Transformer Neural Processes: Uncertainty-Aware Meta Learning Via Sequence Modeling

Tung Nguyen [1]   Aditya Grover [1]

Abstract

Neural Processes (NPs) are a popular class of approaches for meta-learning. Similar to Gaussian Processes (GPs), NPs define distributions over functions and can estimate uncertainty in their predictions. However, unlike GPs, NPs and their variants suffer from underfitting and often have intractable likelihoods, which limit their applications in sequential decision making. We propose Transformer Neural Processes (TNPs), a new member of the NP family that casts uncertainty-aware meta learning as a sequence modeling problem. We learn TNPs via an autoregressive likelihood-based objective and instantiate it with a novel transformer-based architecture. The model architecture respects the inductive biases inherent to the problem structure, such as invariance to the observed data points and equivariance to the unobserved points. We further investigate knobs within the TNP framework that tradeoff expressivity of the decoding distribution with extra computation. Empirically, we show that TNPs achieve state-of-the-art performance on various benchmark problems, outperforming all previous NP variants on meta regression, image completion, contextual multi-armed bandits, and Bayesian optimization.

1. Introduction

The goal of meta learning (Schmidhuber, 1987; Vanschoren, 2018) is to learn models that can adapt quickly to unseen tasks with only a few labeled examples. Since the amount of labeled data for any new task is limited, we ideally require the model to have both high accuracy and quantify the uncertainty in its predictions. This is particularly important within sequential decision making, e.g., for Bayesian optimization (Mockus et al., 1978; Schonlau et al., 1998; Shahriari et al., 2015; Hakhamaneshi et al., 2021) and multi-armed bandits (Cesa-Bianchi & Lugosi, 2006; Riquelme et al., 2018), where the quantified uncertainty can guide data acquisition. We call this paradigm uncertainty-aware meta learning, which is the main focus of our paper.

Neural Processes (NPs) (Garnelo et al., 2018a;b) are a rich class of models for such problems. NPs offer a flexible way to modeling distributions over functions, and are trained via a meta-learning framework that enables rapid adaptation to new functions at test time. More specifically, NPs are structured similar to a Variational Autoencoder (VAE) (Kingma & Welling, 2013) over datasets, wherein we employ a latent variable model to estimate the conditional distribution over the labels of the unlabeled (context) points given the set of labeled (target) points. By amortizing the functional uncertainty in the encoded latent variables, NPs can be quickly adapted to a new task at test-time.

However, NPs and their variants suffer from two major drawbacks. First, many NP variants often have intractable marginal likelihoods due to the presence of latent variables and instead optimize for a surrogate variational lower bound of the log-likelihood. The standard justification for using latent variables is that they can represent functional uncertainty and improve the predictive performance in some cases (Le et al., 2018). However, it is also known that optimizing variational lower bounds of the log-likelihood does not necessarily lead to a meaningful latent representation and can require non-trivial adjustments to the objective (Chen et al., 2016; Alemi et al., 2018). Second, it has been observed that NPs tend to underfit to the data distribution. Attentive Neural Processes (ANPs) (Kim et al., 2019) partly address this problem by incorporating attention mechanisms into the NP encoder-decoder architecture. While ANPs provide a considerably improved fit, these models tend to make overconfident predictions and have poor performance on sequential decision making problems.

We propose Transformer Neural Processes (TNPs), a new framework for uncertainty-aware meta learning derived from a sequence modeling perspective. The learning objective for TNPs is to autoregressively maximize the conditional log-likelihood of the target points (observed only during training) conditioned on the context points (observed

[1]Department of Computer Science, UCLA. Correspondence to: Tung Nguyen <tungnd@cs.ucla.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

during training and testing). This removes the need for latent variables and any variational approximations, while allowing for an expressive parameterization of the predictive distribution. We instantiate TNPs via a transformer-based architecture with a causal mask, similar to GPT-x models (Radford et al., 2018; 2019; Brown et al., 2020). Such models have led to state-of-the-art performance of a wide variety of domains and modalities (Brown et al., 2020; Lu et al., 2021; Chen et al., 2021). However, vanilla transformers cannot be directly used to parameterize a Neural Process as they lack invariance to the conditioned tokens and are not equivariant to the ordering of the target points. We propose suitable modifications to satisfy these desiderata by removing positional embeddings, using a novel padding and masking scheme for the context and target points, and considering a Monte Carlo approximation to a symmetrized predictive distribution that is equivariant by design.

Finally, we also propose two variants of TNPs, which can tradeoff the expressivity of the autoregressive factorization with computational tractability and exact equivariance. These variants consider diagonal and Cholesky approximations to the covariance matrix of the output distribution. Empirically, we evaluate TNPs on various benchmark problems proposed in prior works, including meta regression, image completion, contextual bandits, and Bayesian Optimization (BO). While meta regression and image completion evaluate the quality of predictions produced by TNPs, contextual bandits and BO directly measure the performance of TNPs on sequential decision making tasks. On all these problems, we observe that TNPs outperform several attention (Kim et al., 2019; Lee et al., 2020) and non-attention based variants (Garnelo et al., 2018a;b) of NPs by a large margin.

2. Background

2.1. Uncertainty-Aware Meta Learning

In meta learning, we assume an unknown distribution over functions, say $\mathcal{F}$. During training, we sample a fixed number of functions from $\mathcal{F}$ and observe a finite set of evaluations $\{x_i, y_i\}_{i=1}^N$ from each function $f : \mathcal{X} \to \mathcal{Y}$. At test time, we evaluate the generalization ability of the model on a set of unseen functions, assumed to be drawn from the same or similar distribution as $\mathcal{F}$. For each test function, we provide a small set of labelled training points D_{train}, and test the ability of the model to make predictions on a test set D_{test}.

Our focus in this work is on uncertainty-aware meta learning. That is, we assume that the model outputs a joint predictive distribution over the entire test set D_{test}. For example, if we assume the predictive distribution is a multi-variate Gaussian with a diagonal covariance, we output a mean μ_j and standard deviation σ_j for each input $x_j \in D_{\text{test}}$. Here, the σ_j's quantify the uncertainty in the model's predictions.

2.2. Neural Processes

A Neural Process (NP) is a stochastic process that describes the predictive distribution over a set of unlabelled points (target) given a training set of labelled points (context) (Garnelo et al., 2018a;b). Additionally, NPs incorporate a latent variable z to represent the functional uncertainty. Formally, the likelihood of the NP model is given as:

$$\begin{aligned} & p(y_{m+1:N} | x_{m+1:N}, C) \\ & = \int_z p(y_{m+1:N} | x_{m+1:N}, z) p(z|C) dz, \end{aligned} \tag{1}$$

in which $C = \{x_i, y_i\}_{i=1}^m$ and $T = \{x_i, y_i\}_{i=m+1}^N$ is the set of context and target pairs respectively. We note that there also exists variants of neural processes with no latent variables; we refer the reader to Section 5 for a detailed review. As the likelihood is intractable, NPs maximize an evidence lower bound (ELBO) of the log-likelihood instead:

$$\begin{aligned} & \log p(y_{m+1:N} | x_{m+1:N}, C) \geq \\ & \mathbb{E}_{q(z|C,T)}[\log p(y_{m+1:N} | x_{m+1:N}, z)] - \mathrm{KL}(q(z|C,T) || p(z|C)). \end{aligned} \tag{2}$$

Equivalently, we can view NPs as a VAE (Kingma & Welling, 2013) over the target labels conditioned on the context pairs and the target inputs. The encoder $q(z|C,T)$ is a permutation-invariant function which maps the context pairs to a distribution over z. In practice, q consists of an MLP that maps each pair (x_i, y_i) to its representation, an aggregator that combines these representations, and another MLP that outputs the mean and variance of z. The decoder $p(y_{m+1:N} | x_{m+1:N}, z) = \prod_{i=m+1}^N p(y_i | x_i, z)$ predicts the label for each target independently given the inferred z.

2.3. Transformers

Transformers were proposed by Vaswani et al. (2017) as an efficient architecture for modeling sequential data. Transformers are composed of an encoder and an decoder, which both consist of a stack of self-attention layers with residual connections. The self-attention layer receives n embeddings $\{e_i^{\text{in}}\}_{i=1}^n$, which are associated with n input tokens, and outputs n corresponding embeddings $\{e_i^{\text{out}}\}_{i=1}^n$. Each embedding vector e_i^{in} is first mapped to a key k_i, query q_i, and value v_i via linear transformations. The output embedding e_i^{out} is a weighted linear combination of all the input values, where the weights are given by the normalized dot-product between its query q_i and other keys k_j:

$$e_i^{\text{out}} = \sum_{j=1}^n \frac{\exp(\langle q_i, k_j \rangle)}{\sum_{j'=1}^n \exp(\langle q_i, k_{j'} \rangle)} \cdot v_j. \tag{3}$$

Before the first self-attention layer, each input token is passed through a positional encoder, which incorporates sequential information into the input sequence. This flexible architecture allows transformers to handle input sequences of arbitrary lengths, and provides a simple yet efficient way to model the relationships between input tokens. This architecture has been the key to many recent breakthroughs in language and vision (Radford et al., 2019; Dosovitskiy et al., 2020; Chen et al., 2020; Brown et al., 2020). In this paper, we study how transformers can be applied to uncertainty-aware meta-learning problems.

3. Transformer Neural Processes

We propose to solve uncertainty-aware meta learning via the lens of sequence modeling. In order to do so, we consider each set of evaluations $\{x_i, y_i\}_{i=1}^N$ that the model observes during training as an ordered sequence of N data points. Since we observe evaluations from multiple such functions, we segregate a random subset of each training sequence as the set of context pairs $(x_{1:m}, y_{1:m})$ as few-shot conditioning for the sequence model. Thereafter, we autoregressively model the predictive likelihood of the remaining $(N-m)$ target points and maximize the following objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_{1:N}, y_{1:N}, m} \left[\log p_\theta(y_{m+1:N} \mid x_{1:N}, y_{1:m})\right] \quad (4)$$

$$= \mathbb{E}_{x_{1:N}, y_{1:N}, m} \left[\sum_{i=m+1}^{N} \log p_\theta(y_i \mid x_{1:i}, y_{1:i-1}) \right]. \quad (5)$$

Each conditional in the above objective is a univariate Gaussian distribution. In practice, we optimize a Monte Carlo approximation of the objective in Eq. 5, where we consider a batch of training functions and their randomly sampled evaluations. We uniformly sample an index m that determines the context and target points for each set of evaluations. Next, we list the desiderata for the architectures that will be used for parameterizing the sequence model.

Property 3.1. ***Context invariance.*** *A model p_θ is context invariant if for any choice of permutation function π and $m \in [1, N-1]$, $p_\theta(y_{m+1:N} \mid x_{m+1:N}, x_{1:m}, y_{1:m}) = p_\theta(y_{m+1:N} \mid x_{m+1:N}, x_{\pi(1):\pi(m)}, y_{\pi(1):\pi(m)})$.*

Property 3.2. ***Target equivariance.*** *A model p_θ is target equivariant if for any choice of permutation function π and $m \in [1, N-1]$, $p_\theta(y_{m+1:N} \mid x_{m+1:N}, x_{1:m}, y_{1:m}) = p_\theta(y_{\pi(m+1):\pi(N)} \mid x_{\pi(m+1):\pi(N)}, x_{1:m}, y_{1:m})$.*

Context invariance (Property 3.1) requires that for any underlying function f, the predictions for the target points should not change if we permute the context points. Target equivariance (Property 3.2) requires that whenever we permute the target inputs $\{x_i\}_{i=m+1}^N$, the predictions are permuted accordingly. We instantiate the objective in Eq. 5 with a transformer architecture as shown in Figure 1. Standard transformers such as GPT (Radford et al., 2018) employ a casual mask to enforce the autoregressive structure needed for Eq. 5. However, this naive application of autoregressive transformers fails to satisfy the desiderata in Property 3.1 and Property 3.2. Since this model treats x_i and y_i as two separate tokens, we need to add them with a positional embedding vector for the model to associate them as a pair (x_i, y_i). This positional encoding, unfortunately, makes the model's output depend on the permutation of the target points. Further, autoregressive transformers also violate target equivariance as the ordering of points influences their embeddings and hence, different orderings lead to non-equivariant predictions. Next, we present Transformer Neural Processes (TNPs), a family of transformer-based neural processes that mitigates the aforementioned challenges.

3.1. Autoregressive Transformer Neural Process

Our first attempt at TNPs builds on the autoregressive factorization in Eq. 5. A vanilla transformer violates Property 3.1 due to its use of positional encodings. However, these encodings are also useful for drawing associations between x_i and y_i. To address this challenge, we first concatenate x_i and y_i to form a single token, which allows us to remove positional encodings and yet treat them as a pair. While this scheme works well for the context points $(x_i, y_i)_{i=1}^m$, we cannot apply it naively for any target point $y_{i>m}$ that depends on previous pairs $(x_j, y_j)_{j=1}^{i-1}$ and its input x_i. To respect the autoregressive structure for such points, we introduce auxiliary tokens consisting of $x_{i>m}$ padded with a dummy token (0 in our case) and append them to our original sequence. Our padded sequence consists of N real pairs $(x_i, y_i)_{i=1}^N$ and $N-m$ padded pairs $(x_i, 0)_{i=m+1}^N$:

$$\tau = \{(x_1, y_1), \ldots, (x_N, y_N), (x_{m+1}, 0), \ldots, (x_N, 0)\}. \quad (6)$$

To preserve the autoregressive ordering in (5), we design a masking mechanism in the attention layer such that:
(1) the context points $(x_i, y_i)_{i=1}^m$ only attend to themselves;
(2) the target point (x_i, y_i) for $(i > m)$ attends to all context points and the previous target points $(x_j, y_j)_{j=m+1}^i$;
(3) the padded target point $(x_i, 0)$ for $(i > m)$ attends to all context points and the previous target points $(x_j, y_j)_{j=m+1}^{i-1}$.
We refer to this model as Autoregressive Transformer Neural Processes (TNP-A). Figure 1 illustrates the model, and Figure 2 shows an example mask with $N = 5$ and $m = 2$. It satisfies Property 3.1 as a consequence of the masking scheme described above. For satisfying Property 3.2, we follow a symmetrization argument. Here, we note that any function can be made equivariant to a group by averaging the function evaluations over the entire group (Murphy et al., 2018). Formally, let us denote our joint distribution of interest as $\tilde{p}_\theta(y_{m+1:N} \mid x_{1:N}, y_{1:m})$. We define $\tilde{p}_\theta$ in terms of a base autoregressive model p_θ as:

$$\tilde{p}_\theta(y_{m+1:N} | x_{1:N}, y_{1:m})$$
$$= \mathbb{E}_\pi [p_\theta(y_{\pi(m+1):\pi(N)} | x_{\pi(m+1):\pi(N)}, x_{1:m}, y_{1:m})]. \quad (7)$$

Since the permutation group is intractable to enumerate, we instead consider a Monte Carlo average over randomly

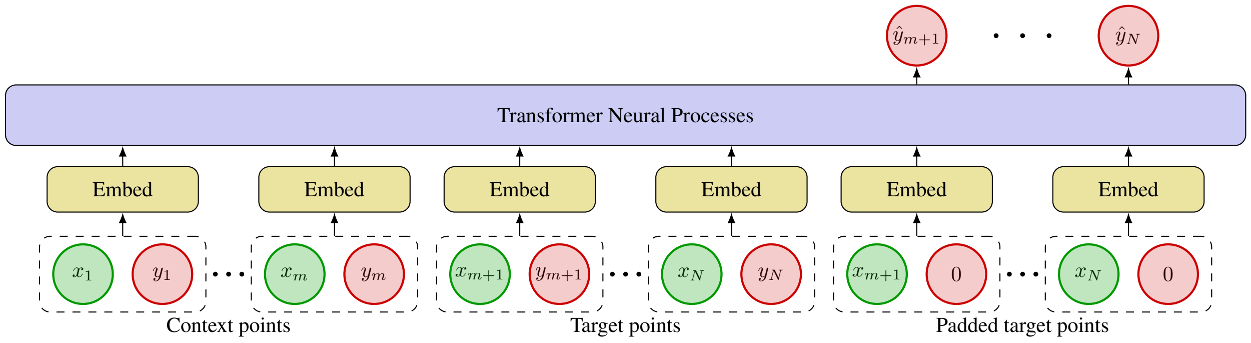


Figure 1. Illustration of the TNP-A architecture. The architecture specifies a custom masking pattern between the contexts, targets, and padded targets to respect autoregressive prediction order. For TNP-D and TNP-ND, we remove the targets from the input sequence.

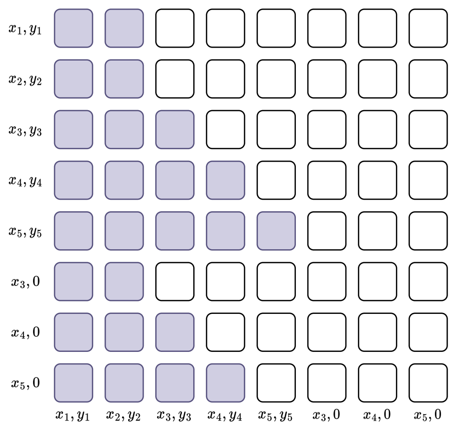


Figure 2. An example mask with $N = 5$ and $m = 2$. Each token is allowed to attend to other filled tokens on its corresponding row. The context points $(x_i, y_i)_{i=1}^2$ attend to themselves. Each target point (x_i, y_i) for $i > 2$ attends to the context points and the previous target points $(x_j, y_j)_{j=3}^i$. Each padded target point $(x_i, 0)$ for $i > 0$ attends to the context points and the previous target points $(x_j, y_j)_{j=3}^{i-1}$. This ensures the prediction for y_i $(i > 2)$ only depends on the context and the previous target points.

sampled permutations to approximate $\tilde{p}_\theta$. Even though the use of Monte Carlo implies that we satisfy Property 3.2 only in the limit, we can compute the predictive distribution tractably during training and evaluation. Next, we introduce alternate decoders for TNPs that exactly satisfy Property 3.2 while trading off expressivity for computational tractability.

3.2. Diagonal Transformer Neural Process

We can simplify the objective in (5) by assuming that the target points $y_{m+1:N}$ are conditionally independent given the context points and the inputs $x_{m+1:N}$. In other words, we consider the factorization:

$$p_\theta(y_{m+1:N}|x_{1:N}, y_{1:m}) = \prod_{i=m+1}^N p_\theta(y_i|x_i, x_{1:m}, y_{1:m}). \tag{8}$$

As the target points are independent, we remove $\{(x_i, y_i)_{i=m+1}^N\}$ from the input sequence in (6) and only feed the sequence consisting of the context points and the padded target points $\{(x_i, 0)_{i=m+1}^N\}$. This decoding distribution can be seen as a multivariate normal distribution with a diagonal covariance matrix and hence, we refer to it as Diagonal Transformer Neural Processes (TNP-D). Unlike TNP-A, we do not need to average over permutations to satisfy Property 3.2. However, for many scenarios, the independence assumption between the target points can be very strong for accurately modeling the underlying function.

3.3. Non-Diagonal Transformer Neural Process

Finally, we introduce Non-Diagonal Transformer Neural Processes (TNP-ND), the third variant of TNPs that balances between the tractability of TNP-D and the expressivity of TNP-A while satisfying both Property 3.1 and Property 3.2. We parameterize the decoding distribution as a multivariate normal distribution with a non-diagonal covariance matrix:

$$\begin{aligned} &p_\theta(y_{m+1:N} \mid x_{1:N}, y_{1:m}) \\ &= \mathcal{N}(y_{m+1:N} \mid \mu_\theta(x_{1:N}, y_{1:m}), \Sigma_\theta(x_{1:N}, y_{1:m})). \end{aligned} \tag{9}$$

Similar to TNP-D, we remove $\{(x_i, y_i)_{i=m+1}^N\}$ from the input sequence as the target points are predicted jointly. Since computing the full covariance matrix can be expensive, we consider two approximations to parameterize Σ:
(1) *Cholesky decomposition:* $\Sigma = LL^T$, where L is a lower triangular matrix with positive diagonal values; and
(2) *Low-rank approximation:* $\Sigma = \exp(D) + AA^T$, where D is a diagonal matrix and A is a low-rank matrix.
For the main experiments of this paper, we use the Cholesky decomposition due to its computational convenience. The results for the low-rank approximation are in Appendix A.1.

For a distribution with dimension n, we need a neural network that outputs $\frac{n(n+1)}{2}$ values to represent the lower triangular matrix L. In practice, the dimension of the distribution depends on the number of the target points, which varies during both training and evaluation. This means we cannot simply use a neural network to directly output L. We instead parameterize the decoder as follows. First, the output

Table 1. Comparison of TNPs with the baselines on log-likelihood of the target points on various GP kernels. We train each method with 5 different seeds and report the mean and standard deviation.

Method	RBF	Matérn 5/2	Periodic
CNP	0.26 ± 0.02	0.04 ± 0.02	-1.40 ± 0.02
CANP	0.79 ± 0.00	0.62 ± 0.00	-7.61 ± 0.16
NP	0.27 ± 0.01	0.07 ± 0.01	-1.15 ± 0.04
ANP	0.81 ± 0.00	0.63 ± 0.00	-5.02 ± 0.21
BNP	0.38 ± 0.02	0.18 ± 0.02	$\mathbf{-0.96 \pm 0.02}$
BANP	0.82 ± 0.01	0.66 ± 0.00	-3.09 ± 0.14
TNP-D	1.39 ± 0.00	0.95 ± 0.01	-3.53 ± 0.37
TNP-A	$\mathbf{1.63 \pm 0.00}$	$\mathbf{1.21 \pm 0.00}$	-2.26 ± 0.17
TNP-ND	1.46 ± 0.00	1.02 ± 0.00	-4.13 ± 0.33

vectors $z_{m+1:N}$ of the last masked self-attention layer are fed to an MLP that produces the mean values $\mu_{m+1:N}$. We then pass $z_{m+1:N}$ through another head consisting of an additional stack of self-attention layers, and a final projection layer (an MLP) that outputs $N-m$ vectors $h_{m+1:N}$. Each vector $h_i \in \mathbb{R}^p$, is projected to a dimension p. The lower triangular matrix L is then computed as:

$$L = \text{lower}(HH^\top),\ H \in \mathbb{R}^{n\times p}, \tag{10}$$

where H is the row-wise stack of $\{h_i\}_{i=m+1}^N$, lower(L) removes the upper triangular parts of L, and $n = N-m$ is the number of the target points. While this particular parameterization does not universally represent all the possible lower triangular matrices, it provides two main benefits. First, it allows us to parameterize a multivariate normal distribution with an arbitrary number of target points. Second, the space complexity of this parameterization is only $\mathcal{O}(n)$ compared to $\mathcal{O}(n^2)$ if we directly output the components of L.

4. Experiments

We evaluate Transformer Neural Processes (TNPs) on several tasks: regression, image completion, Bayesian optimization, and contextual bandits. This set of experiments has been used extensively to benchmark NP-based models in prior works (Garnelo et al., 2018b; Kim et al., 2019; Lee et al., 2020). We compare TNPs with other members of the NP family, namely Conditional Neural Processes (CNPs) (Garnelo et al., 2018a), Neural Processes (NPs) (Garnelo et al., 2018b), and Bootstrapping Neural Processes (BNPs) (Lee et al., 2020), as well as their attentive version (Kim et al., 2019), which are CANPs, ANPs, and BANPs, respectively. We have open-sourced the codebase for reproducing our experiments.[1] The implementation of the baselines is borrowed from the official implementation of BNPs.[2]

[1]https://github.com/tung-nd/TNP-pytorch
[2]https://github.com/juho-lee/bnp

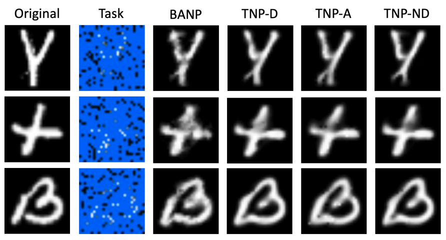


Figure 3. Completed images produced by the best baseline and TNPs from 100 context points. Original images are drawn randomly from EMNIST unseen classes. For stochastic models, we sample multiple times and average the results.

4.1. 1-D Regression

Problem: Given a set of context points $\{x_i, y_i\}_{i=1}^m$ that come from an unknown function f, we train the model to make predictions for a set of target points $\{x_i\}_{i=m+1}^N$ that come from the same function (Garnelo et al., 2018b).

Training: In each epoch of training, we draw B different functions from a Gaussian Process prior with an RBF kernel: $f_i \sim \mathcal{GP}(m,k)$, where $m(x) = 0$ and $k(x,x') = \sigma_f^2 \exp(-\frac{(x-x')^2}{2\ell^2})$. The hyperparameters of the GP (ℓ and σ_f) are randomized for each function, allowing the model to learn from a more diverse set of functions. For each f_i, we choose N random locations to evaluate, and sample an index m that splits the sequence to context and target points. For all methods, $\ell \sim \mathcal{U}[0.6, 1.0), \sigma_f \sim \mathcal{U}[0.1, 1.0)$, $B = 16$, $N \sim \mathcal{U}[6, 50)$, $m \sim \mathcal{U}[3, 47)$.

Evaluation: We test the trained models on unseen functions that are drawn from GPs with RBF, Matérn 5/2 and Periodic kernels. The number of evaluation points N and the number of context points m are generated from the same uniform distribution as in training. The evaluation set contains 48000 functions for each kernel. We evaluate all methods on log-likelihood of the target points. We refer the readers to Appendix B.1 for more evaluation metrics.

Results: Table 1 shows that TNPs outperform the other methods on 2/3 kernels by a large margin. Even though TNPs underperform on regressing functions from the periodic kernel, we show later in Section 4.4 that they can optimize the same class of functions better than the baselines. As expected, among three variants of TNPs, TNP-A achieves the best likelihood due to the use of an autoregressive decoder, followed by TNP-ND, and finally TNP-D.

4.2. Image completion

Problem: The model observes a subset of pixel values of an image and completes the rest. This can be cast as a 2-D meta-regression problem, in which x denotes the coordinates of a pixel, and y denotes the corresponding pixel value. Each image can be thought of as a unique function that maps from coordinate to pixel intensity (Garnelo et al., 2018b).

Table 2. Comparison of TNPs with the baselines on log-likelihood of the target points on two datasets: EMNIST (left) and CelebA (right). We train each method with 5 different seeds and report the mean and standard deviation.

Method	CelebA
CNP	2.15 ± 0.01
CANP	2.66 ± 0.01
NP	2.48 ± 0.02
ANP	2.90 ± 0.00
BNP	2.76 ± 0.01
BANP	3.09 ± 0.00
TNP-D	3.89 ± 0.01
TNP-A	$\mathbf{5.82 \pm 0.01}$
TNP-ND	5.48 ± 0.02

Method	EMNIST	
	Seen classes (0-9)	Unseen classes (10-46)
CNP	0.73 ± 0.00	0.49 ± 0.01
CANP	0.94 ± 0.01	0.82 ± 0.01
NP	0.79 ± 0.01	0.59 ± 0.01
ANP	0.98 ± 0.00	0.89 ± 0.00
BNP	0.88 ± 0.01	0.73 ± 0.01
BANP	1.01 ± 0.00	0.94 ± 0.00
TNP-D	1.46 ± 0.01	1.31 ± 0.00
TNP-A	$\mathbf{1.54 \pm 0.01}$	$\mathbf{1.41 \pm 0.01}$
TNP-ND	1.50 ± 0.00	1.31 ± 0.00

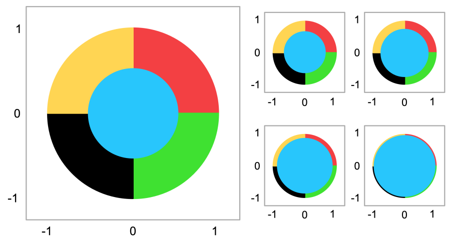


Figure 4. The wheel bandit problem with varying values of δ.

Training: We use two datasets for this experiment: EMNIST (Cohen et al., 2017) and CelebA (Liu et al., 2018). EMNIST contains black and white images of handwritten letters, and CelebA contains colored images of celebrity faces. We down-sample each image to 32×32. For EMNIST, we only use 10 classes for training. Similar to the 1-D regression experiment, we randomly select subsets of pixels as context points and target points. For both datasets, $N \sim \mathcal{U}[6, 200), m \sim \mathcal{U}[3, 197)$. The x values are rescaled to $[-1, 1]$ and the y values are rescaled to $[-0.5, 0.5]$.

Evaluation: We evaluate each method on log-likelihood of the target points on held-out datasets. The number of pixels and the number of context points are generated from the same uniform distribution as in training.

Results: Table 2 show significant improvements of TNPs over the baselines. Similar to the 1-D regression, TNP-A achieves the best likelihood, followed by TNP-ND and TNP-D. For EMNIST, the performance of TNPs on unseen classes is only slightly worse than on seen classes, indicating better generalization compared to the other methods. Figure 3 shows that TNPs produce noticeably better completed images than the best baseline. We refer the readers to Appendix B.2 for different samples produced by TNPs.

4.3. Contextual bandits

Problem: We compare TNPs with the baselines on the wheel bandit problem introduced in Riquelme et al. (2018) (Figure 4). In this problem, a unit circle is divided into a low-reward region (blue area) and four high-reward regions (the other four coloured areas). A scalar δ determines the size of the low-reward region, and other regions have equal sizes. The agent does not know the underlying δ, and has to choose among $k = 5$ arms given its coordinates $X = (X_1, X_2)$ within the circle. If $||X|| \leq \delta$, the agent falls within the low-reward region (blue). In this case the optimal action is $k = 1$, which provides a reward $r \sim \mathcal{N}(1.2, 0.012)$, while all other actions only return $r \sim \mathcal{N}(1.0, 0.012)$. If the agent falls within any of the four high-reward region ($||X|| > \delta$), the optimal arm will be one of the remaining four $k = 2-5$, depending on the specific area. Pulling the optimal arm here results in a high reward $r \sim \mathcal{N}(50.0, 0.012)$, and as before all other arms receive $\mathcal{N}(1.0, 0.012)$ except for arm $k = 1$ which always returns $\mathcal{N}(1.2, 0.012)$.

Training: We sample a dataset of B different wheel problems $\{\delta_i\}_{i=1}^{B}$, which are drawn from a uniform distribution $\delta \sim \mathcal{U}(0, 1)$. For each problem, we sample N points to evaluate and pick m points as context, in which each point is a tuple (X, r) of the coordinates X and the corresponding reward values r of all 5 arms. The training objective is to regress the reward values from the coordinates. We set $B = 8, N = 562, m = 512$ in our experiments.

Evaluation: We test TNPs and the baselines on problems with varying δ values. We run with 50 different seeds for each value of δ, and each run consists of 2000 steps. In each step, the agent predicts the reward values of 5 arms based on the coordinates X, chooses an arm according to the Upper Confidence Bound (UCB) algorithm and receives the ground-truth reward value for the chosen arm. We use cumulative regret as the evaluation metric. See Appendix B.3 for simple regret results.

Results: Table 3 shows that TNPs outperform all baselines by a large margin on all settings, especially for harder problems (higher values of δ). Moreover, the performance of Transformers only slightly drops when the difficulty in-

Table 3. Comparison of TNPs with the baselines on cumulative regret on contextual bandit problems with different values of δ. We run each model 50 times for each value of δ and report the mean and standard deviation.

Method	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$	$\delta = 0.995$	$\delta = 0.999$	Average
Uniform	100.00 ± 1.18	100.00 ± 3.03	100.00 ± 4.16	100.00 ± 7.52	100.00 ± 8.11	100.00 ± 7.96	100.00 ± 5.97
CNP	4.08 ± 0.29	8.14 ± 0.33	8.01 ± 0.40	26.78 ± 0.85	38.25 ± 1.01	93.17 ± 2.81	29.74 ± 30.85
CANP	8.08 ± 9.93	11.69 ± 11.96	24.49 ± 13.25	47.33 ± 20.49	49.59 ± 17.87	33.29 ± 5.05	29.08 ± 21.28
NP	1.56 ± 0.13	2.96 ± 0.28	4.24 ± 0.22	18.00 ± 0.42	25.53 ± 0.18	62.73 ± 1.49	19.17 ± 21.36
ANP	1.62 ± 0.16	4.05 ± 0.31	5.39 ± 0.50	19.57 ± 0.67	27.65 ± 0.95	73.36 ± 5.95	21.94 ± 24.92
BNP	62.51 ± 1.07	57.49 ± 2.13	58.22 ± 2.27	58.91 ± 3.77	62.50 ± 4.85	77.46 ± 6.18	62.85 ± 7.78
BANP	4.23 ± 16.58	12.42 ± 29.58	31.10 ± 36.10	52.59 ± 18.11	49.55 ± 14.52	45.45 ± 11.71	32.56 ± 29.43
TNP-D	$\mathbf{1.18 \pm 0.94}$	1.70 ± 0.41	2.55 ± 0.43	$\mathbf{3.57 \pm 1.22}$	$\mathbf{4.68 \pm 1.09}$	9.56 ± 0.44	$\mathbf{3.87 \pm 2.91}$
TNP-A	3.67 ± 4.88	4.04 ± 2.38	4.29 ± 2.36	5.79 ± 5.27	9.29 ± 7.62	$\mathbf{6.13 \pm 2.50}$	5.54 ± 4.98
TNP-ND	1.76 ± 0.61	$\mathbf{1.41 \pm 0.98}$	$\mathbf{1.61 \pm 1.65}$	4.98 ± 2.84	7.22 ± 3.28	13.66 ± 2.92	5.11 ± 4.94

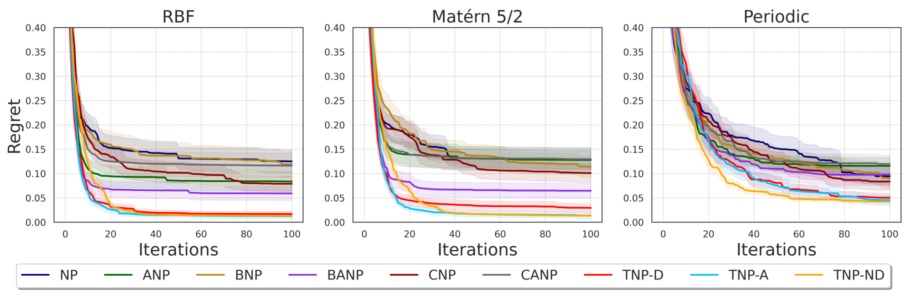


Figure 5. Regret performance on 1D BO tasks. For each kernel, we generate 100 functions and report the mean and standard deviation.

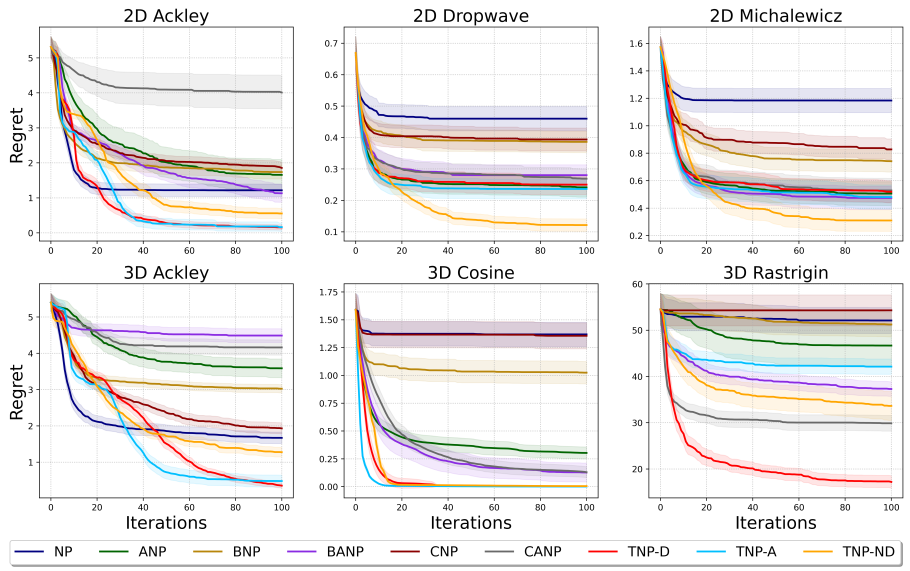


Figure 6. Regret performance on 2D and 3D BO tasks. For each function, we run BO for 100 times with different seeds and report the mean and standard deviation.

creases, which is a significant improvement over the baselines, as these methods barely work for hard problems. We also note that for other NP-based methods, using attention hurts the performance. This can be clearly seen from the table, as CANPs, ANPs and BANPs are inferior to the non-attentive counterparts CNPs, NPs, and BNPs, respectively.

4.4. Bayesian Optimization

Problem: The goal of Bayesian optimization (BO) (Frazier, 2018) is to optimize a black-box function $f(x)$ that we can evaluate but do not have access to the gradient information. The BO process runs in a loop, where in each iteration, we use a surrogate function to estimate the unknown function f, and an acquisition function to choose the next point to evaluate based on the estimation. We use TNPs and the baselines as the surrogate function, and UCB as the acquisition function. We consider both one-dimensional (1D) and multi-dimensional (2D and 3D) settings.

Training: Training in the one-dimensional setting is identical to training in Section 4.1. When x is multi-dimensional, we use GPytorch (Gardner et al., 2018) to generate training data from multivariate GPs with RBF kernel. We use $N \sim \mathcal{U}[60, 128), m \sim \mathcal{U}[30, 98)$ for the 2D setting, and $N \sim \mathcal{U}[128, 256), m \sim \mathcal{U}[64, 192)$ for the 3D setting.

Evaluation: In the 1D setting, the objective functions are generated from GPs with RBF, Matérn 5/2, and Periodic kernels. In the multi-dimensional setting, we use various benchmark functions in the optimization literature (Kim & Choi, 2017; Kim, 2020), and Botorch (Balandat et al., 2020) is used as the implementation of the overall BO process (e.g., optimization and acquisition function). For each objective function, we run BO for 100 iterations, and simple regret is used as the evaluation metric.

Results: Figure 5 shows that TNPs outperform the baselines significantly in all three kernels. This includes the Periodic kernel, where TNPs were outperformed by other methods for meta regression (Table 1), suggesting that simple meta regression metrics may not be sufficient for model selection in online decision making. Figure 6 shows the results for the multi-dimensional setting. All three TNP variants outperform the baselines in at least $3/6$ tasks: 2D Ackley, 3D Ackley, and 3D Cosine, and achieve competitive results to the best baseline in 2D Dropwave and 2D Michalewicz. TNP-ND is the best variant in this setting, which beats the baselines in $5/6$ tasks, followed by TNP-D, which outperforms the baselines in $4/6$ tasks. TNP-A also performs well and is competitive on all but the 3D Rastrigin task.

4.5. Memory and time complexity of TNPs

The major successes of Transformers in language (Brown et al., 2020) and vision (Dosovitskiy et al., 2020) largely attribute to the ability to scale to billions of parameters, thus having unprecedented results. Similarly, one can claim that the superior performance of TNPs is merely due to having more parameters than the baselines. Therefore, we compare the parameter counts, training and prediction time of TNPs and the baselines in Table 4. All models are comparable, barring TNP-A which as expected, incurs a higher prediction time because of the autoregressive nature. This indicates that scaling is not all we need, but the transformer architecture itself is important too.

Table 4. Number of parameters, training time and prediction time of TNPs and the baselines. We measure run time on the 1-D regression task on an RTX2080Ti, with 1000 batches of size 16.

Method	# Parameters	Training (s)	Prediction (s)
CNP	215682	11.20	0.76
CANP	331906	20.05	1.64
NP	232194	16.79	1.24
ANP	348418	21.07	2.15
BNP	248450	20.07	4.86
BANP	364674	24.07	16.72
TNP-A	222082	20.14	337.09
TNP-D	222082	20.13	2.75
TNP-ND	332821	26.89	4.05

5. Related Work

Neural Processes. The Conditional Neural Process (CNP) (Garnelo et al., 2018a) was the first member of the NP family. CNPs encode the context points to a deterministic latent vector z and hence do not have a notion of functional uncertainty. The Neural Process (NP) (Garnelo et al., 2018b) was proposed to address these shortcomings by introducing stochasticity in the form of latent variables. Le et al. (2018) made a careful empirical examination of different objectives and hyperparameter choices when training a Neural Process. Since then, many extensions have been proposed which improve NPs by building translation equivariance (Gordon et al., 2019), incorporating attention to address underfitting (Kim et al., 2019), using bootstrapping to overcome the limitations of Gaussian assumption on the latent variables (Lee et al., 2020), and modeling predictive correlations (Bruinsma et al., 2021). In addition, many works have also extended NPs to a wider range of problems, which include modeling a sequence of stochastic processes (Singh et al., 2019) and modeling stochastic physics fields (Holderrieth et al., 2021). Mathieu et al. (2021) recently proposed a contrastive learning framework that replaces the reconstruction objective in NP to learn a better representation. In Xu et al. (2020), the authors proposed a meta-learner based on CANP that achieved good results on large scale image classification. Grover et al. provide a series of fine-grained diagnostics for the uncertainty modeled via various neural processes. Finally, Galashov et al. (2019) empirically studied the effectiveness of NPs as a meta-learner in various sequential decision making problems.

Transformers. The transformer architecture was introduced by Vaswani et al. (2017) for flexible modeling of natural language. In recent years, transformers have made breakthroughs in language (Devlin et al., 2018; Radford et al., 2018; 2019; Brown et al., 2020) and vision (Dosovitskiy et al., 2020; Radford et al., 2021; Chen et al., 2020).

Recent works (Lee et al., 2019; Kossen et al., 2021) have also applied transformers to modeling the relationship between different data points, in addition to modeling interactions between components of the same data point. Set Transformers (Lee et al., 2019) are closely related to TNPs, and were proposed to solve set-input problems, such as finding the maximum number in a set or counting unique characters. Set Transformers adopt the transformer architecture to model the interactions between items of the same set and remove positional encodings for permutation invariance. However, our focus in this work is on uncertainty-aware meta learning. For such tasks, TNPs incorporate important architectural design choices such as the input representation and masking mechanism. It is crucial for TNPs to model the interaction between the inputs $x's$ as well as the relationship between x and y to accurately infer the underlying function. Finally, Chen et al. (2021) proposed an autoregressive transformer based model for offline reinforcement learning, which was extended to the online setting by Zheng et al. (2022). Besides differences in the agent setup, this work considers trajectories from a single task, unlike our focus on meta-learning.

6. Discussion

Neural Processes offer a promising approach for learning flexible stochastic processes directly from data. However, it is unclear which aspects of their design are essential for downstream applications in uncertainty-aware meta learning, such as regression and sequential decision making. Popular claims, such as the use of latent variables for representing functional uncertainty and diverse sampling, have mixed empirical evidence (Garnelo et al., 2018b) and a deterministic path between the encoder and decoder is important for good performance (Le et al., 2018). Moreover, the standard evidence lower bounds for variational autoencoders (including NPs) can completely ignore the latent code with powerful decoders (Chen et al., 2016; Alemi et al., 2018).

In this regard, we proposed Transformer Neural Processes (TNPs), an alternative framing of uncertainty-aware meta learning via sequence modeling. TNPs optimize an autoregressive modeling objective and benefit from the use of a transformer backbone (Vaswani et al., 2017; Radford et al., 2018). While attention mechanisms have also previously been used for parameterizing NPs (Kim et al., 2019), we showed that replacing the entire architecture stack can drastically improve performance across various benchmark tasks. We also showed these empirical benefits cannot be attributed solely to the use of a more expressive decoding distribution as in TNP-A, but can also be obtained to a good degree via tractable and equivariant (but relatively less expressive) parameterizations such as TNP-D and TNP-ND.

In the future, we are keen to further build on the merits of the TNP architecture for scaling to high-dimensional problems beyond current benchmarks. We are also interested in pursuing future work towards a clean separation of functional and point uncertainties (as in GPs), potentially via recent advances in stochastic transformers (Lin et al., 2020).

Acknowledgements

We would like to thank Hritik Bansal, Shashank Goel, Siddarth Krishnamoorthy, Satvik Mashkaria, and Tuan Pham for the insightful discussions during the early development of the paper. We also want to thank the IDRE's Research Technology group for the GPU computing resources on the UCLA Hoffman2 Cluster.

References

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. Fixing a broken elbo. In *International Conference on Machine Learning*, pp. 159–168. PMLR, 2018.

Balandat, M., Karrer, B., Jiang, D., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33, 2020.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Bruinsma, W. P., Requeima, J., Foong, A. Y., Gordon, J., and Turner, R. E. The gaussian neural process. *arXiv preprint arXiv:2101.03606*, 2021.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, learning, and games*. Cambridge university press, 2006.

Chan, S. C., Santoro, A., Lampinen, A. K., Wang, J. X., Singh, A., Richemond, P. H., McClelland, J., DeepMind, S., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *CoRR*, 2022.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. Generative pretraining from pixels. In *International Conference on Machine Learning*, pp. 1691–1703. PMLR, 2020.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Chung, Y., Char, I., Guo, H., Schneider, J., and Neiswanger, W. Uncertainty toolbox: an open-source library for assessing, visualizing, and improving uncertainty quantification. *arXiv preprint arXiv:2109.10254*, 2021.

Cohen, G., Afshar, S., Tapson, J., and Van Schaik, A. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Foong, A. Y., Bruinsma, W. P., Gordon, J., Dubois, Y., Requeima, J., and Turner, R. E. Meta-learning stationary stochastic process prediction with convolutional neural processes. *arXiv preprint arXiv:2007.01332*, 2020.

Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Galashov, A., Schwarz, J., Kim, H., Garnelo, M., Saxton, D., Kohli, P., Eslami, S., and Teh, Y. W. Meta-learning surrogate models for sequential decision making. *arXiv preprint arXiv:1903.11907*, 2019.

Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *arXiv preprint arXiv:1809.11165*, 2018.

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

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

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

Grover, A., Tran, D., Shu, R., Poole, B., and Murphy, K. Probing uncertainty estimates of neural processes.

Hakhamaneshi, K., Abbeel, P., Stojanovic, V., and Grover, A. Jumbo: Scalable multi-task bayesian optimization using offline data. *arXiv preprint arXiv:2106.00942*, 2021.

Holderrieth, P., Hutchinson, M. J., and Teh, Y. W. Equivariant learning of stochastic fields: Gaussian processes and steerable conditional neural processes. In *International Conference on Machine Learning*, pp. 4297–4307. PMLR, 2021.

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

Kim, J. Benchmark functions for bayesian optimization. https://github.com/jungtaekkim/bayeso-benchmarks, 2020.

Kim, J. and Choi, S. BayesO: A Bayesian optimization framework in Python. https://bayeso.org, 2017.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., and Gal, Y. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *arXiv preprint arXiv:2106.02584*, 2021.

Le, T. A., Kim, H., Garnelo, M., Rosenbaum, D., Schwarz, J., and Teh, Y. W. Empirical evaluation of neural process objectives. In *NeurIPS workshop on Bayesian Deep Learning*, 2018.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753. PMLR, 2019.

Lee, J., Lee, Y., Kim, J., Yang, E., Hwang, S. J., and Teh, Y. W. Bootstrapping neural processes. *arXiv preprint arXiv:2008.02956*, 2020.

Lin, Z., Winata, G. I., Xu, P., Liu, Z., and Fung, P. Variational transformers for diverse response generation. *arXiv preprint arXiv:2003.12738*, 2020.

Liu, Z., Luo, P., Wang, X., and Tang, X. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15 (2018):11, 2018.

Lu, K., Grover, A., Abbeel, P., and Mordatch, I. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021.

Mathieu, E., Foster, A., and Teh, Y. W. On contrastive representations of stochastic processes. *arXiv preprint arXiv:2106.10052*, 2021.

Mockus, J., Tiesis, V., and Zilinskas, A. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding with unsupervised learning. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

Riquelme, C., Tucker, G., and Snoek, J. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Schonlau, M., Welch, W. J., and Jones, D. R. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, pp. 11–25, 1998.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104 (1):148–175, 2015.

Singh, G., Yoon, J., Son, Y., and Ahn, S. Sequential neural processes. *arXiv preprint arXiv:1906.10264*, 2019.

Vanschoren, J. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Xu, J., Ton, J.-F., Kim, H., Kosiorek, A., and Teh, Y. W. Metafun: Meta-learning with iterative functional updates. In *International Conference on Machine Learning*, pp. 10617–10627. PMLR, 2020.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer. In *ICML*, 2022.

Supplementary Materials to Transformer Neural Processes

A. Implementation details

A.1. Non-Diagonal Transformer Neural Processes

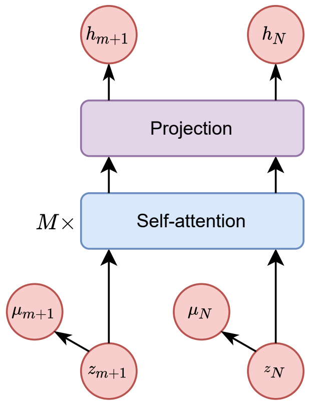


Figure 7. The decoder architecture of TNP-ND.

Figure 7 depicts the decoder architecture of TNP-NP presented in section 3.3, in which $\{z_i\}_{i=m+1}^{N}$ are the output of the last transformer decoder. Given $\{h_i\}_{i=m+1}^{N}$, the lower triangular matrix of the Multivariate Normal distribution is computed as:

$$L = \text{lower}(HH^\top),\ H \in \mathbb{R}^{n \times p}, \tag{11}$$

in which lower(L) removes the upper triangular parts of L, and $n = N - m$ is the number of the target points.

A.2. Contextual bandits

There is a crucial problem mismatch between training and evaluation. In training, we are given the reward values for all the arms, while during test time, we can only observe the reward value for the arm that we actually select, and the reward for all other arms will be drawn from a standard Gaussian distribution. To alleviate this problem, during training of TNPs, we randomly drop reward values for some arms of the context points and make the model to regress the ground-truth of those arms. This closes the gap between training and testing. We note that we also tried using this trick for other NP variants but it did not improve the performance.

A.3. Hyperparameters

In this section we present the hyperparameters that we used to train TNPs in the experiments. For the baselines, we use the hyperparameters reported in Lee et al. (2020).

1-D regression

- Model dimension: 64
- Number of embeddings layers: 4
- Feed forward dimension: 128
- Number of attention heads: 4

- Number of transformer layers: 6
- (TNP-ND) Number of self-attention layers on $\{z_i\}_{i=m+1}^{N}$: 2
- (TNP-ND) Projection dimension: 20
- (TNP-ND) Number of projection layers: 4
- Dropout: 0.0
- Number of training steps: 100000
- Learning rate: $5e^{-4}$ with Cosine annealing scheduler

Image completion

- Model dimension: 64
- Number of embeddings layers: 4
- Feed forward dimension: 128
- Number of attention heads: 4
- Number of transformer layers: 6
- (TNP-ND) Number of self-attention layers on $\{z_i\}_{i=m+1}^{N}$: 2
- (TNP-ND) Projection dimension: 20
- (TNP-ND) Number of projection layers: 4
- Dropout: 0.0
- Batch size: 100
- Number of training epochs: 200
- Learning rate: $5e^{-4}$ with Cosine annealing scheduler

Contextual bandits

- Model dimension: 16
- Number of embeddings layers: 3
- Feed forward dimension: 64
- Number of attention heads: 1
- Number of transformer layers: 4
- (TNP-ND) Number of self-attention layers on $\{z_i\}_{i=m+1}^{N}$: 2
- (TNP-ND) Projection dimension: 20
- (TNP-ND) Number of projection layers: 4
- Dropout: 0.0
- Number of training epochs: 100000
- Learning rate: $5e^{-4}$ with Cosine annealing scheduler
- Drop rate of reward values during training: 0.5

Bayesian optimization

- Model dimension: 64
- Number of embeddings layers: 4
- Feed forward dimension: 128
- Number of attention heads: 8
- Number of transformer layers: 6
- (TNP-ND) Number of self-attention layers on $\{z_i\}_{i=m+1}^{N}$: 2
- (TNP-ND) Projection dimension: 20
- (TNP-ND) Number of projection layers: 4
- Dropout: 0.0
- Number of training steps: 100000
- Learning rate: $5e^{-4}$ with Cosine annealing scheduler

B. Additional results

B.1. 1-D regression with additional metrics

We compare TNPs and the baselines on various metrics, which include root-mean-square error, calibration error, and log-likelihood. The metrics are all computed on the predictions of the target points by using the Uncertainty Toolbox (Chung et al., 2021).

Table 5 shows that three variants of TNPs outperform the baselines on all metrics in $2/3$ kernels. There is an interesting pattern among the NP-based methods: while using attention nearly always leads to better accuracy (CANPs, ANPs, BANPs versus CNPs, NPs, BNPs), it often results in poorer calibration. TNPs have the best of both worlds, since they are competitively accurate and calibrated in most tasks. The three variants achieve similar accuracy and degree of calibration, while TNP-A is the best variant with respect to log-likelihood metric.

Table 5. Comparison of TNPs with the baselines on various GP kernels and evaluation metrics. We train 5 instances with different seeds for each method and report the mean and std.

Metric	Method	RBF	Matérn 5/2	Periodic
RMSE	CNP	0.278 ± 0.003	0.310 ± 0.003	0.652 ± 0.001
	CANP	0.193 ± 0.000	0.228 ± 0.000	0.699 ± 0.002
	NP	0.282 ± 0.003	0.315 ± 0.003	0.650 ± 0.002
	ANP	0.193 ± 0.001	0.230 ± 0.000	0.703 ± 0.002
	BNP	0.269 ± 0.003	0.301 ± 0.003	$\mathbf{0.649 \pm 0.002}$
	BANP	0.192 ± 0.001	0.228 ± 0.001	0.701 ± 0.008
	TNP-D	$\mathbf{0.177 \pm 0.001}$	$\mathbf{0.222 \pm 0.000}$	0.664 ± 0.014
	TNP-A	0.178 ± 0.000	$\mathbf{0.222 \pm 0.000}$	0.660 ± 0.002
	TNP-ND	0.180 ± 0.001	0.223 ± 0.000	0.670 ± 0.009
CE	CNP	0.078 ± 0.002	0.051 ± 0.000	0.143 ± 0.002
	CANP	0.232 ± 0.001	0.165 ± 0.000	0.255 ± 0.004
	NP	0.093 ± 0.002	0.056 ± 0.001	0.130 ± 0.007
	ANP	0.235 ± 0.001	0.169 ± 0.001	0.265 ± 0.002
	BNP	0.093 ± 0.003	0.054 ± 0.002	$\mathbf{0.115 \pm 0.004}$
	BANP	0.236 ± 0.002	0.171 ± 0.002	0.217 ± 0.005
	TNP-D	$\mathbf{0.043 \pm 0.000}$	0.045 ± 0.000	0.129 ± 0.012
	TNP-A	0.045 ± 0.000	$\mathbf{0.044 \pm 0.000}$	0.119 ± 0.008
	TNP-ND	0.048 ± 0.001	0.050 ± 0.001	0.155 ± 0.009
Log-Likelihood	CNP	0.26 ± 0.02	0.04 ± 0.02	-1.40 ± 0.02
	CANP	0.79 ± 0.00	0.62 ± 0.00	-7.61 ± 0.16
	NP	0.27 ± 0.01	0.07 ± 0.01	-1.15 ± 0.04
	ANP	0.81 ± 0.00	0.63 ± 0.00	-5.02 ± 0.21
	BNP	0.38 ± 0.02	0.18 ± 0.02	$\mathbf{-0.96 \pm 0.02}$
	BANP	0.82 ± 0.01	0.66 ± 0.00	-3.09 ± 0.14
	TNP-D	1.39 ± 0.00	0.95 ± 0.01	-3.53 ± 0.37
	TNP-A	$\mathbf{1.63 \pm 0.00}$	$\mathbf{1.21 \pm 0.00}$	-2.26 ± 0.17
	TNP-ND	1.46 ± 0.00	1.02 ± 0.00	-4.13 ± 0.33

B.2. Image completion

Additional metrics We compare TNPs and the baselines on the metrics used in Section B.1, except for the Calibration error which took too much time to compute. Tables 6 and 7 show the results for CelebA and EMNIST, respectively. It is clear that three variants of TNPs outperform the baselines on both datasets on all 3 evaluation metrics. While three variants are similar in terms of accuracy and sharpness, TNP-A is the best variant with respect to log-likelihood.

Table 6. Comparison of TNPs vs the baselines on CelebA dataset with various evaluation metrics. We train 5 instances with different seeds for each method and report the mean and std.

Method	RMSE	Log-likelihood
CNP	0.137 ± 0.000	2.148 ± 0.008
CANP	0.116 ± 0.001	2.657 ± 0.010
NP	0.138 ± 0.001	2.480 ± 0.018
ANP	0.119 ± 0.000	2.904 ± 0.003
BNP	0.134 ± 0.000	2.764 ± 0.006
BANP	0.119 ± 0.000	3.087 ± 0.004
TNP-D	0.112 ± 0.000	3.891 ± 0.006
TNP-A	0.115 ± 0.000	$\mathbf{5.818 \pm 0.011}$
TNP-ND	$\mathbf{0.111 \pm 0.000}$	5.477 ± 0.016

Table 7. Comparison of TNPs vs the baselines on EMNIST dataset with various evaluation metrics. We train 5 instances with different seeds for each method and report the mean and std. We evaluate on both seen and unseen classes.

Setting	Method	RMSE	Log-likelihood
Seen classes (0-9)	CNP	0.184 ± 0.001	0.733 ± 0.004
	CANP	0.140 ± 0.002	0.935 ± 0.008
	NP	0.185 ± 0.003	0.794 ± 0.010
	ANP	0.142 ± 0.000	0.984 ± 0.003
	BNP	0.178 ± 0.002	0.876 ± 0.007
	BANP	0.142 ± 0.001	1.008 ± 0.003
	TNP-D	0.119 ± 0.002	1.461 ± 0.010
	TNP-A	0.122 ± 0.001	$\mathbf{1.537 \pm 0.005}$
	TNP-ND	$\mathbf{0.116 \pm 0.000}$	1.497 ± 0.002
Unseen classes (10-46)	CNP	0.225 ± 0.002	0.491 ± 0.010
	CANP	0.163 ± 0.002	0.823 ± 0.010
	NP	0.228 ± 0.003	0.591 ± 0.009
	ANP	0.166 ± 0.001	0.887 ± 0.004
	BNP	0.219 ± 0.003	0.728 ± 0.008
	BANP	0.161 ± 0.001	0.943 ± 0.003
	TNP-D	$\mathbf{0.139 \pm 0.001}$	1.308 ± 0.003
	TNP-A	0.142 ± 0.001	$\mathbf{1.413 \pm 0.005}$
	TNP-ND	0.140 ± 0.001	1.314 ± 0.004

Evaluation on full-image completion In Tables 2, 6, and 7, we computed the metrics on a target set, which is a subset of the entire image. We additionally report the performance of TNP-D and BANP (the strongest baseline) when the target is the entire image in Table 8, which shows that TNPs achieve a similarly better performance.

Table 8. Comparison of TNP-D and BANP on EMNIST unseen classes. We use 100 context points, and the target is the entire image.

Method	RMSE	Log-likelihood
BANP	0.144 ± 0.001	1.005 ± 0.003
TNP-D	$\mathbf{0.117 \pm 0.001}$	$\mathbf{1.466 \pm 0.005}$

Comparison with Conv(C)NP ConvCNP and ConvNP (Gordon et al., 2019; Foong et al., 2020) build translation equivariance into NPs, which should be helpful in specific domains such as images. However, table 9 shows that even in image completion, TNPs still outperform Conv(C)NP by a large margin. For off-grid data, Conv(C)NP is only applicable when x is one-dimensional, as they require a discretization step, limiting their application to high-dimensional BO and contextual bandits.

Table 9. TNP vs Conv(C)NP on EMNIST image completion. We train 5 instances with different seeds for each method and report the mean and std.

Method	Seen classes (0-9)	Unseen classes (10-46)
ConvCNP	1.02 ± 0.00	0.92 ± 0.00
ConvNP	1.15 ± 0.00	1.10 ± 0.00
TNP-A	$\mathbf{1.54 \pm 0.01}$	$\mathbf{1.41 \pm 0.01}$
TNP-D	1.46 ± 0.01	1.31 ± 0.00
TNP-ND	1.50 ± 0.00	1.31 ± 0.00

Qualitative results We visualize the completion of randomly selected images produced by TNPs and the baselines for both CelebA and EMNIST datasets.

Figures 8, 9, and 10 show the visualizations. It is clear from the visualizations that the variants of TNPs produce more accurate images and with less artifacts. This is especially evident when we look at Figure 10 which shows the completed images for unseen classes. TNPs are the only method that produced interpretable letters.

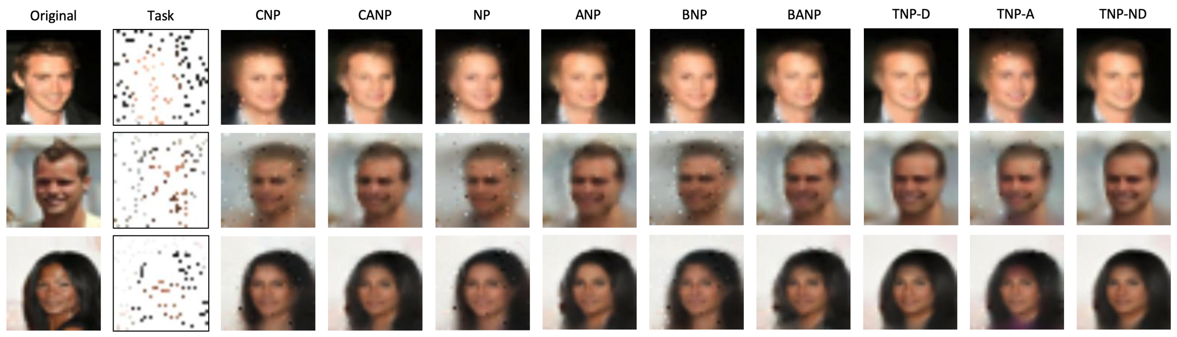


Figure 8. Completed images produced by TNPs and the baselines. The number of context points is 100.

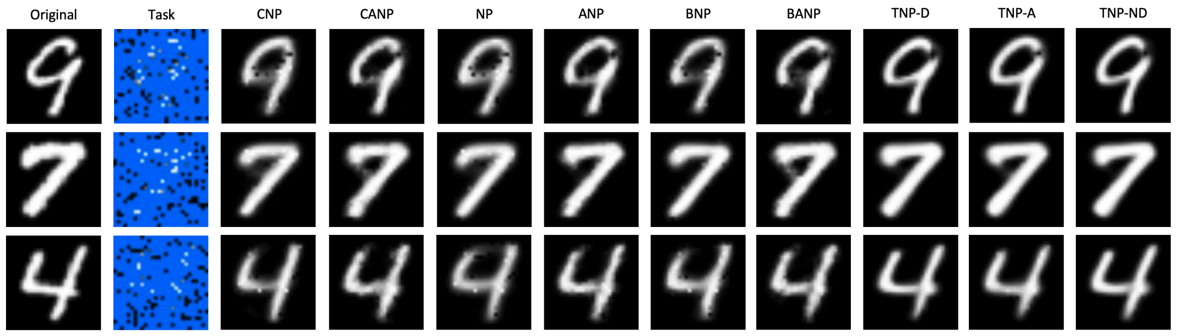


Figure 9. Image completion on seen classes by TNPs and the baselines. The number of context points is 100.

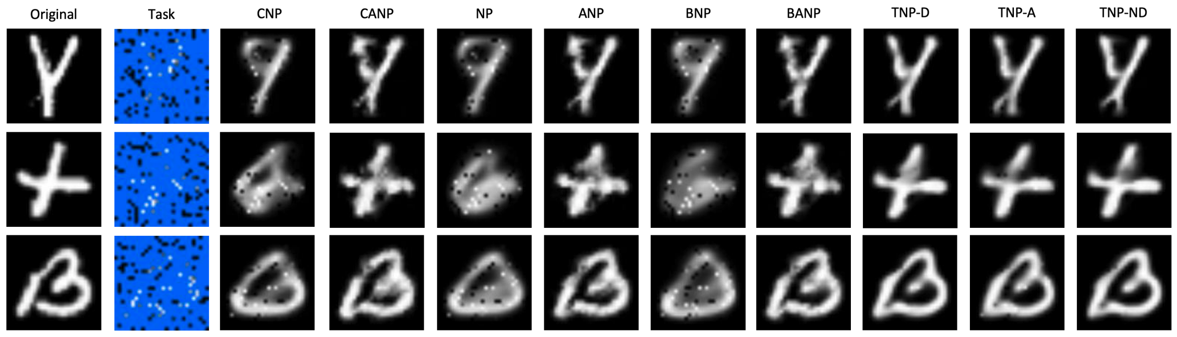


Figure 10. Image completion on unseen classes TNPs and the baselines. The number of context points is 100.

B.3. Contextual bandit

Table 10. Comparison of TNPs with the baselines on simple regret on contextual bandit problems with different values of δ. We run each model 50 times for each value of δ and report the mean and standard deviation.

Method	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$	$\delta = 0.995$	$\delta = 0.999$
Uniform	100.00 ± 20.77	100.00 ± 34.60	100.00 ± 50.34	100.00 ± 96.59	100.00 ± 114.30	100.00 ± 120.11
CNP	4.06 ± 4.15	8.13 ± 9.44	8.06 ± 9.46	28.05 ± 19.91	39.50 ± 27.30	93.57 ± 64.70
CANP	1.25 ± 2.42	2.97 ± 5.39	8.82 ± 14.01	41.01 ± 69.41	46.49 ± 88.31	34.62 ± 115.25
NP	1.65 ± 2.43	2.95 ± 4.07	4.45 ± 4.56	18.64 ± 1.87	26.14 ± 2.65	62.13 ± 6.21
ANP	1.58 ± 2.22	3.92 ± 5.10	5.38 ± 5.66	19.69 ± 1.88	27.44 ± 2.63	66.39 ± 6.34
BNP	62.72 ± 15.44	57.22 ± 25.55	58.58 ± 37.25	63.02 ± 75.67	65.77 ± 84.45	78.07 ± 94.31
BANP	3.32 ± 3.12	11.31 ± 14	34.19 ± 28.13	65.60 ± 91.17	59.33 ± 101.53	30.13 ± 71.82
TNP-D	$\mathbf{0.67 \pm 3}$	1.41 ± 2.86	2.24 ± 5.46	3.13 ± 1.21	4.31 ± 1.75	9.16 ± 3.86
TNP-A	1.17 ± 1.73	2.53 ± 5.03	2.47 ± 6.83	$\mathbf{2.12 \pm 5.66}$	$\mathbf{3.53 \pm 13.42}$	$\mathbf{5.07 \pm 2.98}$
TNP-ND	1.40 ± 2.33	$\mathbf{0.78 \pm 2.30}$	$\mathbf{0.63 \pm 0.26}$	3.22 ± 1.25	4.86 ± 1.88	11.08 ± 4.36

C. TNPs vs other NP variants on sampling multiple functions

An advantage of using latent variables is the ability to sample diverse functions from the same set of context points. However, we can also sample multiple functions using an autoregressive sequence model. In this section, we present how we obtain samples from TNP-A, and qualitatively measure the sample diversity of TNP-A and the baselines on 1-D regression and image completion tasks.

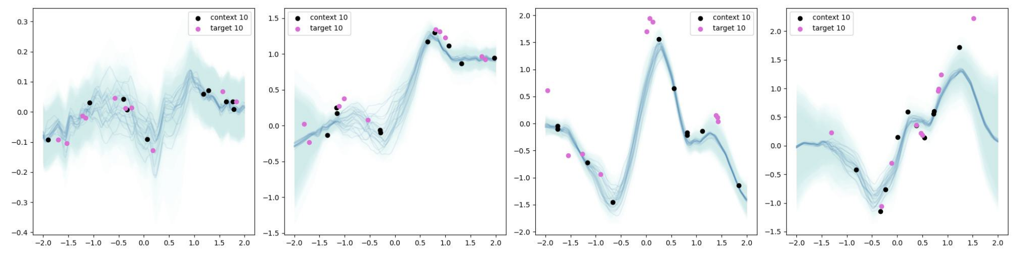
C.1. Sampling procedure of TNP-A

We sequentially sample the target points $y_{m+1:N}$ given the target inputs $x_{m+1:N}$ and the set of context points $\{x_i, y_i\}_{i=1}^m$. Specifically, for each step $i = 1$ to $i = N - m$, we:

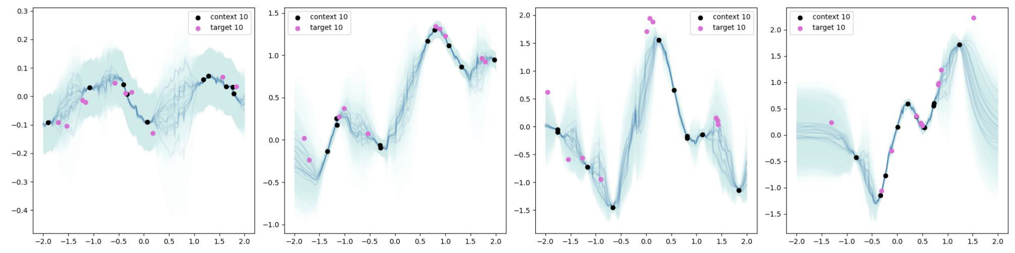
1. Obtain the predictive distribution $p(y_{m+i} \mid x_{1:m+i}, y_{1:m}, \hat{y}_{m+1:m+i-1})$, in which $\hat{y}_{m+1:m+i-1}$ are samples of the previous target points.
2. Sample $\hat{y}_{m+i} \sim p(y_{m+i} \mid x_{1:m+i}, y_{1:m}, \hat{y}_{m+1:m+i-1})$.
3. Repeat

C.2. Sampling results

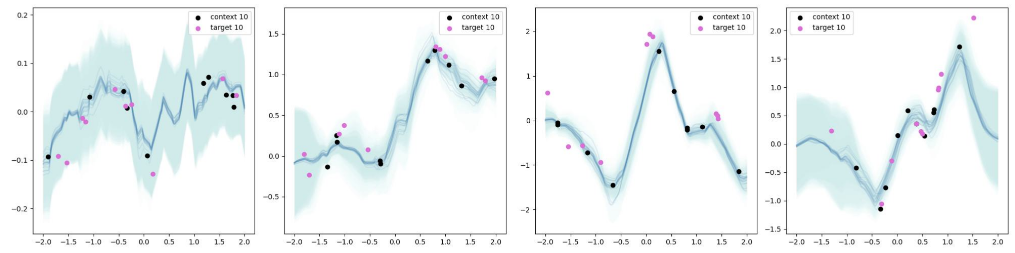
1-D regression Figure 11 and 12 show sample functions produced by NP, ANP, BNP, BANP, and TNP-A given 10 and 30 context points, respectively. Each method is tested on the same set of 4 underlying functions, which are sampled from a GP with an RBF kernel. In the figures, each solid blue curve is a sample function, and the blue area around the curve represents the variance of the predictive distribution over y. We can see that when the number of context points is 10, all methods can produce different samples. However, TNP-A tends to have more diverse samples (more diverse solid curves), while the other NP methods use the predictive variance to account for their uncertainty, which result in large blue areas. When there are 30 context points, all methods produce consistent functions.



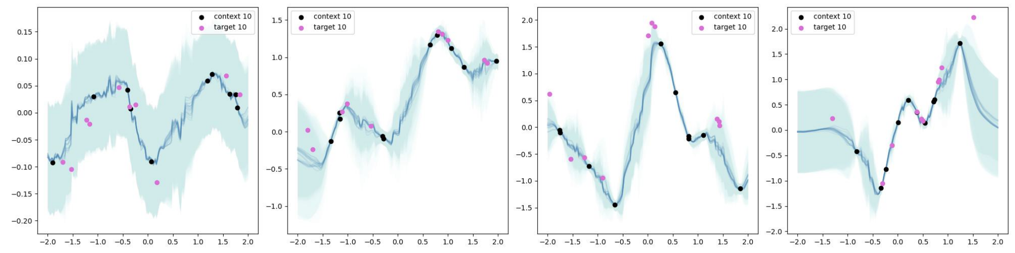
(a) Sample functions produced by NPs.



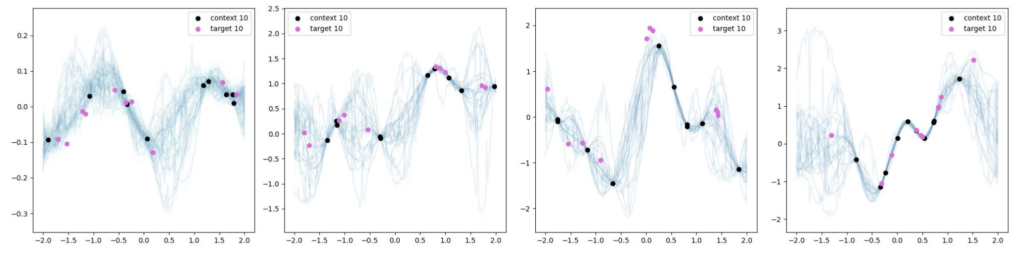
(b) Samples functions produced by ANPs.



(c) Samples functions produced by BNPs.

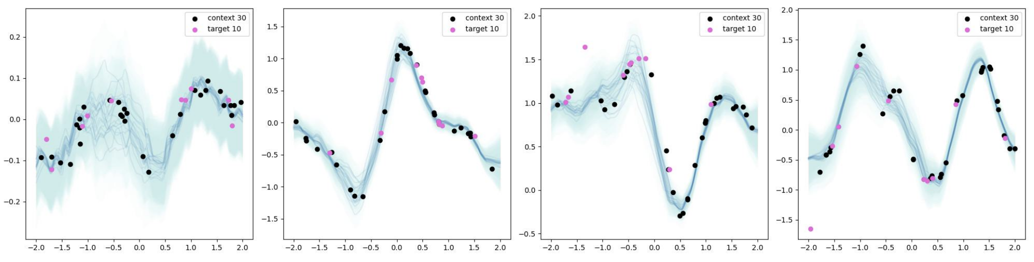


(d) Samples functions produced by BANPs.

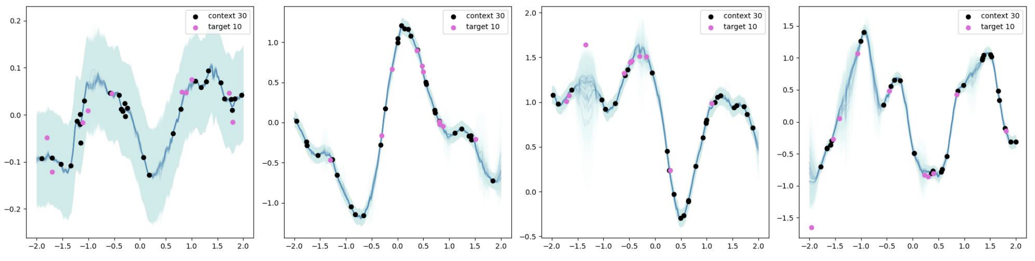


(e) Samples functions produced by TNP-A.

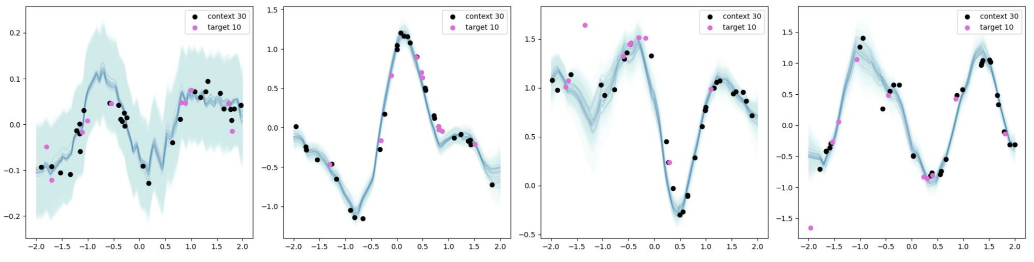
Figure 11. Sample functions produced by TNPs and the baselines given 10 context points. Data is generated from a GP with an RBF kernel. Each solid blue curve corresponds to one sample function, and the blue area around each curve represents the variance in the predictive distribution.



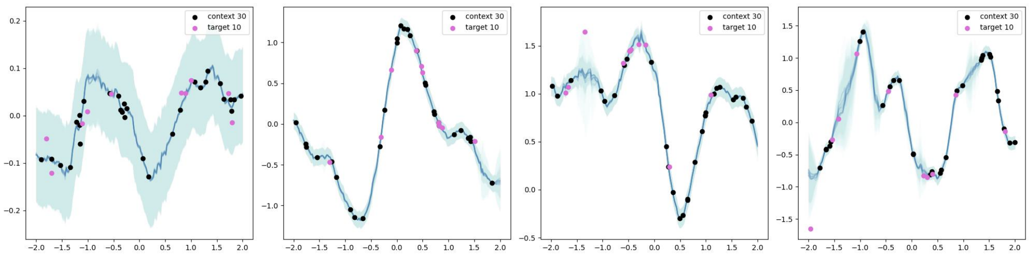
(a) Sample functions produced by NPs.



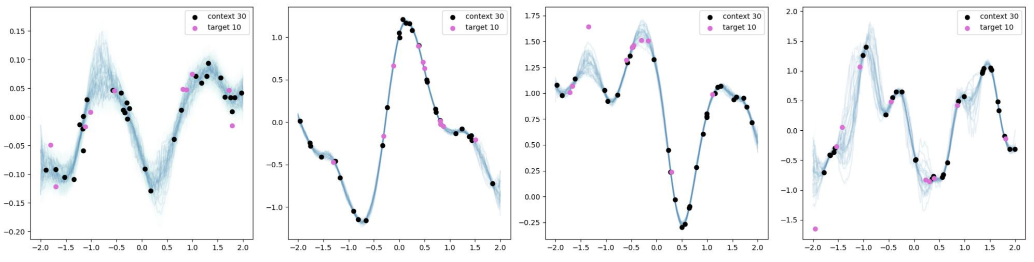
(b) Samples functions produced by ANPs.



(c) Samples functions produced by BNPs.



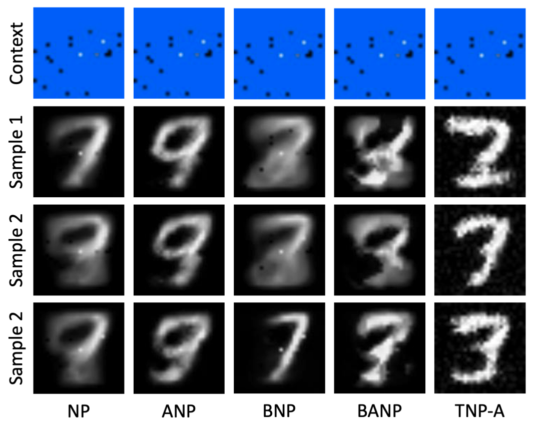
(d) Samples functions produced by BANPs.



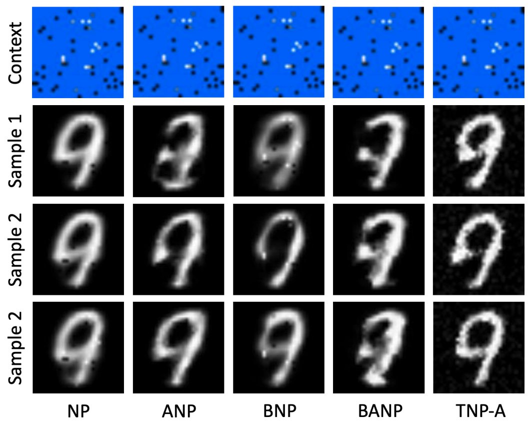
(e) Samples functions produced by TNP-A.

Figure 12. Sample functions produced by TNPs and the baselines given 30 context points. Data is generated from a GP with an RBF kernel. Each solid blue curve corresponds to one sample function, and the blue area around each curve represents the variance in the predictive distribution.

Image completion Figure 13 show different completed images produced by NP, ANP, BNP, BANP, and TNP-A given 20 and 50 context points, respectively. When the number of context points is only 20, there are multiple possible digits that can be generated from the same set of context points. TNP-A produces three samples that represent three different digits, while the diversity of other NP methods is not clear. When the number of context points is 50, all methods produce consistent samples that represent the digit 9.



(a) Samples produced by TNPs and the baselines given 20 context points.



(b) Samples produced by TNPs and the baselines given 50 context points.

Figure 13. Sample images produced by TNPs and the baselines given the same set of context points. The original image is drawn randomly from the test set.

D. Architectural ablation analysis

Before coming up with the final architectures of TNPs, we had experimented with other possible architectures. In this section, we present these alternative design choices, and their performances compared to TNPs.

D.1. TNPs with alternative input representations

There are other input representations to TNPs in addition to concatenating x and y. We present these alternatives in this section and show how they perform compared to TNPs. We show ablation for TNP-D only for simplicity.

Separate embedding layers for context and target points: The first idea is to use two different embedding layers, one for the context points and one for the target points. The embedding layer for context points will take $(x_{1:C}, y_{1:C})$ as the input, while the embedding layer for target points will only take $x^*_{1:T}$ as the input. Figure 14 depicts this model.

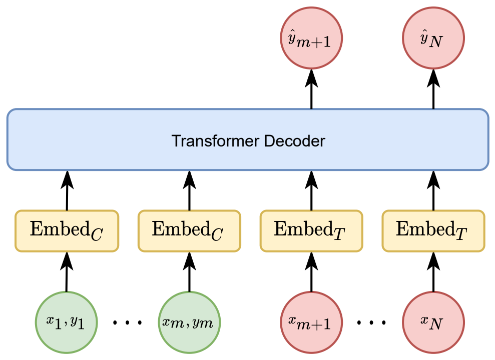


Figure 14. TNPs with two separate embedding layers for the context points and target points.

Cross attention: Another idea is to use cross-attention, which is depicted in Figure 15. Specifically, we first implement a stack of self-attention layers to learn the interaction between the $x's$. The outputs of this module will serve as queries and keys in a cross-attention layer, while the context labels $y_{1:C}$ serve as values. This eliminates the need of a mask because the self-attention layer only computes the similarity of $x's$ by using them as queries and keys.

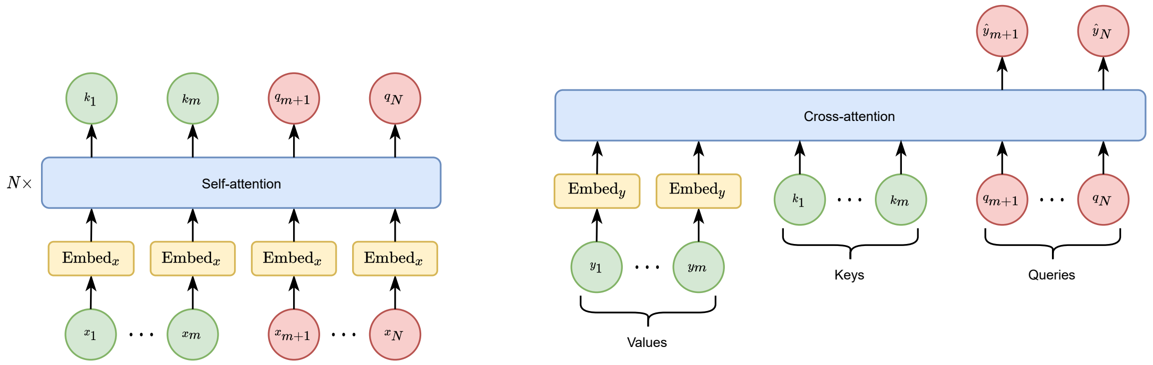


Figure 15. TNPs with cross attention.

Comparison: Table 11 compares the two alternative input representations with TNP-D. We see that TNP-D is slightly better than Sep-emb, while Cross-att barely works for any evaluation kernel. This is because the model is not allowed to look at pairs of context points $(x_i, y_i)_{i=1}^{C}$, thus cannot infer the underlying distribution f, or in other words cannot tell which realisation of GP these pairs come from. Therefore, the best thing it can do is to make a random guess with a very large variance. This confirms that coupling x and y to form the input representation is crucial to the performance of TNP.

Table 11. Comparison of TNP-D vs. the alternative input representations.

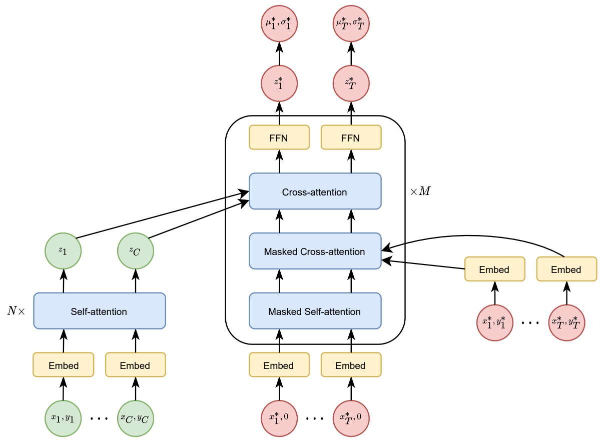
Metric	Method	RBF	Matérn 5/2	Periodic
RMSE	TNP-D	$\mathbf{0.177 \pm 0.001}$	$\mathbf{0.222 \pm 0.000}$	$\mathbf{0.664 \pm 0.014}$
	Sep-emb	$\mathbf{0.177 \pm 0.000}$	$\mathbf{0.222 \pm 0.000}$	0.685 ± 0.006
	Cross-att	0.304 ± 0.122	0.328 ± 0.112	0.668 ± 0.050
CE	TNP-D	$\mathbf{0.043 \pm 0.000}$	0.045 ± 0.000	$\mathbf{0.129 \pm 0.012}$
	Sep-emb	0.045 ± 0.001	$\mathbf{0.044 \pm 0.001}$	0.135 ± 0.008
	Cross-att	0.151 ± 0.055	0.117 ± 0.038	0.203 ± 0.081
Log-Likelihood	TNP-D	$\mathbf{1.388 \pm 0.004}$	$\mathbf{0.954 \pm 0.005}$	-3.525 ± 0.367
	Sep-emb	1.375 ± 0.004	$\mathbf{0.954 \pm 0.003}$	$\mathbf{-3.232 \pm 0.238}$
	Cross-att	0.115 ± 0.337	-0.012 ± 0.281	-2.989 ± 1.225

D.2. Autoregressive Transformer Neural Processes

There are other architectures that enable autoregressiveness in addition to what was presented in Section 3.1. We present those architectures here, their differences with TNP-A, and their relative performances.

Figure 16 shows the two alternative architectures. In TNP-A-1 (Figure 16a), we encode the context points and the target points separately, and then use a stack of masked self-attention, masked-cross attention, and cross-attention layers that allow each target point to attend all the context points and the previous target points. In each masked layer, we ensure that the component at position i^{th} cannot attend positions $j^{th}, j > i$. Note that in this model, there is no interaction between the context points and the target points.

In the TNP-A-2 (Figure 16b), we allow this interaction by feeding all context points and target points into the same encoder. In the encoder, we use a masking mechanism such that the context points do not attend the target points, as in evaluation we do not have ground-truth but only predicted target. Similarly to TNP-A-2, the masked cross-attention module in Pure GPT 2 stops the component at position i^{th} from attending positions j^{th} with $j > i$.


(a) TNP-A-1

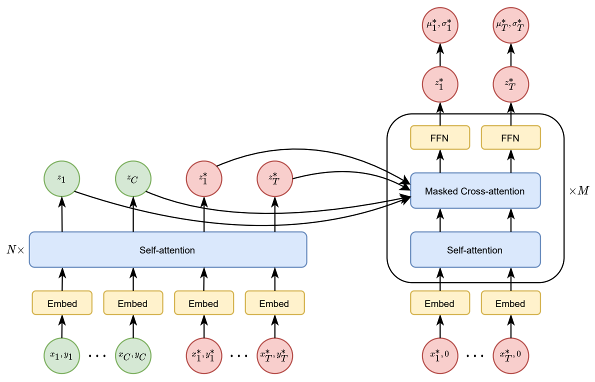

(b) TNP-A-2

Figure 16. The alternative architectures for TNP-A

Comparison: We compare the relative performances of TNP, TNP-A-1, and TNP-A-2 in the meta regression task. Table 12 shows the results.

Table 12. Comparison of TNP-A vs. the alternative autoregressive models.

Metric	Method	RBF	Matérn 5/2	Periodic
RMSE	TNP-A	$\mathbf{0.178 \pm 0.000}$	$\mathbf{0.222 \pm 0.000}$	$\mathbf{0.660 \pm 0.002}$
	TNP-A-1	0.182 ± 0.001	0.225 ± 0.000	0.673 ± 0.007
	TNP-A-2	0.184 ± 0.002	0.223 ± 0.001	0.667 ± 0.008
CE	TNP-A	0.045 ± 0.000	0.044 ± 0.000	$\mathbf{0.119 \pm 0.008}$
	TNP-A-1	$\mathbf{0.044 \pm 0.000}$	0.044 ± 0.000	0.147 ± 0.009
	TNP-A-2	0.046 ± 0.002	$\mathbf{0.042 \pm 0.001}$	0.168 ± 0.009
Log-Likelihood	TNP-A	$\mathbf{1.628 \pm 0.001}$	$\mathbf{1.207 \pm 0.003}$	$\mathbf{-2.257 \pm 0.168}$
	TNP-A-1	1.477 ± 0.016	1.070 ± 0.018	-3.380 ± 0.314
	TNP-A-2	1.577 ± 0.024	1.167 ± 0.018	-4.607 ± 0.390

D.3. Low-rank approximation for TNP-ND

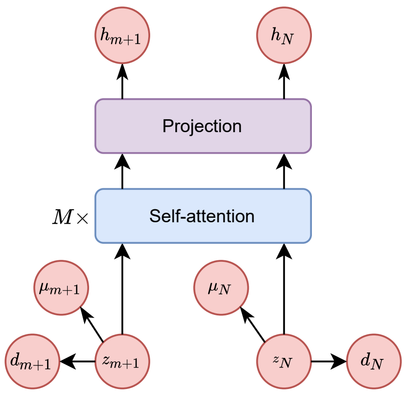


Figure 17. The decoder architecture of the low-rank version of TNP-ND.

As mentioned in Section 3.3, we can use Low-rank approximation to parameterize the covariance matrix in the predictive distribution $\mathcal{N}(y^*_{1:T} \mid \mu_\theta(x^*_{1:T}, C), \Sigma_\theta(x^*_{1:T}, C))$. Figure 17 depicts the parameterization, which is similar to TNP-ND, except that we have an additional MLP that outputs $d_{m+1:N}$. The covariance is computed as:

$$\Sigma = HH^T + \exp(D),\ H \in \mathbb{R}^{n\times p},\ D \in \mathbb{R}^{n\times n}, \tag{12}$$

in which D is a diagonal matrix formed by $d_{m+1:N}$. This parameterization guarantees that Σ is a positive definite matrix.

Results We compare TNP-ND with Cholesky decomposition vs TNP-ND with low-rank approximation in the 1-D regression problem. Table 13 shows the results.

Table 13. Comparison of TNP-ND with Cholesky decomposition vs TNP-ND with low-rank approximation on various GP kernels and evaluation metrics. We train 5 instances with different seeds for each model and report the mean and std.

Metric	Method	RBF	Matérn 5/2	Periodic
RMSE	TNP-ND-Cholesky	$\mathbf{0.180 \pm 0.001}$	$\mathbf{0.223 \pm 0.000}$	$\mathbf{0.670 \pm 0.009}$
	TNP-ND-Lowrank	0.186 ± 0.001	0.225 ± 0.001	0.680 ± 0.010
CE	TNP-ND-Cholesky	0.048 ± 0.001	0.050 ± 0.001	0.155 ± 0.009
	TNP-ND-Lowrank	$\mathbf{0.043 \pm 0.001}$	$\mathbf{0.044 \pm 0.001}$	$\mathbf{0.153 \pm 0.019}$
Log-Likelihood	TNP-ND-Cholesky	1.46 ± 0.00	1.02 ± 0.00	$\mathbf{-4.13 \pm 0.33}$
	TNP-ND-Lowrank	$\mathbf{1.565 \pm 0.007}$	$\mathbf{1.117 \pm 0.003}$	-5.702 ± 0.509

E. Pretraining TNPs

In the main results of the paper, we trained TNPs in a meta-training regime, where the model makes predictions for the target points conditioning on the context. This was explicitly designed to match the evaluation setting. However, previous studies (Brown et al., 2020; Chan et al., 2022) have shown that the capacity for in-context learning in transformer-based models is emergent. That is, the model is capable of performing few-shot learning, without being explicitly trained to do so. We conduct experiments to investigate this emergent in-context learning behavior of TNPs. Specifically, during training, the model observes sequences of evaluations $(x_{1:N}, y_{1:N})$, and learns to predict each point given the preceding points:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_{1:N},y_{1:N}}[\log p_\theta(y_{1:N}|x_{1:N})] = \mathbb{E}_{x_{1:N},y_{1:N}}\left[\sum_{m=1}^{N} \log p_\theta(y_m|x_{1:m}, y_{1:m-1})\right] \tag{13}$$

We term the variant trained using this autoregressive objective TNP-A-Pretrained. During evaluation, we introduce the notions of context and target via proper masking, where each target conditions on the context points and the preceding predicted target points. This resembles TNP-A, and the only difference is in the training phase. We compare TNP-A-Pretrained to TNP-A, TNP-D, and TNP-ND in all 4 meta-learning tasks below.

E.1. 1D Regression

Table 14. Comparison of TNP-A-Pretrained vs three TNP variants with various evaluation metrics. We train 5 instances with different seeds for each model and report the mean and std.

Metric	Method	RBF	Matérn 5/2	Periodic
RMSE	TNP-D	$\mathbf{0.177 \pm 0.001}$	$\mathbf{0.222 \pm 0.000}$	0.664 ± 0.014
	TNP-A	0.178 ± 0.000	$\mathbf{0.222 \pm 0.000}$	$\mathbf{0.660 \pm 0.002}$
	TNP-ND	0.180 ± 0.001	0.223 ± 0.000	0.670 ± 0.009
	TNP-A-Pretrained	0.180 ± 0.001	0.224 ± 0.001	0.679 ± 0.007
CE	TNP-D	$\mathbf{0.043 \pm 0.000}$	0.045 ± 0.000	0.129 ± 0.012
	TNP-A	0.045 ± 0.000	$\mathbf{0.044 \pm 0.000}$	$\mathbf{0.119 \pm 0.008}$
	TNP-ND	0.048 ± 0.001	0.050 ± 0.001	0.155 ± 0.009
	TNP-A-Pretrained	$\mathbf{0.043 \pm 0.005}$	0.052 ± 0.007	0.148 ± 0.020
Log-Likelihood	TNP-D	1.39 ± 0.00	0.95 ± 0.01	-3.53 ± 0.37
	TNP-A	$\mathbf{1.63 \pm 0.00}$	$\mathbf{1.21 \pm 0.00}$	$\mathbf{-2.26 \pm 0.17}$
	TNP-ND	1.46 ± 0.00	1.02 ± 0.00	-4.13 ± 0.33
	TNP-A-Pretrained	$\mathbf{1.63 \pm 0.01}$	1.19 ± 0.03	-3.69 ± 0.92

E.2. Image Completion

Table 15. Comparison of TNP-A-Pretrained vs three TNP variants on CelebA dataset with various evaluation metrics. We train 5 instances with different seeds for each model and report the mean and std.

Method	RMSE	Log-likelihood
TNP-D	0.112 ± 0.000	3.891 ± 0.006
TNP-A	0.115 ± 0.000	$\mathbf{5.818 \pm 0.011}$
TNP-ND	$\mathbf{0.111 \pm 0.000}$	5.477 ± 0.016
TNP-A-Pretrained	0.114 ± 0.000	4.457 ± 0.011

Table 16. Comparison of TNP-A-Pretrained vs three TNP variants on EMNIST dataset with various evaluation metrics. We train 5 instances with different seeds for each model and report the mean and std. We evaluate on both seen and unseen classes.

Setting	Method	RMSE	Log-likelihood
Seen classes (0-9)	TNP-D	0.119 ± 0.002	1.461 ± 0.010
	TNP-A	0.122 ± 0.001	$\mathbf{1.537 \pm 0.005}$
	TNP-ND	$\mathbf{0.116 \pm 0.000}$	1.497 ± 0.002
	TNP-A-Pretrained	0.130 ± 0.000	1.497 ± 0.002
Unseen classes (10-46)	TNP-D	$\mathbf{0.139 \pm 0.001}$	1.308 ± 0.003
	TNP-A	0.142 ± 0.001	$\mathbf{1.413 \pm 0.005}$
	TNP-ND	0.140 ± 0.001	1.314 ± 0.004
	TNP-A-Pretrained	0.159 ± 0.000	1.256 ± 0.008

E.3. Contextual Bandits

Table 17. Comparison of TNP-A-Pretrained vs three TNP variants on cumulative regret on contextual bandit problems with different values of δ. We run each model 50 times for each value of δ and report the mean and std.

Method	$\delta = 0.7$	$\delta = 0.9$	$\delta = 0.95$	$\delta = 0.99$	$\delta = 0.995$	$\delta = 0.999$	Average
Uniform	100.00 ± 1.18	100.00 ± 3.03	100.00 ± 4.16	100.00 ± 7.52	100.00 ± 8.11	100.00 ± 7.96	100.00 ± 5.97
TNP-D	$\mathbf{1.18 \pm 0.94}$	1.70 ± 0.41	2.55 ± 0.43	$\mathbf{3.57 \pm 1.22}$	$\mathbf{4.68 \pm 1.09}$	9.56 ± 0.44	$\mathbf{3.87 \pm 2.91}$
TNP-A	3.67 ± 4.88	4.04 ± 2.38	4.29 ± 2.36	5.79 ± 5.27	9.29 ± 7.62	$\mathbf{6.13 \pm 2.50}$	5.54 ± 4.98
TNP-ND	1.76 ± 0.61	$\mathbf{1.41 \pm 0.98}$	$\mathbf{1.61 \pm 1.65}$	4.98 ± 2.84	7.22 ± 3.28	13.66 ± 2.92	5.11 ± 4.94
TNP-A-Pretrained	1.53 ± 0.09	4.96 ± 0.17	8.00 ± 0.13	23.98 ± 0.42	34.35 ± 0.01	83.79 ± 0.02	26.10 ± 28.22

E.4. Bayesian Optimization

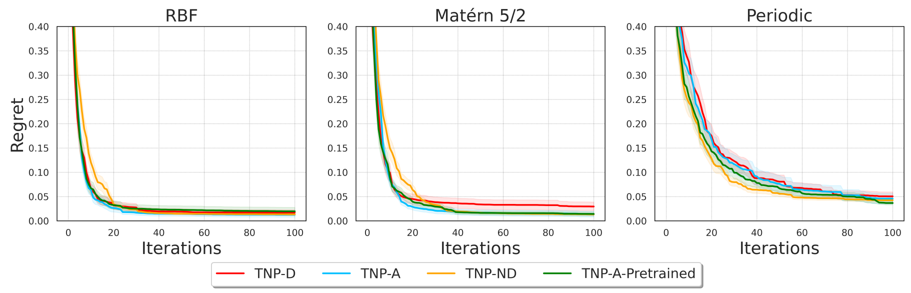


Figure 18. Regret performance on 1D BO tasks of TNP-A-Pretrained and the three TNP variants. For each kernel, we generate 100 functions and report the mean and standard deviation.

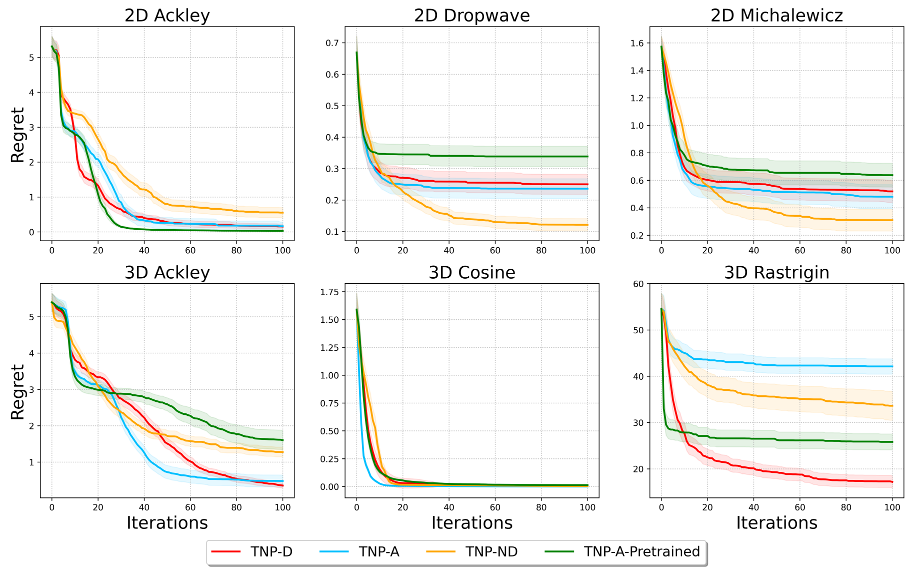


Figure 19. Regret performance on 2D and 3D BO tasks of TNP-A-Pretrained and the three TNP variants. For each function, we run BO for 100 times with different seeds and report the mean and standard deviation.

Overall, TNP-A-Pretrained performs reasonably well in all tasks, but still lags behind the meta-trained TNPs, especially in the two sequential decision making tasks. Improving this pretraining scheme to match the performance of the meta-trained models is an interesting research direction for future work.