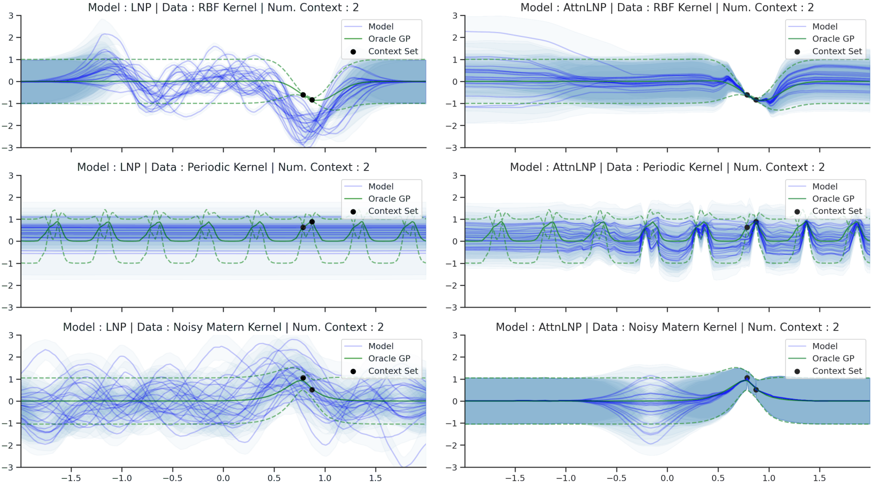
Computational graphs (left) Latent NP (right) Attentive LNP [Dubois et al., 2020].

Attentive Latent Neural Process



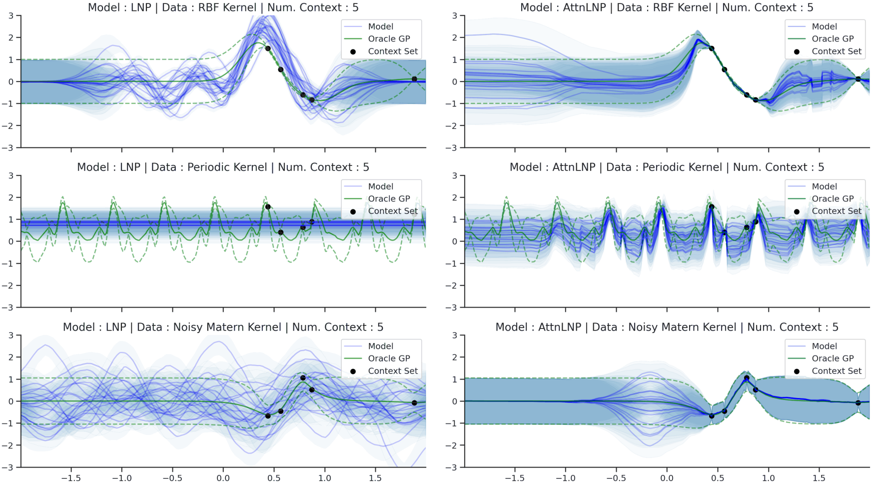
LNP with/without attention from Dubois et al. [2020].

Attentive Latent Neural Process



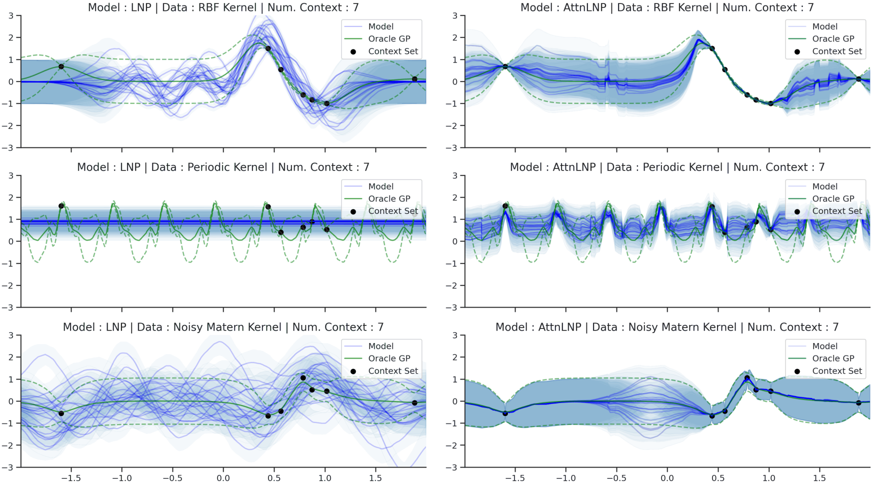
LNP with/without attention from Dubois et al. [2020].

Attentive Latent Neural Process



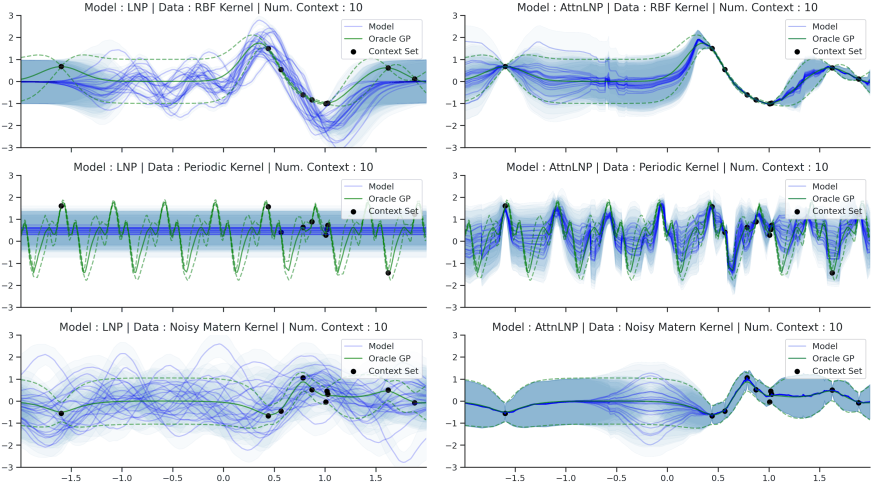
LNP with/without attention from Dubois et al. [2020].

Attentive Latent Neural Process



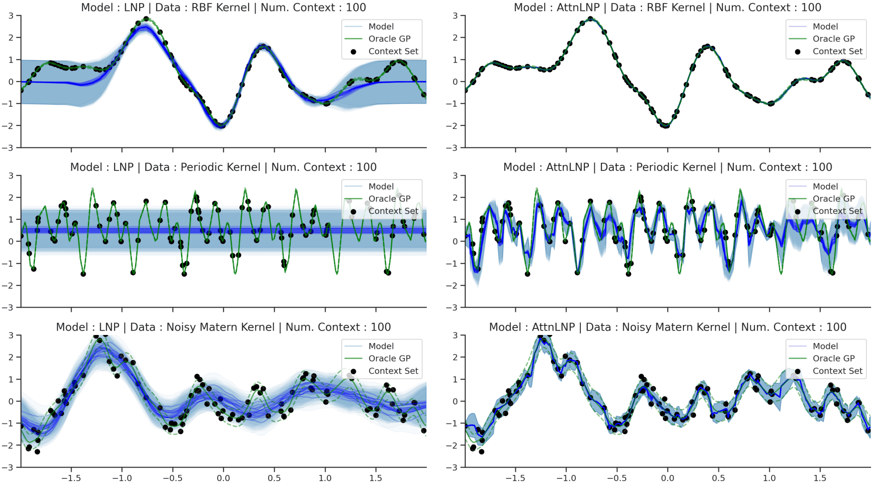
LNP with/without attention from Dubois et al. [2020].

Attentive Latent Neural Process



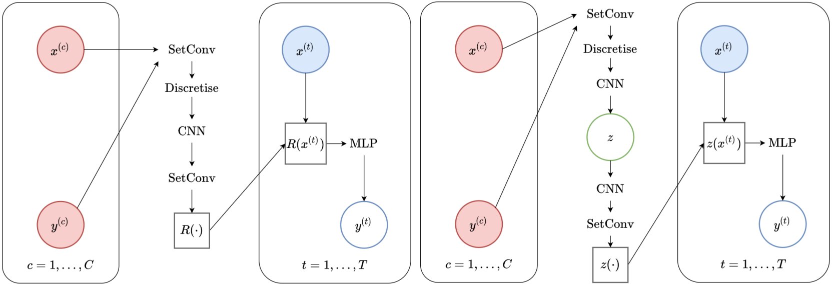
LNP with/without attention from Dubois et al. [2020].

Attentive Latent Neural Process



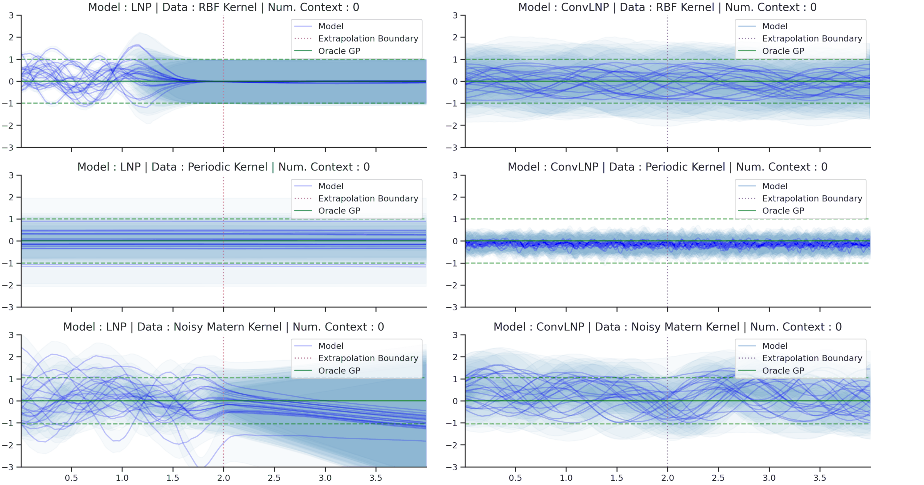
LNP with/without attention from Dubois et al. [2020].

Convolutional Latent Neural Process



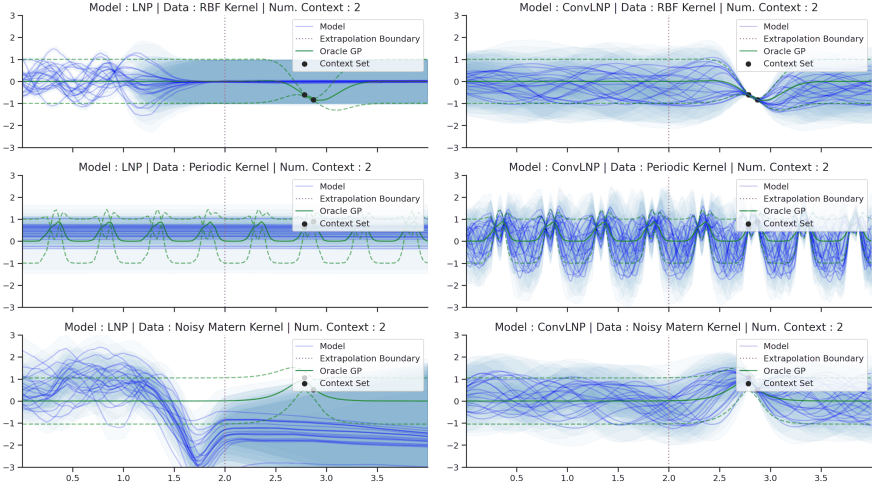
Computational graphs (left) ConvCNP (right) Latent ConvNP [Dubois et al., 2020].

Convolutional Latent Neural Process



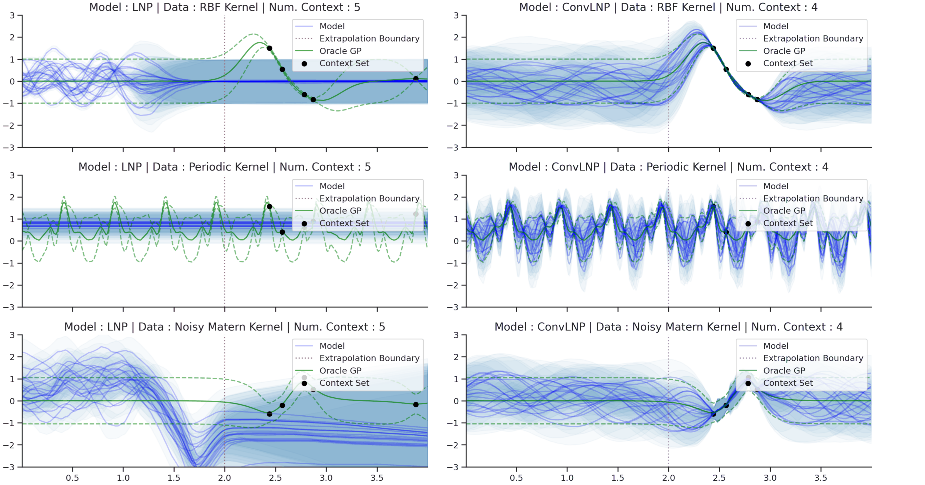
LNP with/without convolutional architecture from Dubois et al. [2020].

Convolutional Latent Neural Process



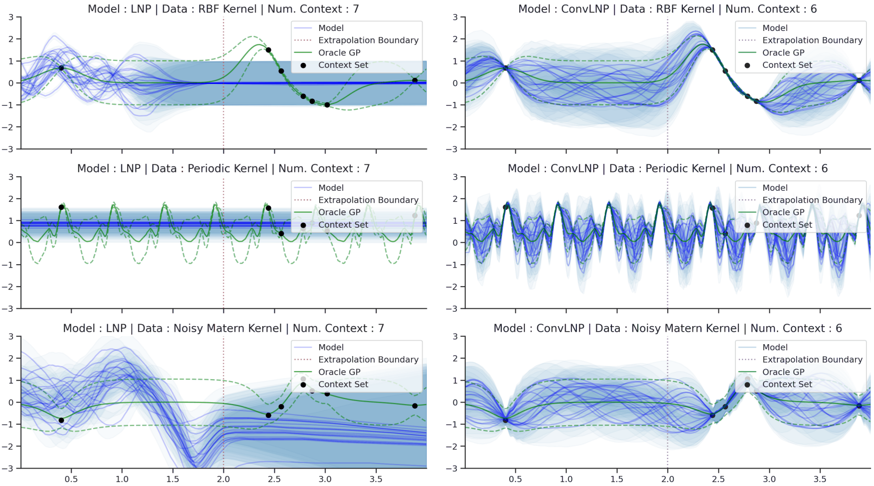
LNP with/without convolutional architecture from Dubois et al. [2020].

Convolutional Latent Neural Process



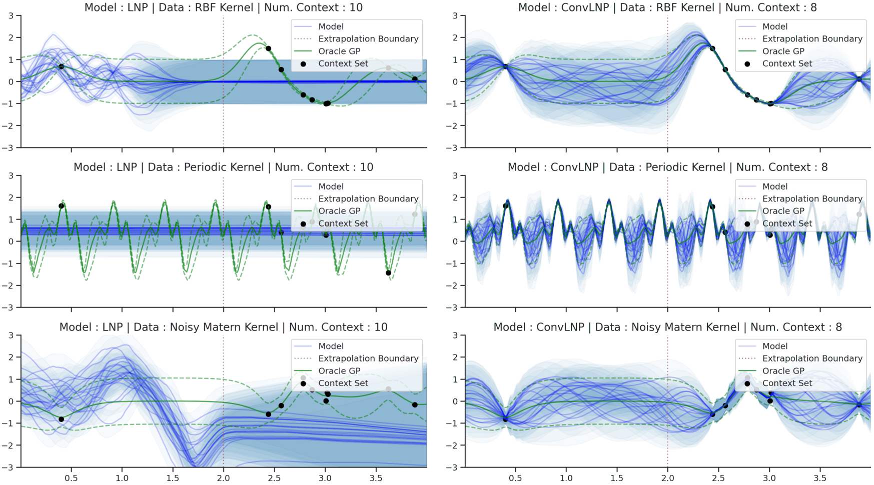
LNP with/without convolutional architecture from Dubois et al. [2020].

Convolutional Latent Neural Process



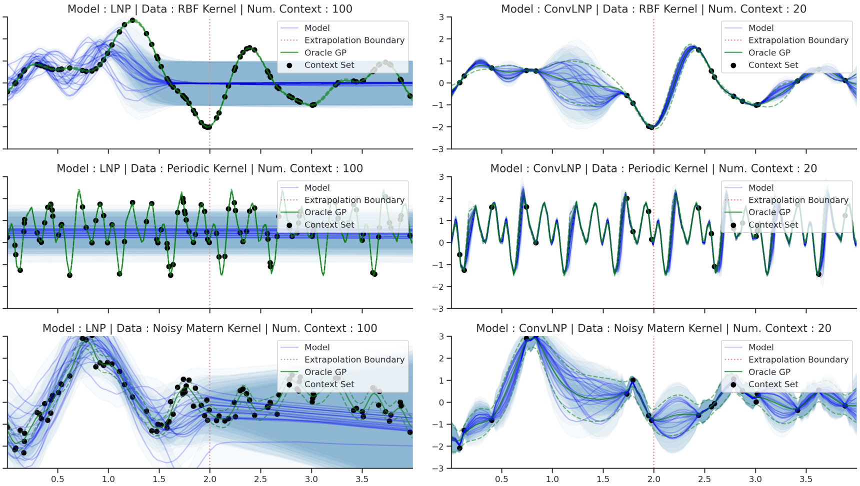
LNP with/without convolutional architecture from Dubois et al. [2020].

Convolutional Latent Neural Process



LNP with/without convolutional architecture from Dubois et al. [2020].

Convolutional Latent Neural Process



LNP with/without convolutional architecture from Dubois et al. [2020].

Conclusions

- NPs do **meta-learning** on functions.
- Family splits into **conditional** and **latent** models.

Strong points:

- Fast inference at test time.
- Well calibrated uncertainty (if enough $\mathcal{D}$'s available).
- Data driven, more flexible than hand-picked priors.
- Can bake in (some) required properties - translation equivariance.

Weak points:

- Need a large collection of meta-learning datasets.
- Underfitting and smoothness issues.

References I

Y. Dubois, J. Gordon, and A. Y. Foong. Neural process family. `http://yanndubs.github.io/Neural-Process-Family/`, September 2020.

A. Foong, W. Bruinsma, J. Gordon, Y. Dubois, J. Requeima, and R. Turner. Meta-learning stationary stochastic process prediction with convolutional neural processes. *Advances in Neural Information Processing Systems*, 33, 2020.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713, 2018a.

M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

References II

J. Gordon, W. P. Bruinsma, A. Y. Foong, J. Requeima, Y. Dubois, and R. E. Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

E. Wagstaff, F. Fuchs, M. Engelcke, I. Posner, and M. A. Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494, 2019.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. *Advances in Neural Information Processing Systems*, 30:3391–3401, 2017.