Appendices to Contrastive Conditional Neural Processes

This contains three sections for describing experimental datasets, additional experimental results and detailed model implementation.

A. Dataset Description

The experiments in this work are three folds, including 1D time-series function, 2D predator-prey dynamics and high-dimensional time-series datasets.

A.1. 1D time-series functions

We run few-shot regression on four function families, see Tab. 1 for the specific definition of each. Each function instantiation used in training/validation/testing is generated with respect to the specified range therein. By following conventional meta-learning setting, instantiations within a batch are randomly sampled from the dataset of sinusoid functions without replacement, *i.e.* $f_i \neq f_{i'},\ \forall f_i \in f_B$, with $f_B = \{f_i\}_{i=1:|B|}$.

We generate 500 samples for each function family with splitting training/validation/testing in the $9:1:1$ ratio, without function overlap. For each instantiation, context size is sampled from $(0, N]$ (N refers to N-shot) with extra target size is drawn from $(0, 10]$ in the training phase, whilst in validation and testing phase the context size is fixed at N and prediction is evaluated on the whole sequence. The reported results are acquired on the test set. The dimension of observation space is $\mathcal{Y} \subseteq \mathbb{R}$.

A.2. 1D GP-generated functions

In addition to the above four function families, we also conduct 1D regression experiments where the data-generating functions are Gaussian Processes with different kernels. We use the kernels as in AttnCNP and ConvCNP, including RBF, Periodic and Noisy Matérn:

$$
\begin{aligned}
k(x_i, x_j)_{\mathrm{RBF}} &= \exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right) \\
k(x_i, x_j)_{\mathrm{PER}} &= \exp\left(-\frac{2\sin^2\left(\pi d\left(x_i, x_j\right)/p\right)}{l^2}\right) \\
k(x_i, x_j)_{\mathrm{MAT}} &= \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l} d\left(x_i, x_j\right)\right)^{\nu} \\
&\quad K_\nu\left(\frac{\sqrt{2\nu}}{l} d\left(x_i, x_j\right)\right) + \epsilon
\end{aligned}
\tag{1}
$$

The number of instantiations we generate for each kernel is 4096, of which 256 are used for testing and 256 for validation, respectively. Other than that, we use the same sampling strategy as time-series functions, and perform N-shot regression.

A.3. 2D predator-prey dynamics

To model the 2D population dynamics, we fit the Lotka-Volterra equations (LV) with CNPs. Given values of y_1 and y_2 at initial time index $x = 0$, the poluations of both species vary after each time increment based on interaction coefficients $\alpha, \beta, \delta, \gamma$. The growth rates are defined by

$$
\begin{aligned}
\nabla_x y_1 &= \alpha y_1 - \beta y_1 y_2 \\
\nabla_x y_2 &= \delta y_1 y_2 - \gamma y_2
\end{aligned}
\tag{2}
$$

with respect to time increment $x : 0 \rightarrow x_{\max}$. Thus, in each simulated trajectory, the populations at each time index are determined by the initial values of $y_1, y_2, \alpha, \beta, \gamma, \delta$. We consider two modes of simulation. One is *Greek* mode where $\alpha, \beta, \gamma, \delta$ are set to fixed values with y_1 and y_2 are randomly sampled from the given range, while in *Population* mode y_1 and y_2 are assigned with fixed initial numbers with $\alpha, \beta, \gamma, \delta$ become random variables (Tab. 2). For both modes, we run 200 trials as the meta-dataset with accumulating 150 timesteps, *i.e.* $x_{\max} = 150$ for every trial. We generate 200 samples for each function family with splitting training/validation/testing in the $9:1:1$ ratio, without function overlap. For each trajectory, context size is sampled from $(0, 80]$ with extra target size is drawn from $(0, 20]$ in the training phase, whilst in validation and testing phase

1D Synthetic Function				
Family	**Form**	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	$\boldsymbol{x}$
Sinusoid	$y = \alpha \sin(x - \beta)$	$(-1, 1)$	$(-0.5, 0.5)$	$(-\pi, \pi)$
Exponentials	$y = \alpha \times \exp(x - \beta)$	$(-1, 1)$	$(-0.5, 0.5)$	$(-1, 4)$
Oscillators	$y = \alpha \sin(x - \beta)\exp(-0.5t)$	$(-1, 1)$	$(-0.5, 0.5)$	$(0, 5)$
Straight lines	$y = \alpha\, x + \beta$	$(-1, 1)$	$(-0.5, 0.5)$	$(0, 5)$

Table 1. Details of 1D times-series functions. Columns of α, β, x corresponds to the range where a function instantiation is randomly sampled therein (e.g., $f_1 = -0.5\sin(x - 0.3)$ and $f_2 = 0.4\sin(x + 0.1)$).

2D Population Dynamics						
Mode	y_1	y_2	α	β	γ	δ
Greek	(0.5, 2.0)	(0.5, 2.0)	4/3	2/3	1	1
Population	1.6	0.8	(0.9, 1.1)	(0.05, 0.15)	(1.25, 1.75)	(0.5, 1.0)

Table 2. Details of 2D LV systems. y_1 and y_2 denote the initial population of predator and prey, resepctively. $\alpha, \beta, \delta, \gamma$ refer to the interaciton coefficients. Either initial population or interaction coefficients are set to fixed in a specific mode, while another group is randomly initialized. All the values have been normalized to avoid the impact caused by magnitude.

the context size is fixed at 80 and prediction is evaluated on the whole trajectory. The reported results are acquired on the test set. In this case, the dimension of observation space is $\mathcal{Y} \subseteq \mathbb{R}^2$.

A.4. Higher-dimensional time-series

We experiment with two higher-dimensional time-series dataset, where observations are depicted as images, including a BouncingBall dataset and a RotMNIST dataset. For Bouncing Ball, each trajectory contains the movements of three interacting balls within a rectangular box, with the length of 20 steps, where each timestep is framed as a $32*32$ image. We randomly grab 10000 abd 500 trajectories for training and testing, respectively. For RotMNIST, each trajectory contains the rotation of a handwritten digit "3" presenting 16 angles (so as with the length of 16 steps), where each timestep is frame as a $28*28$ image. Randomly drawn 400 sequences are used for experiment with a 9:1 ratio for splitting training/testing set. During the training phase for both datasets, the context size and extra target size are randomly drawn in the range $(0, 5]$ and $(0, 5]$. In this case, the dimension of observation space are $\mathcal{Y} \subseteq \mathbb{R}^{784}$ (RotMNIST) and $\mathcal{Y} \subseteq \mathbb{R}^{1024}$ (Bouncing Ball).

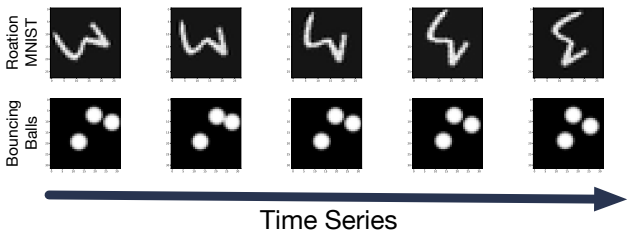

Figure 1. Examples of 5 consecutive steps of Bouncing Ball data and RotMNIST data.

B. Additional Experimental Results

B.1. Few-shot Regression on GP-generated Function

We supplement the results for 5-shot and 20-shot regression for GP-generated functions in three different kernels, with reconstruction error (MSE) used as the evaluation metrics. It is noticeable that ConvCNP shows particular merits when predicting periodic data, while CCNP performs better with noise contained (Noisy Matérn). Also, the *translation-equivariance* assumption baked into ConvCNP may not hold for every case.

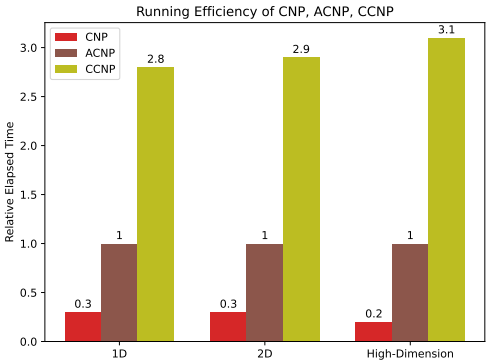

Figure 2. Comparison on Computational Efficiency

	5-shot regression			20-shot regression		
	RBF	**Periodic**	**Matérn**	**RBF**	**Periodic**	**Matérn**
CNP	0.723 ± 0.008	0.588 ± 0.001	0.972 ± 0.005	0.456 ± 0.019	0.535 ± 0.001	0.923 ± 0.044
AttnCNP	0.587 ± 0.004	0.552 ± 0.007	0.908 ± 0.007	0.119 ± 0.003	0.497 ± 0.022	0.569 ± 0.283
ConvCNP	$\underline{0.581 \pm 0.004}$	$\mathbf{0.278 \pm 0.007}$	$\underline{0.785 \pm 0.013}$	$\underline{0.102 \pm 0.001}$	$\mathbf{0.065 \pm 0.128}$	$\underline{0.468 \pm 0.151}$
CCNP	$\mathbf{0.509 \pm 0.032}$	$\underline{0.443 \pm 0.138}$	$\mathbf{0.635 \pm 0.004}$	$\mathbf{0.100 \pm 0.015}$	$\underline{0.183 \pm 0.029}$	$\mathbf{0.412 \pm 0.002}$

Table 3. Additional 1D Few-Shot Regression with GP in three kernels, running over 3 different seeds.

B.2. Running Efficiency

We have discussed that one of the limitations of CCNP is running efficiency due to the negative sampling steps when performing contrastive learning. We provide a comparison of initial relative elapsed time between CNP, AttnCNP, and CCNP running on the same epochs (see Fig. 2). CCNP primarily emphasizes predictive efficacy, so its efficiency may be sacrificed somewhat. Meanwhile, AttnCNP and CCNP are more comparable since both involve calculating attention, while CCNP costs more time to train contrastive targets. It might be possible to optimize CCNP's efficiency by using ideas like MoCo.

B.3. Ablation studies for Projection Heads

As it is a commonsense that the projection head plays a significant role in determining the performance of contrastive learning, we also examine the effects of setting different sizes for projection head through studies performed within high-dimensional sequences.

Dim.	RotMNIST MSE ($\times 10^{-2}$)	BouncingBall MSE ($\times 10^{-1}$)
8	0.751 ± 0.126	0.537 ± 0.002
16	0.687 ± 0.078	0.511 ± 0.001
32	0.654 ± 0.072	0.483 ± 0.008
64	$\underline{0.648 \pm 0.044}$	$\underline{0.470 \pm 0.005}$
128	$\mathbf{0.646 \pm 0.079}$	$\mathbf{0.458 \pm 0.004}$

Table 4. Ablation studies of Projection Head with pred_size=10

C. Additional Implementation Details

We implement the model with PyTorch 1.8.0 on a Nvidia GTX Titan XP GPU. See below for the details of CCNP's components.

C.1. CCNP for 1D

Input Encoder architecture for 1D. [1]

$$\begin{aligned}(x, \boldsymbol{y}) \in (\mathbb{R}, \mathbb{R}) &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \Rightarrow \mathbf{r}_C / \mathbf{r}_T / \mathbf{r}_F\end{aligned}$$

Position-Aware Self-Attention architecture for 1D.

$$\begin{aligned}\text{MultiHeadAttention} &= \{K, Q, V, A, H\} \\ \text{with } K &= \text{FCL}(64), \text{key\_transformation} \\ Q &= \text{FCL}(64), \text{query\_transformation} \\ V &= \text{FCL}(64), \text{value\_transformation} \\ A &= \text{DotProductAttention}(K, Q, V) \\ H &= \text{FCL}(64), \text{head\_fusion}\end{aligned}$$

where

$$\begin{aligned}\text{DotProductAttention} &= \{K, Q, V\} \\ \text{with } K &= \text{FCL}(64), \text{key\_transformation} \\ Q &= \text{FCL}(64), \text{query\_transformation} \\ V &= \text{FCL}(64), \text{value\_transformation}\end{aligned}$$

Temporal Contrastive Component for 1D.

$$\begin{aligned}(x_t, \mathbf{r}_T) \in (\mathbb{R}, \mathbb{R}^{64}) &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \Rightarrow \varphi(x_t, \mathbf{r}_T) \\ \varphi(x_t, \mathbf{r}_T) &\to \text{FCL}(8) \Rightarrow \hat{\boldsymbol{z}}_t \\ \boldsymbol{y}_t &\to \text{FCL}(8) \Rightarrow \boldsymbol{z}_t \\ (\hat{\boldsymbol{z}}_t, \boldsymbol{z}_t) &\to \text{InfoNCE} \\ \text{with } \tau &= 0.5\end{aligned}$$

[1] FCL(d) = Fully Connected Layer(output_dimension)

Function Contrastive Component for 1D. Taking 2 instantiations for illustration. We use instantiations within a whole batch in practice.

$$\begin{aligned}\mathbf{x}_C^{f_1}, \mathbf{x}_C^{f_2} &\to \mathbf{x}_{C_1}^{f_1}, \mathbf{x}_{C_2}^{f_1}, \mathbf{x}_{C_1}^{f_2}, \mathbf{x}_{C_2}^{f_2} \\ \mathbf{x}_{C_1}^{f_1}, \mathbf{x}_{C_2}^{f_1}, \mathbf{x}_{C_1}^{f_2}, \mathbf{x}_{C_2}^{f_2} &\to \mathbf{r}_{C_1}^{f_1}, \mathbf{r}_{C_2}^{f_1}, \mathbf{r}_{C_1}^{f_2}, \mathbf{r}_{C_2}^{f_2} \\ \mathbf{r}_{C_1}^{f_1} &\to \text{FCL}(8) \Rightarrow \mathbf{q}_i^{f_1} \\ \mathbf{r}_{C_2}^{f_1} &\to \text{FCL}(8) \Rightarrow \mathbf{q}_j^{f_1} \\ \mathbf{r}_{C_1}^{f_2} &\to \text{FCL}(8) \Rightarrow \mathbf{q}_i^{f_2} \\ \mathbf{r}_{C_2}^{f_2} &\to \text{FCL}(8) \Rightarrow \mathbf{q}_j^{f_2} \\ \mathbf{q}_i^{f_1}, \mathbf{q}_j^{f_1}, \mathbf{q}_i^{f_2}, \mathbf{q}_j^{f_2} &\to \text{InfoNCE} \\ \text{with } \tau &= 0.5\end{aligned}$$

Output Decoder architecture for 1D.

$$\begin{aligned}(x_t, \mathbf{r}_C, \mathbf{r}_T, \mathbf{r}_F) \in (\mathbb{R}, \mathbb{R}^{64}, \mathbb{R}^{64}, \mathbb{R}^{64}) &\to \text{concat}(\cdot, \cdot, \cdot, \cdot) \\ &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \to \text{ReLU} \\ &\to \text{FCL}(64) \to \text{ReLU} \\ &\Rightarrow \mathbf{r}_g \\ \mathbf{r}_g \in \mathbb{R}^{64} &\to \text{FCL}(1) \Rightarrow \hat{\mu}_t \\ \mathbf{r}_g \in \mathbb{R}^{64} &\to \text{FCL}(1) \Rightarrow \hat{\sigma}_t\end{aligned}$$

C.2. CCNP for 2D

Similar to 1D, except for $\boldsymbol{y} \in \mathbb{R}^2$, thus in decoder

$$\begin{aligned}\mathbf{r}_g \in \mathbb{R}^{64} &\to \text{FCL}(2) \Rightarrow \hat{\boldsymbol{\mu}}_t \\ \mathbf{r}_g \in \mathbb{R}^{64} &\to \text{FCL}(2) \Rightarrow \hat{\boldsymbol{\sigma}}_t\end{aligned}$$

C.3. CCNP for High-dimensional data

We replace $\psi(\cdot)$ for encoding observations $\mathbf{y}$ with Convolutional Blocks [2], with BN[3]. For decoding we use ConvT[4]

Input Encoder architecture for RotMNIST.

$$\begin{aligned}\boldsymbol{y} \in \mathbb{R}^{784} &\to \text{Conv}(16, 5, 2, 2) \to \text{BN}(16) \to \text{ReLU} \\ &\to \text{Conv}(32, 5, 2, 2) \to \text{BN}(32) \to \text{ReLU} \\ &\to \text{Conv}(64, 5, 2, 2) \to \text{BN}(64) \to \text{ReLU} \\ &\to \text{Conv}(128, 5, 2, 2) \to \text{BN}(128) \to \text{ReLU} \\ (x \in \mathbb{R}, \psi(\boldsymbol{y})) &\to \text{FCL}(128) \Rightarrow \mathbf{r}_C / \mathbf{r}_T / \mathbf{r}_F\end{aligned}$$

Output Decoder architecture for RotMNIST.

$$\begin{aligned}(x_t, \mathbf{r}_C, \mathbf{r}_T, \mathbf{r}_F) \in (\mathbb{R}, \mathbb{R}^{128}, \mathbb{R}^{128}, \mathbb{R}^{128}) &\to \text{concat}(\cdot, \cdot, \cdot, \cdot) \\ &\to \text{FCL}(72) \Rightarrow \mathbf{r}_g\end{aligned}$$

[2] Conv(f, k, s, p) = Convolution2D(feat_maps, kernel, stride, pad)
[3] BN(d) = BatchNormalization2D(dim)
[4] ConvT(f, k, s, p) = ConvTranspose2D(feat_map, kernel, stride, pad)

$$\begin{aligned}\mathbf{r}_g &\to \text{reshape}(\text{batch\_size}, 8, 28, 28)\\ &\to \text{ConvT}(128, 3, 1, 0) \to \text{BN}(128) \to \text{ReLU}\\ &\to \text{ConvT}(64, 5, 2, 0) \to \text{BN}(64) \to \text{ReLU}\\ &\to \text{ConvT}(32, 5, 2, 1) \to \text{BN}(32) \to \text{ReLU}\\ &\to \text{ConvT}(1, 5, 1, 2) \Rightarrow \hat{\boldsymbol{\mu}} \in \mathbb{R}^{28\times 28}\end{aligned}$$

Input Encoder architecture for Bouncing Ball.

$$\begin{aligned}\boldsymbol{y} \in \mathbb{R}^{1024} &\to \text{Conv}(16, 5, 2, 2) \to \text{BN}(16) \to \text{ReLU}\\ &\to \text{Conv}(32, 5, 2, 2) \to \text{BN}(32) \to \text{ReLU}\\ &\to \text{Conv}(64, 5, 2, 2) \to \text{BN}(64) \to \text{ReLU}\\ &\to \text{Conv}(128, 5, 2, 2) \to \text{BN}(128) \to \text{ReLU}\\ (x \in \mathbb{R}, \psi(\boldsymbol{y})) &\to \text{FCL}(128) \Rightarrow \mathbf{r}_C/\mathbf{r}_T/\mathbf{r}_F\end{aligned}$$

Output Decoder architecture for Bouncing Ball.

$$\begin{aligned}(x_t, \mathbf{r}_C, \mathbf{r}_T, \mathbf{r}_F) \in (\mathbb{R}, \mathbb{R}^{128}, \mathbb{R}^{128}, \mathbb{R}^{128}) &\to \text{concat}(\cdot, \cdot, \cdot, \cdot)\\ &\to \text{FCL}(72) \Rightarrow \mathbf{r}_g\end{aligned}$$

$$\begin{aligned}\mathbf{r}_g &\to \text{reshape}(\text{batch\_size}, 8, 32, 32)\\ &\to \text{ConvT}(128, 3, 2, 1) \to \text{BN}(128) \to \text{ReLU}\\ &\to \text{ConvT}(64, 5, 2, 1) \to \text{BN}(64) \to \text{ReLU}\\ &\to \text{ConvT}(32, 5, 2, 1) \to \text{BN}(32) \to \text{ReLU}\\ &\to \text{ConvT}(1, 5, 1, 2) \Rightarrow \hat{\boldsymbol{\mu}} \in \mathbb{R}^{32\times 32}\end{aligned}$$

References