Meta Learning

Alan Chau, Ana Ignatieva, James Thornton & Valerie Bradley

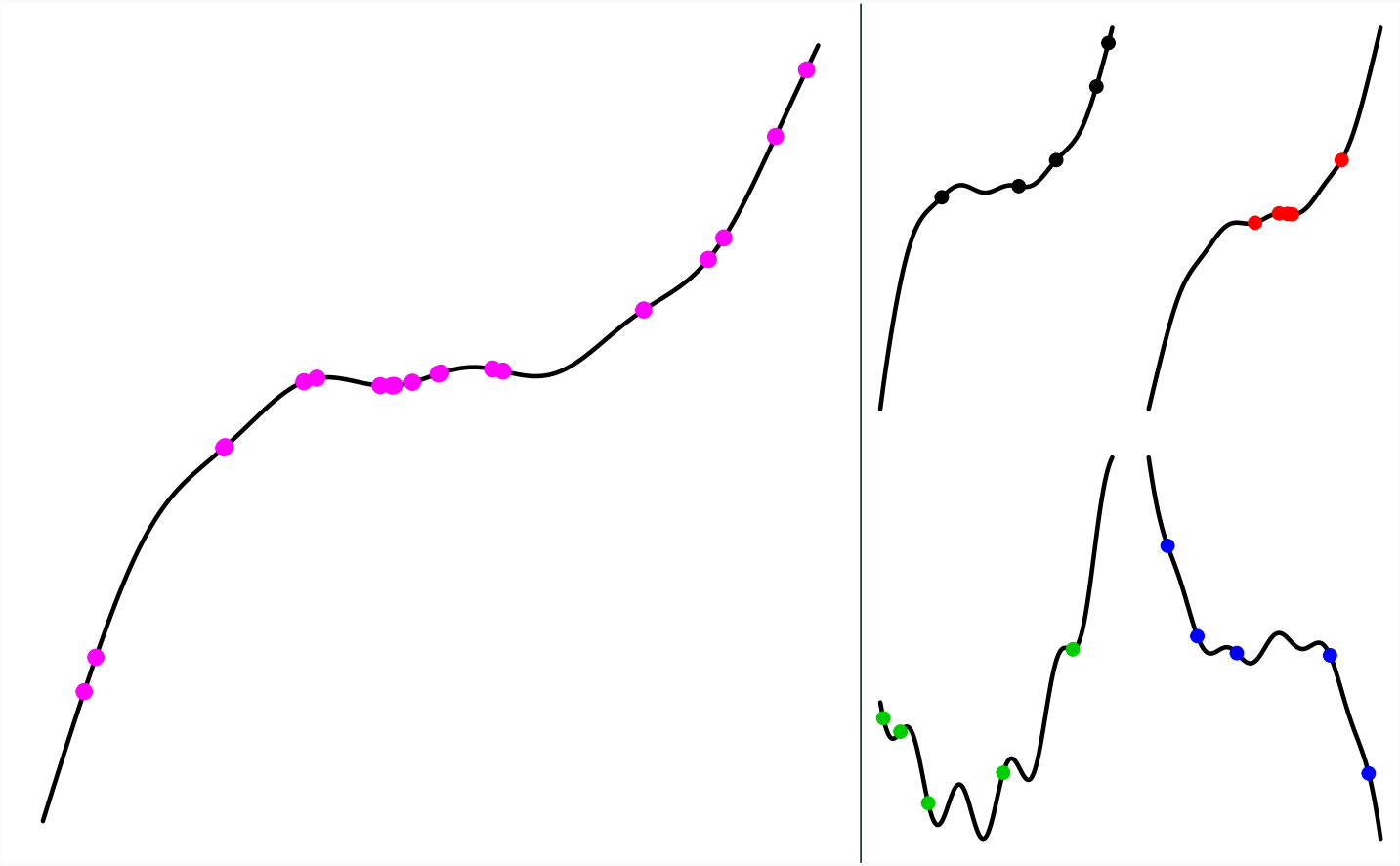
Department of Statistics, University of Oxford

Introduction

Meta-learning

- "Learning to learn"
- Approximate a distribution over functions
- First: learn about the general domain with a large training set and a variety of tasks
- Then: learn a function for a specific task using this knowledge and a small number of new data points (few-shot learning)

Introduction



Want to learn the distribution over functions, then do few-shot learning

Introduction

We look at:

- Neural processes [Garnelo et al., 2018b]
  - + Conditional neural processes [Garnelo et al., 2018a]
  - + Attentive neural processes [Kim et al., 2019]
- Model-agnostic meta-learning (MAML) [Finn et al., 2017]

Neural Processes

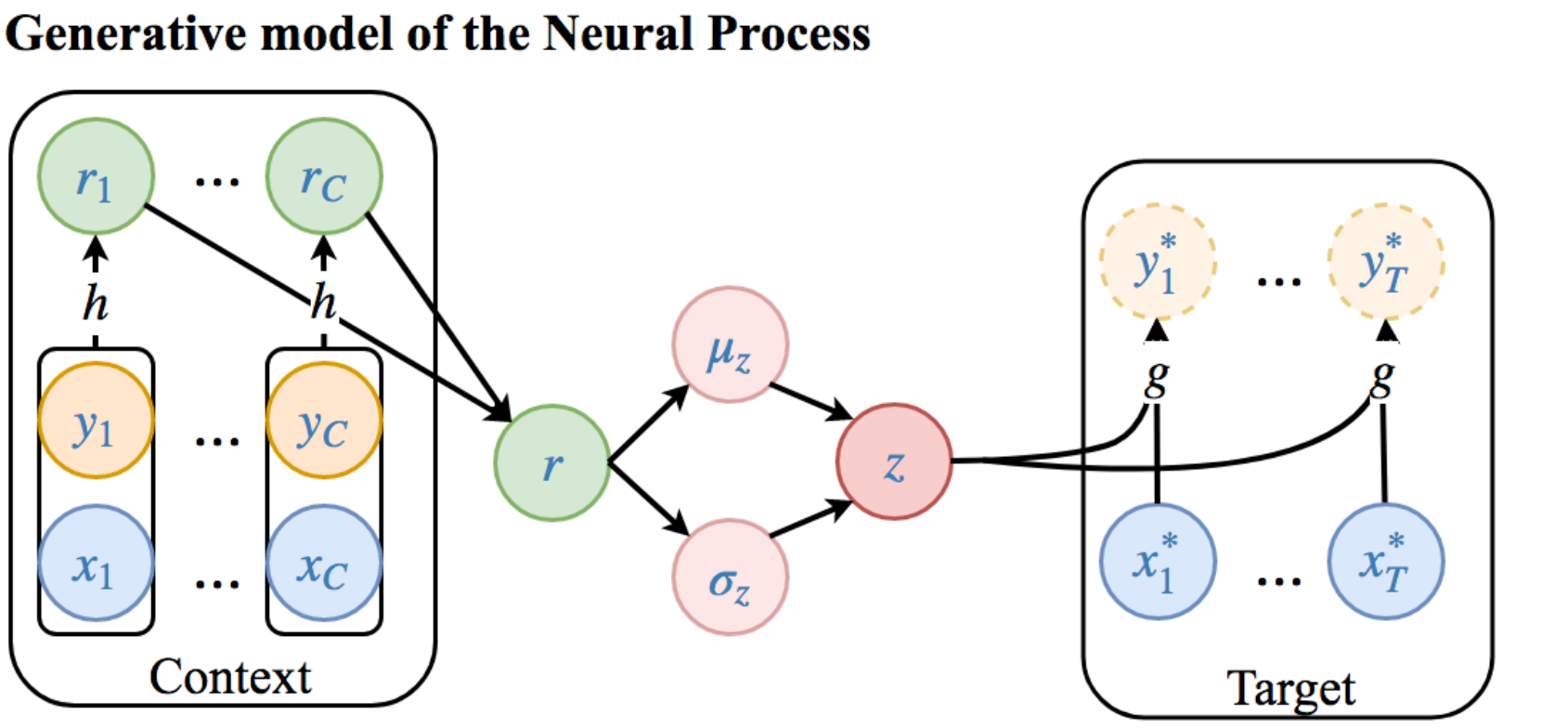
- Combine ideas of Neural Networks and Gaussian Processes
- NNs: computationally efficient, learn from data, can't update outputs after training
- GPs: computationally intensive, flexible, need kernel, distributions over functions, give measure of uncertainty
- NPs:
  - learn function approximations from training data
  - fast evaluation at test time
  - uncertainty quantification

Neural Processes

Generative model:

1. Context set $\{(x_i, y_i)\}_{i=1}^{C}$ and unlabelled $\{(x_t^*)\}_{t=1}^{T}$.
2. Pass context set through NN h to learn the latent representation vector $\{r_i\}_{i=1}^{C}$.
3. Aggregate (mean) to get a representation r of the context.
4. Use r to parametrise Gaussian distribution of a latent variable z
5. Sample z, pass with x_t^*, through a decoder MLP g
   $\rightarrow$ obtain y_t^*.

Neural Processes



From Martens [2018]

Neural Processes

Training:

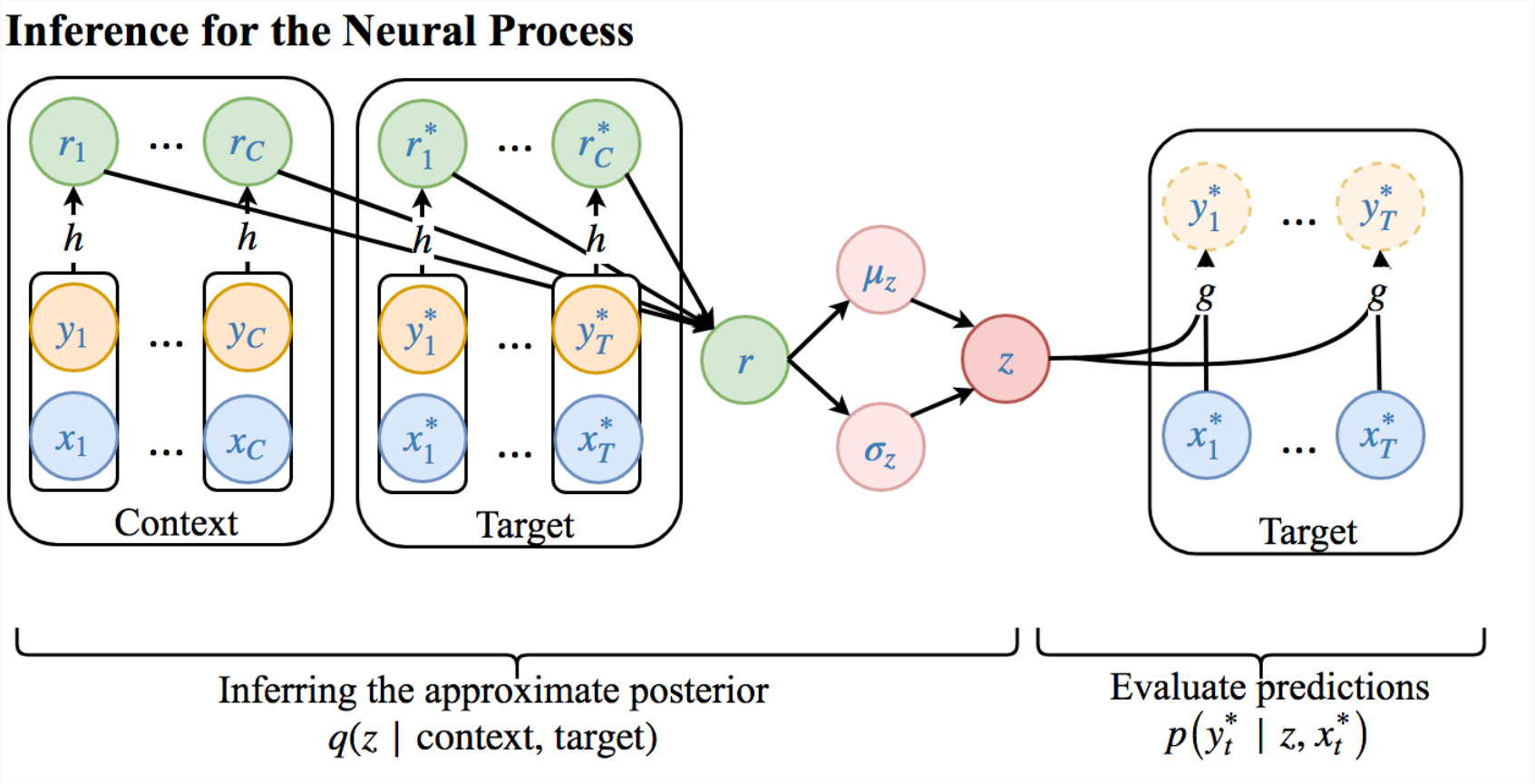
1. Sample $f \sim \mathcal{D}$ and generate a dataset
2. Split into context $\mathcal{C}$ and target set T
3. Make a forward pass with $\mathcal{C}$ to get approximation $q(z|\mathcal{C})$
4. Make a forward pass with full dataset to get approximation $q(z|\mathcal{C}, \mathcal{T})$
5. Make predictions using z and target points x^*, get $p(y_t^*|z, x_t^*)$
6. Compute the loss (ELBO):

$$\text{ELBO} = \mathbb{E}_{z|\mathcal{C},\mathcal{T}}\Big[\sum_{t=1}^{T} \log p(y_t^*|z, x_t^*) + \log \frac{q(z|\mathcal{C})}{q(z|\mathcal{C}, \mathcal{T})}\Big]$$

7. Optimise using gradient descent and back-propagate

Repeat!

Neural Processes



From Martens [2018]

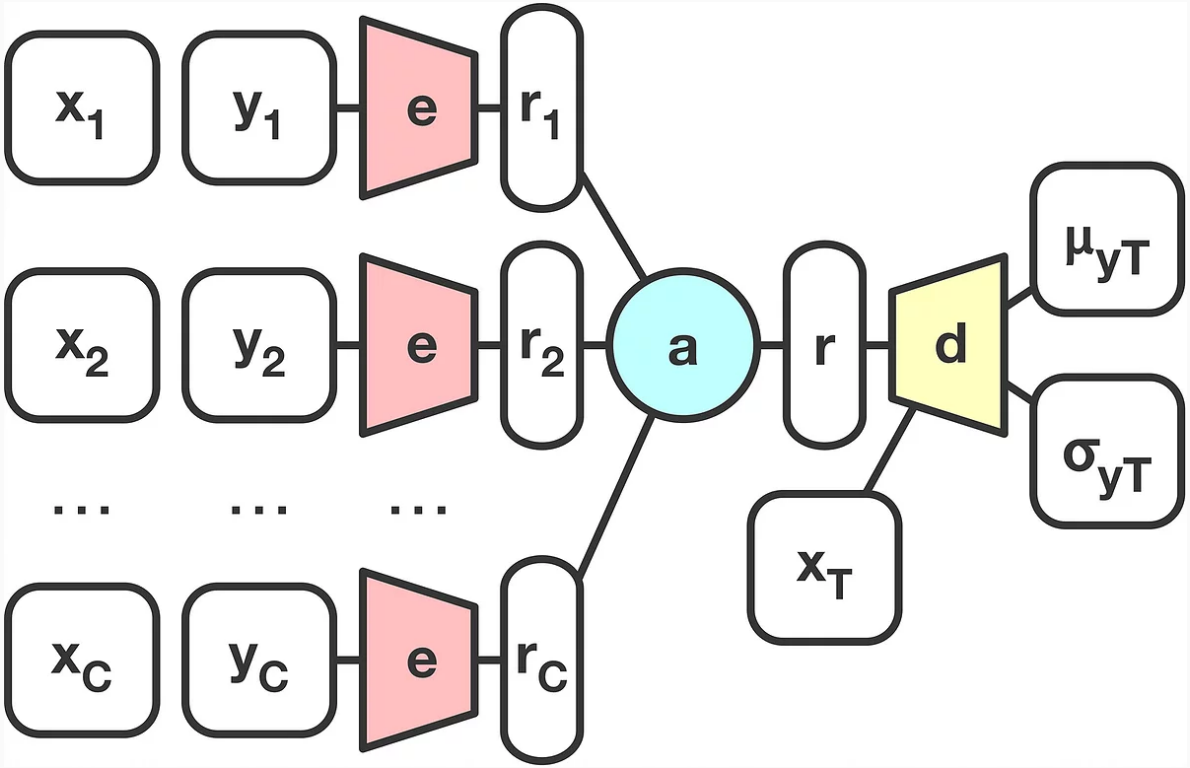
Neural Processes

To sum up:

- NPs capture the variability of the functions seen at training
- At test time, we use this knowledge to make predictions based on a few observations
- Have lots of choices to make:
  - dimensions of r and z
  - NNs h and g
  - how many training iterations
  - how many context points

Conditional Neural Processes

- Similar idea to NPs
- No stochastic global variable z
- Deterministic r to predict mean and variance of target outputs
- Can't generate different function samples for the same context data



From Garnelo et al. [2019]

Attentive Neural Processes

In practice we see NP will usually under-fit the task of interest.

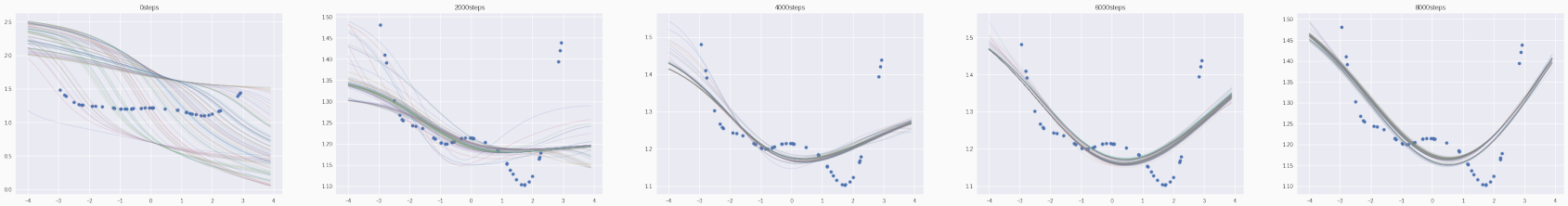


Figure 1: Demonstrating how NP will suffer from underfitting

[Kim et al., 2019] hypothesizes that this is due to taking the mean of latent representation of the context pairs without considering the similarity of these representations with the task we will predict.

Attentive Neural Processes

Borrowing the idea from the smoothness control from Gaussian Processes, we introduce the following metric to measure similarities between context values and target values,

- **Uniform:** $((k_i, v_i)_{i \in I}, q) = \frac{1}{|I|} \sum_i v_i$
- **Laplace:** $((k_i, v_i)_{i \in I}, q) = \sum_i w_i v_i, \quad w_i \propto \exp(-\frac{||q - k_i||_1}{l})$
- **Dotproduct:** $((k_i, v_i)_{i \in I}, q) = \sum_i w_i v_i, \quad w_i \propto \exp(-\frac{||q - k_i||_1}{l})$

Attentive Neural Processes

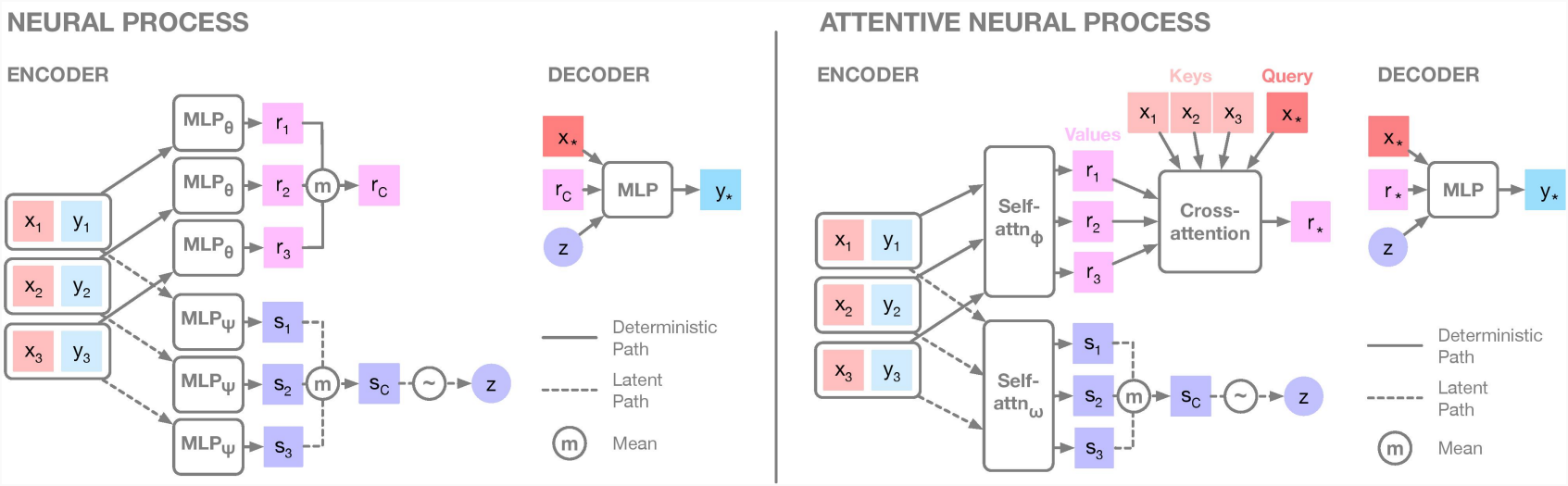
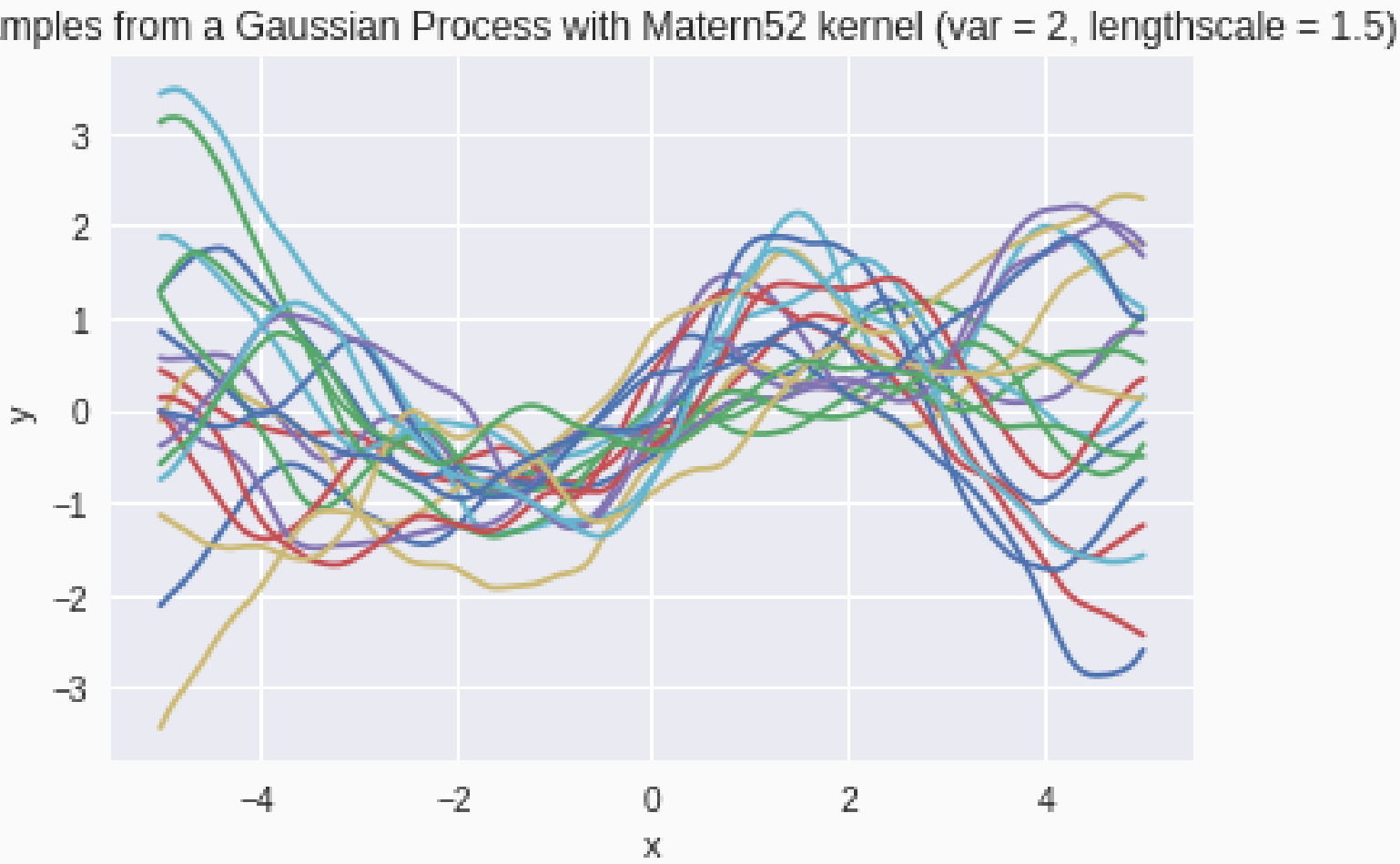


Figure 2: The general architecture of NP and ANP, taken from [Kim et al., 2019]

Experiments

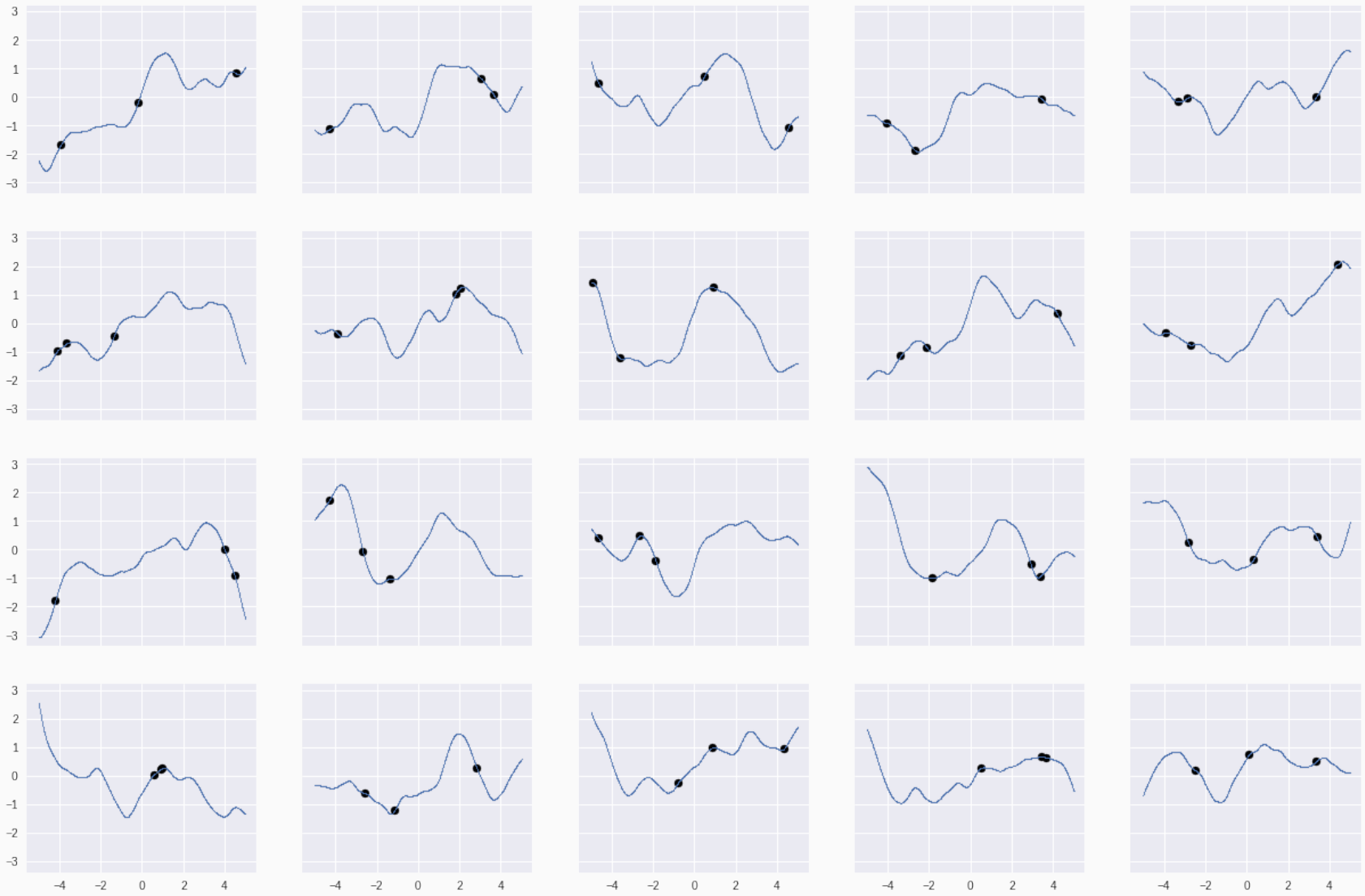
Gaussian Processes

We will now 20 tasks from the same underlying Gaussian Processes (MaternKernel).

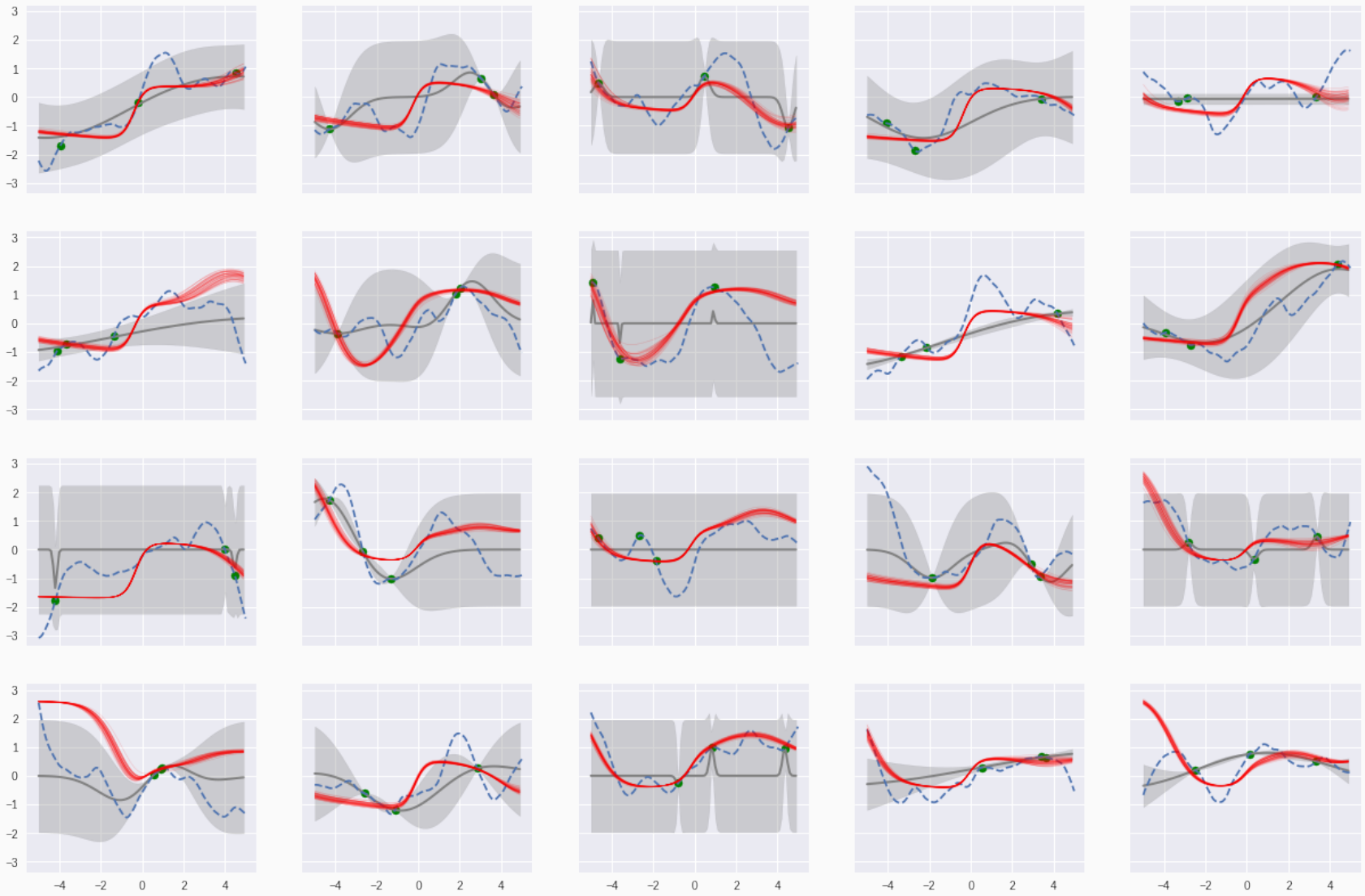


Our target is to see whether the NP will be able to learn the underlying structure shared among the 20 tasks.

Gaussian Processes

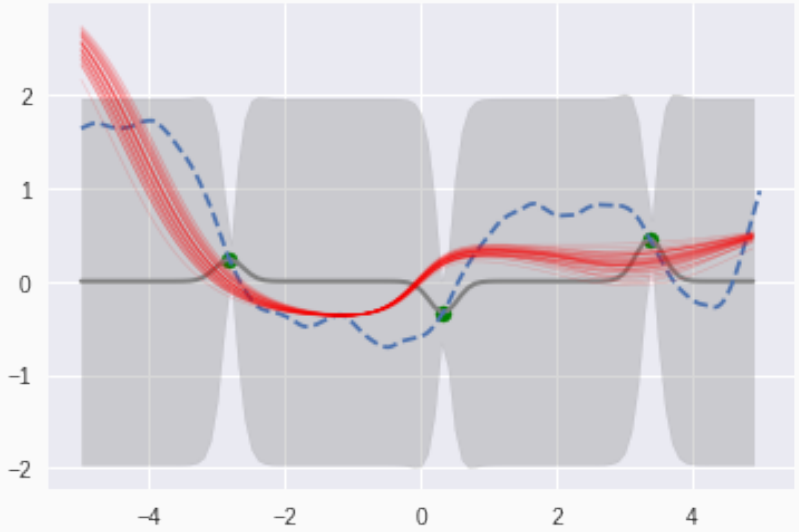


Gaussian Processes

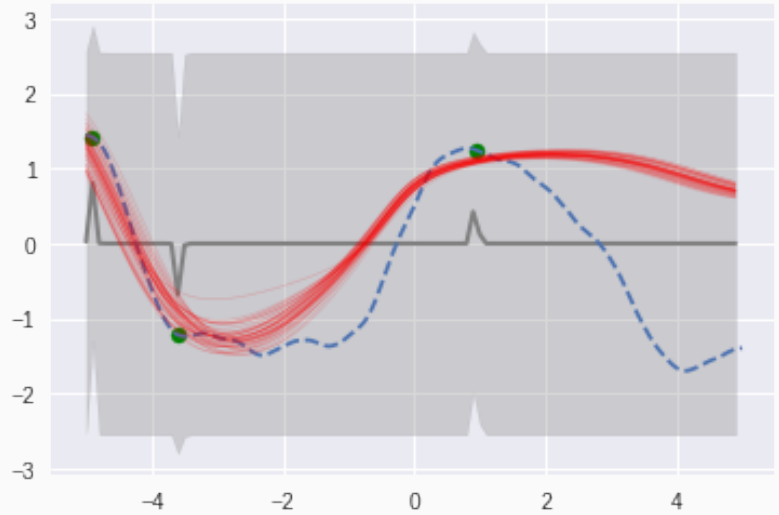


Gaussian Processes

One example the NP did learn the kernel structure from other tasks



Another example the NP did learn the kernel structure from other tasks

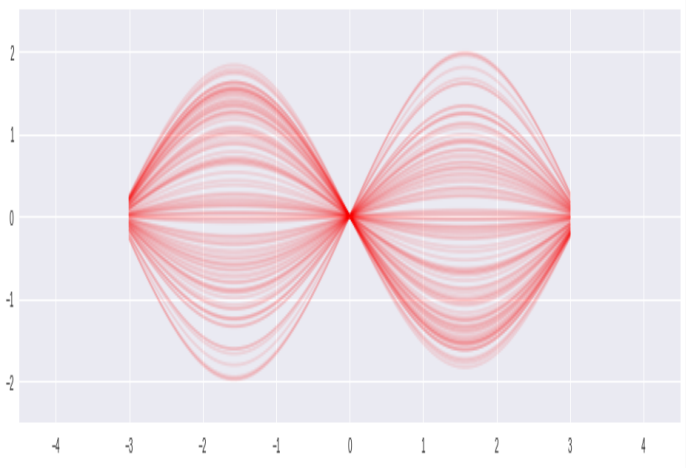


Sine Function Class (1/2)

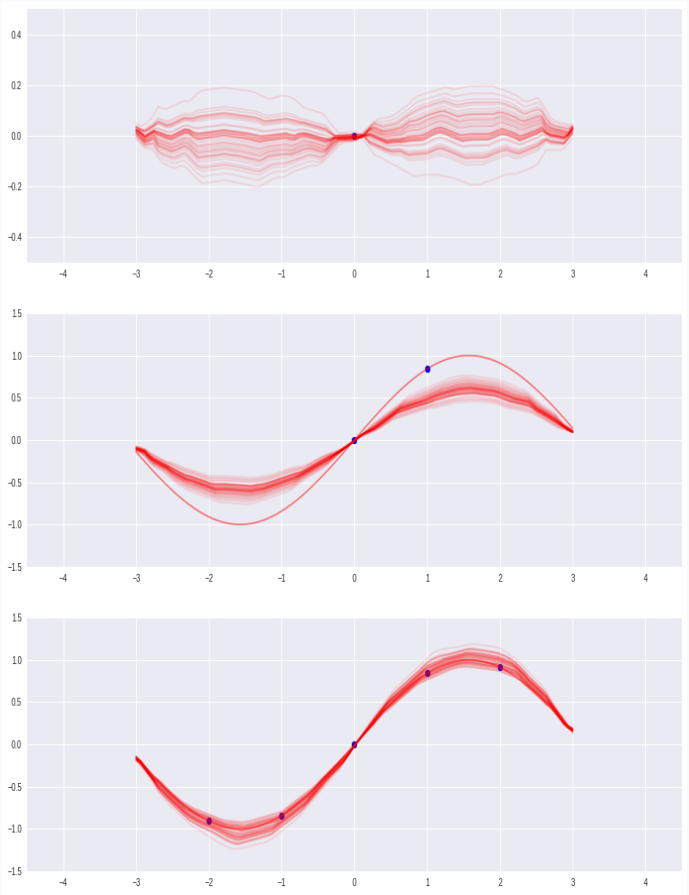
Data Generator:

$$a \sim \mathcal{U}(-2, 2)$$
$$x \sim \mathcal{U}(-3, 3)$$
$$y = a * \sin(x)$$

Context:



Mean prediction:

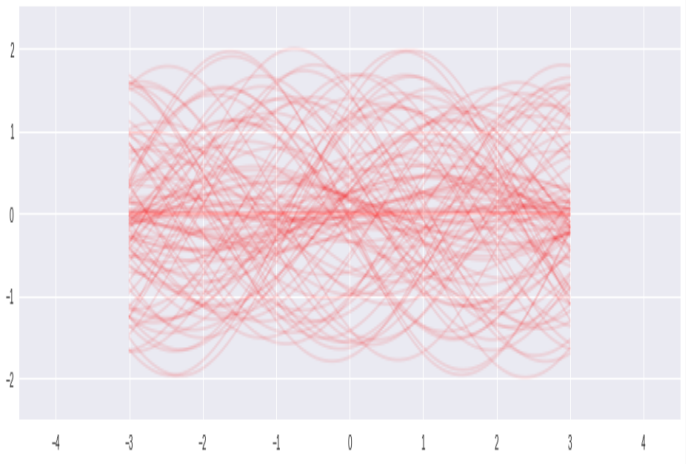


Sine Function Class (2/2)

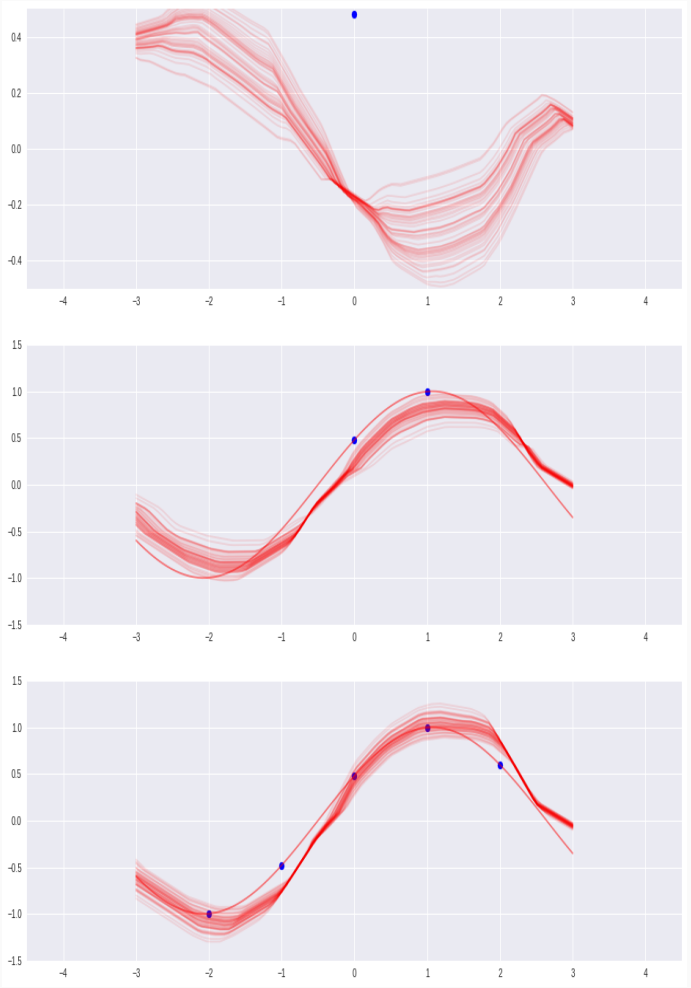
Data Generator:

$$a \sim \mathcal{U}(-2, 2)$$
$$p \sim \mathcal{U}(0, \pi)$$
$$x \sim \mathcal{U}(-3, 3)$$
$$y = a * \sin(x + p)$$

Context:



Mean prediction:



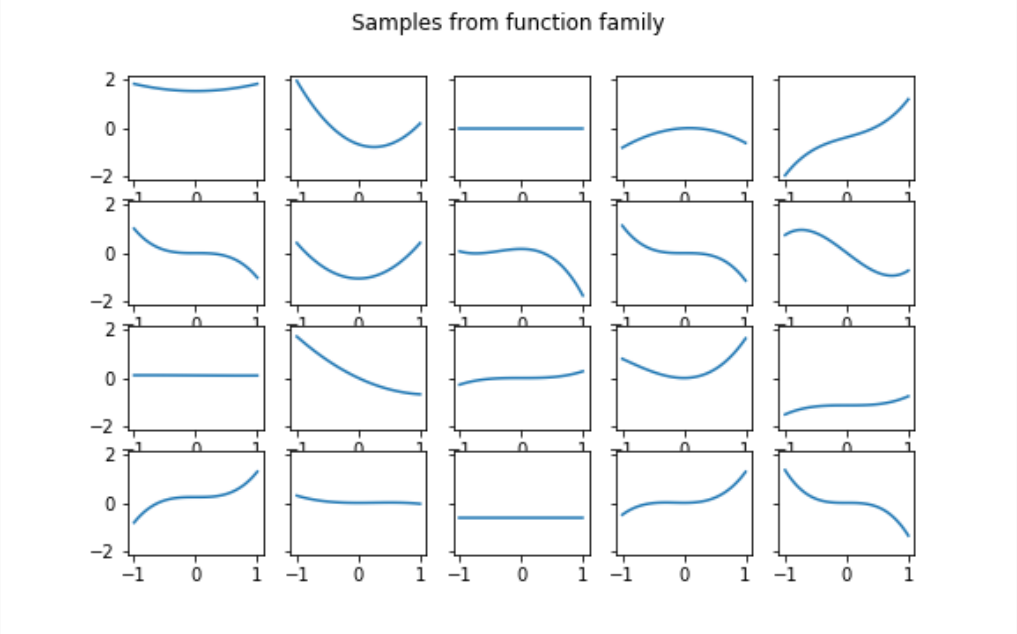
Polynomial Functions (1)

Functions of the form $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

Data generator:

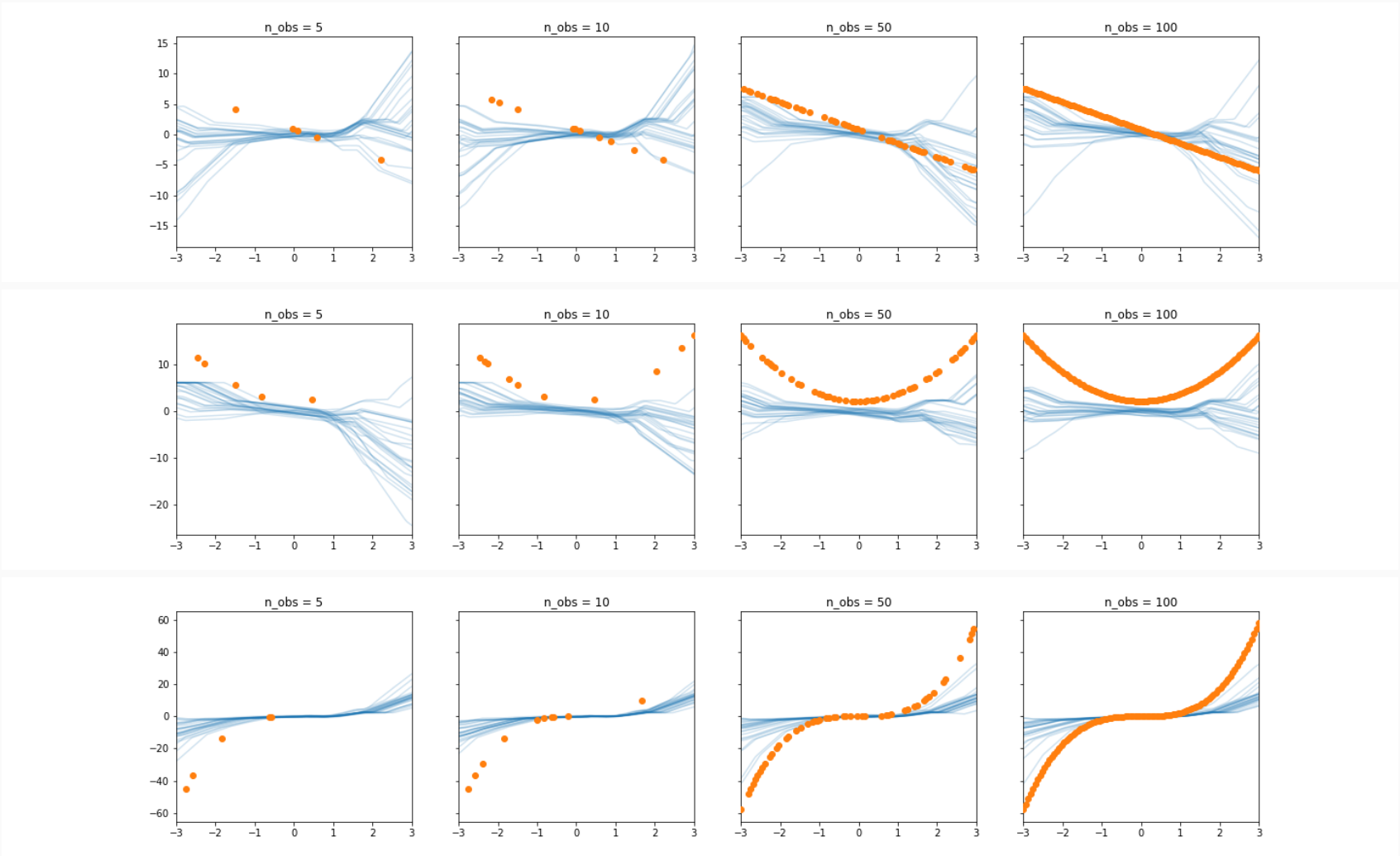
$$\beta_p \sim \begin{cases} \mathcal{N}(0,1) & \text{with prob } (1-\phi) \\ 0 & \text{with prob } \phi \end{cases}$$

$$x \sim \mathcal{U}(-3,3)$$



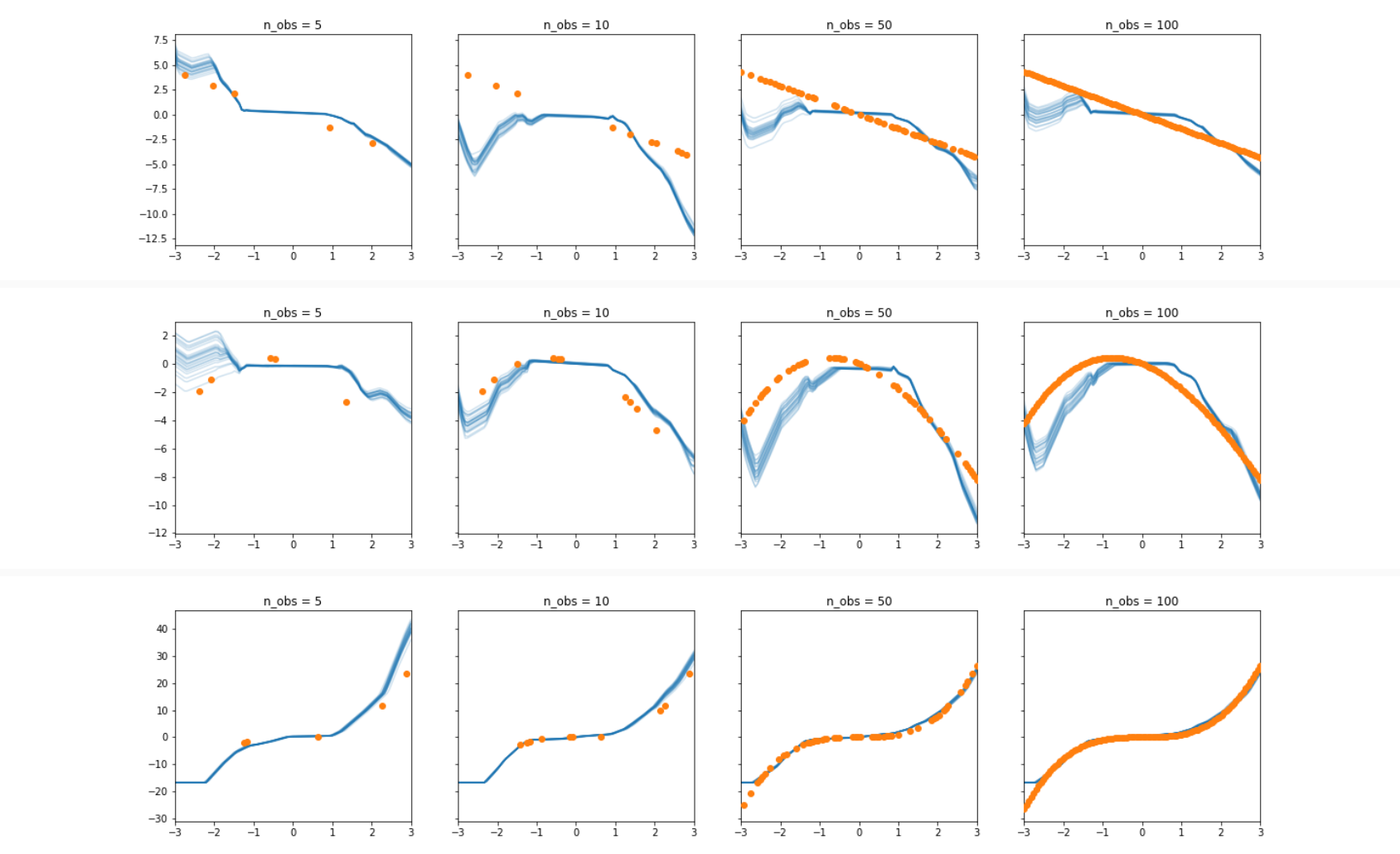
Polynomial Functions (1)

Training iterations: 200,000



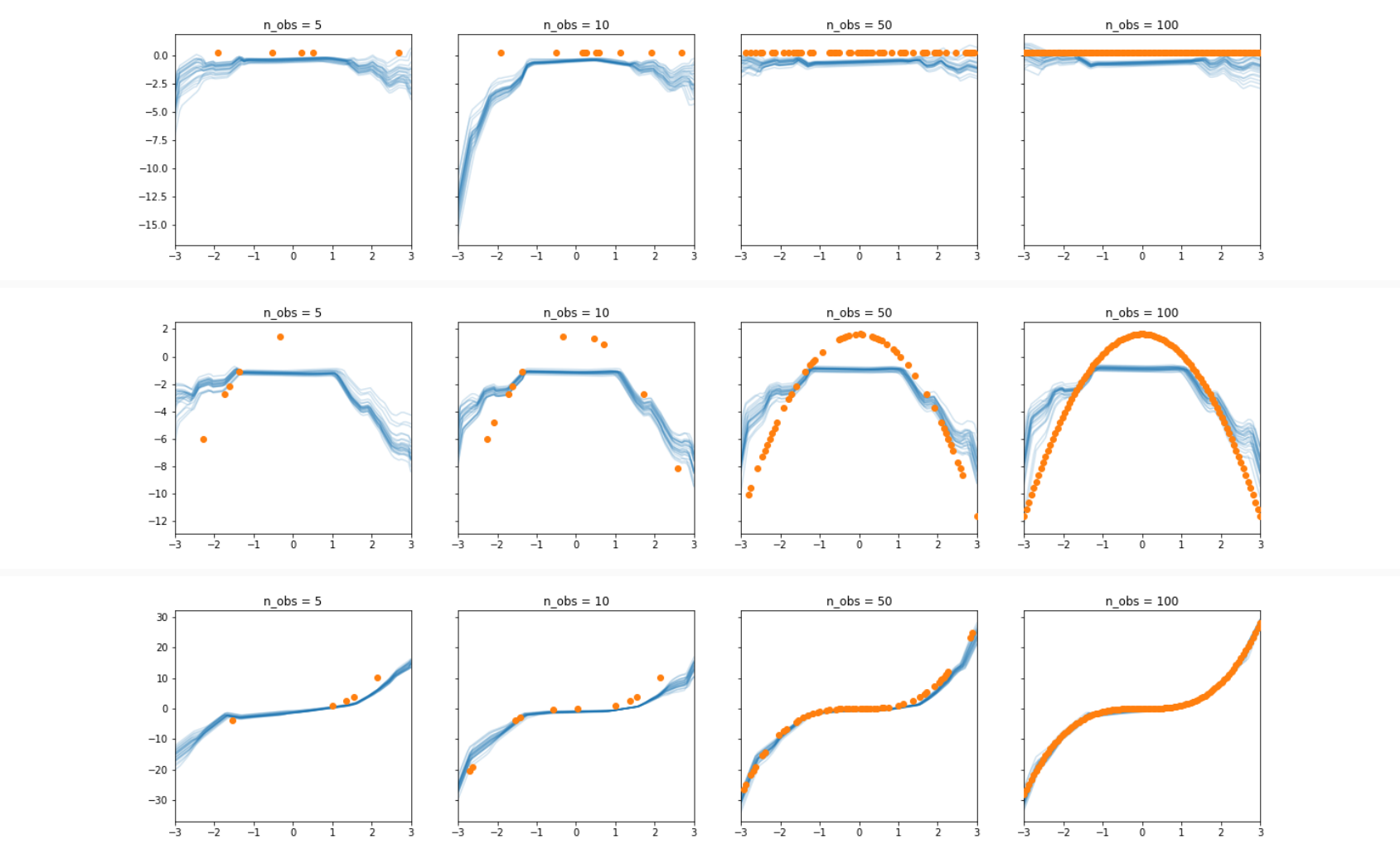
Polynomial Functions (1)

Training iterations: 400,000



Polynomial Functions (1)

Training iterations: 500,000



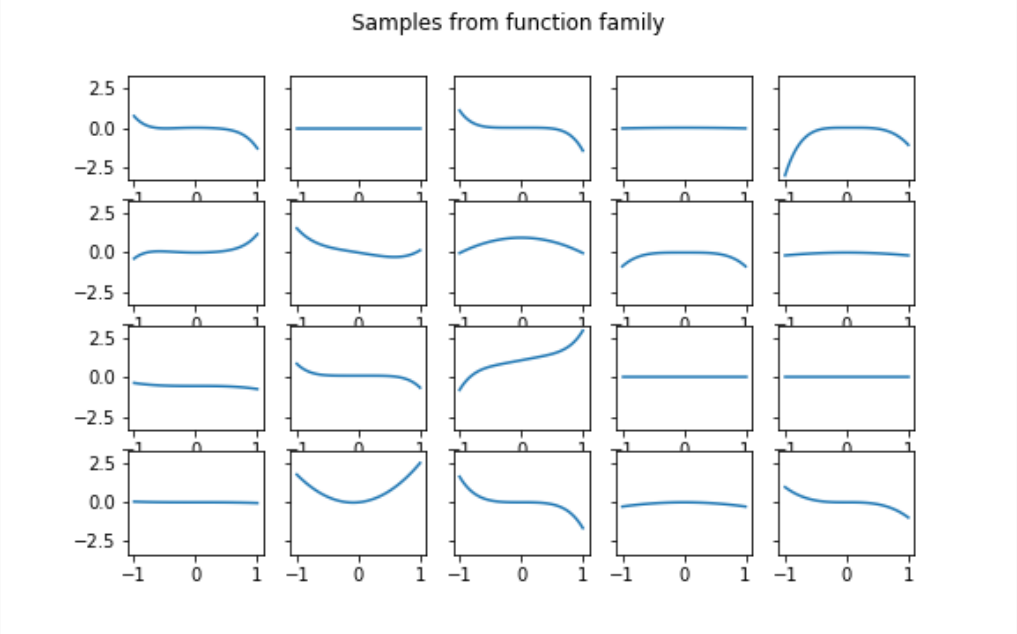
Polynomial Functions (2)

Functions of the form $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5$

Data generator:

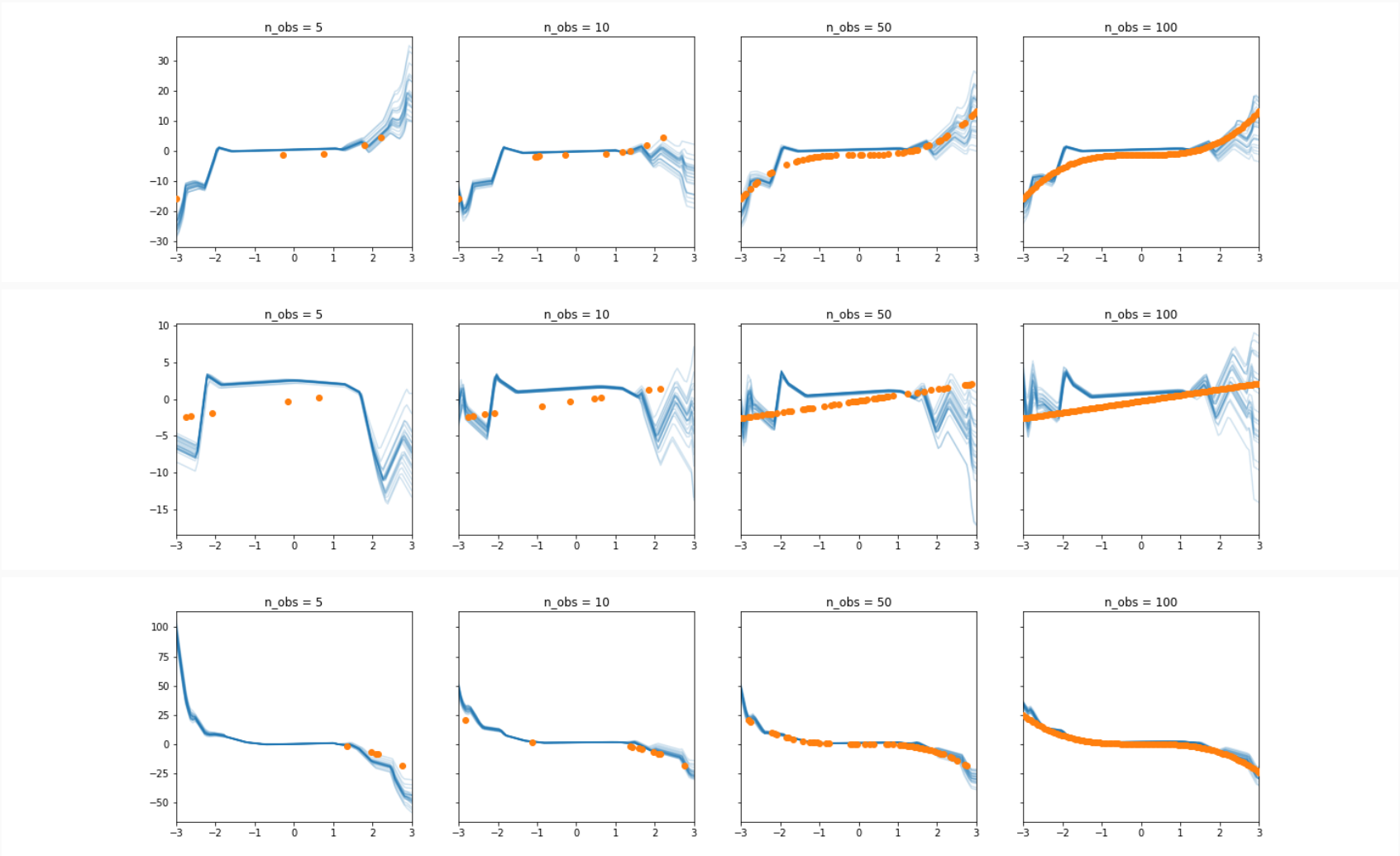
$$\beta_p \sim \begin{cases} \mathcal{N}(0,1) & \text{with prob } (1-\phi) \\ 0 & \text{with prob } \phi \end{cases}$$

$$x \sim \mathcal{U}(-3,3)$$



Polynomial Functions (2)

Training iterations: 500,000



NP Discussion

- NPs were able to fit very large, complex classes of functions
- Increasing training iterations solved most problems
- Predictions are data-efficient: no drastic improvement from additional context points
- Highly sensitive to underlying NN architecture, no clear guidelines on how to specify
  - Number of layers in encoder h and decoder g
  - Activation functions (almost all ReLU here)
  - Dimensions of r and z
- Hard to quantify uncertainty of predictions (unlike GPs)

Model-Agnostic Meta-Learning (MAML)

- Motivation: train models in such a way that are then able to quickly adapt to new tasks
- Requirement: task level loss, and fit via gradient descent

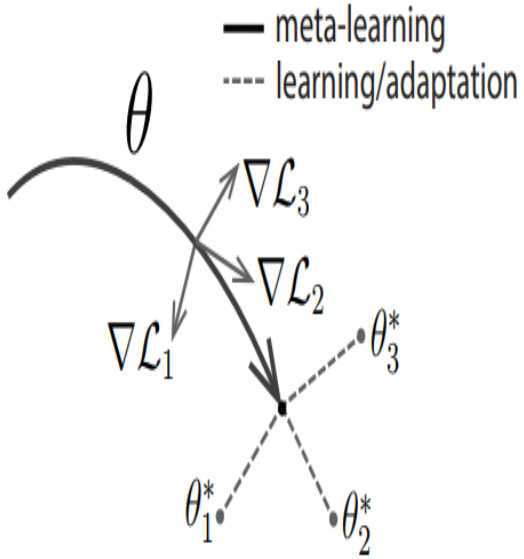


Figure 1. Diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

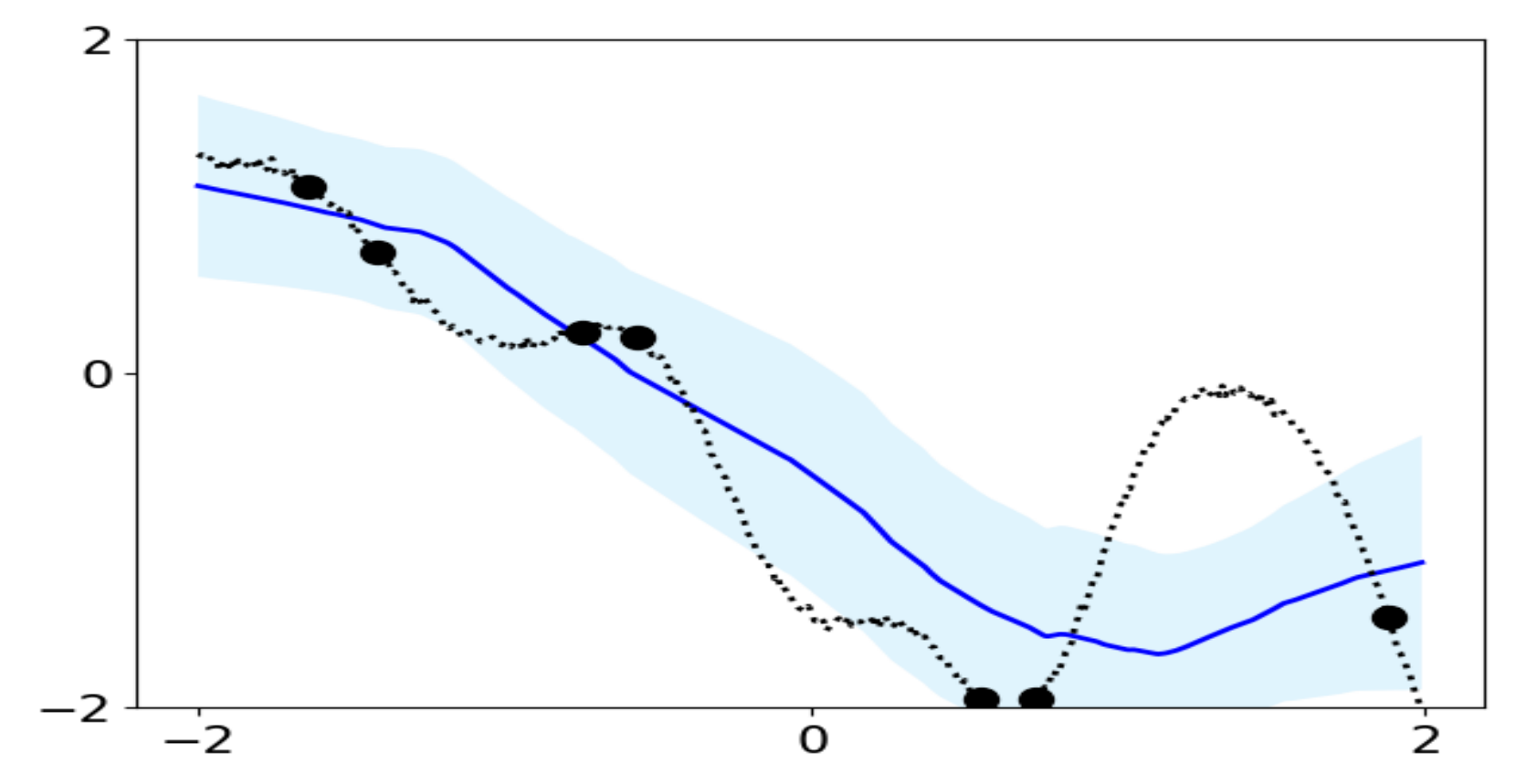
Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
11: **end while**

Finn et al. [2017]

MAML-CNP

- GP Data Generator:



Mean prediction $+/-$ 1 standard deviation

References

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami. Conditional neural processes. *arXiv preprint arXiv:1807.01613*, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

References ii

Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami. Neural Processes repository, 2019. URL `https://github.com/deepmind/neural-processes`.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

Kaspar Martens. Neural Processes as distributions over functions, 2018. URL `https://kasparmartens.rbind.io/post/np/`.