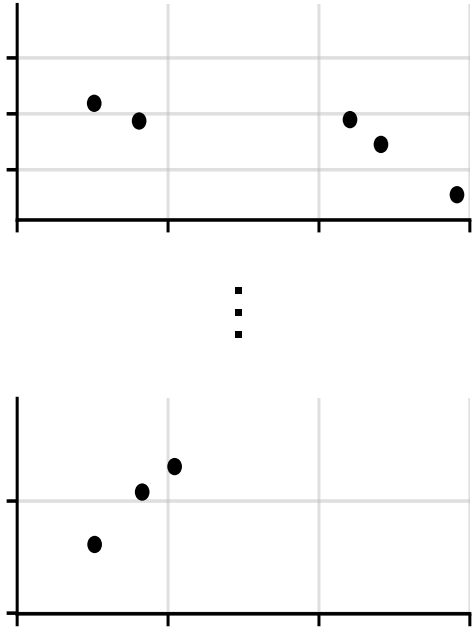
`NeuralProcesses.jl`: Composing Neural Processes with Flux

Wessel P. Bruinsma[1,2,*], Jonathan Gordon[1,*], Richard E. Turner[1]

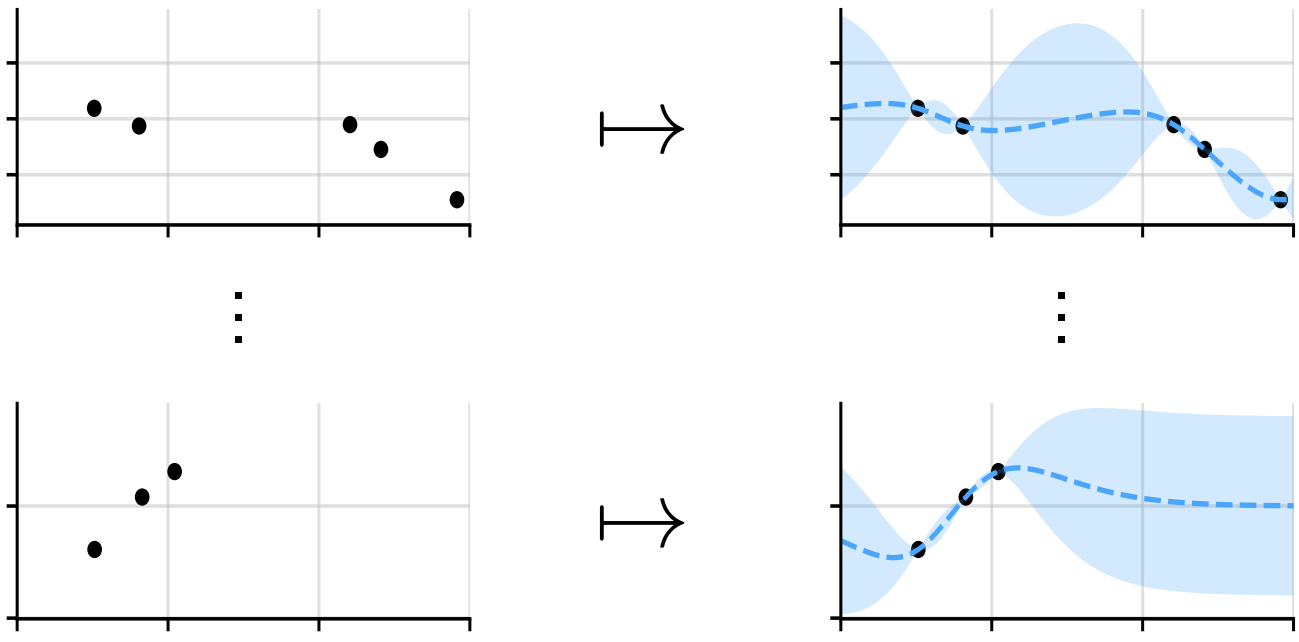
[1]University of Cambridge, [2]Invenia Labs
[*]Equal contribution

JuliaCon 2020

Setting: Learning to Predict

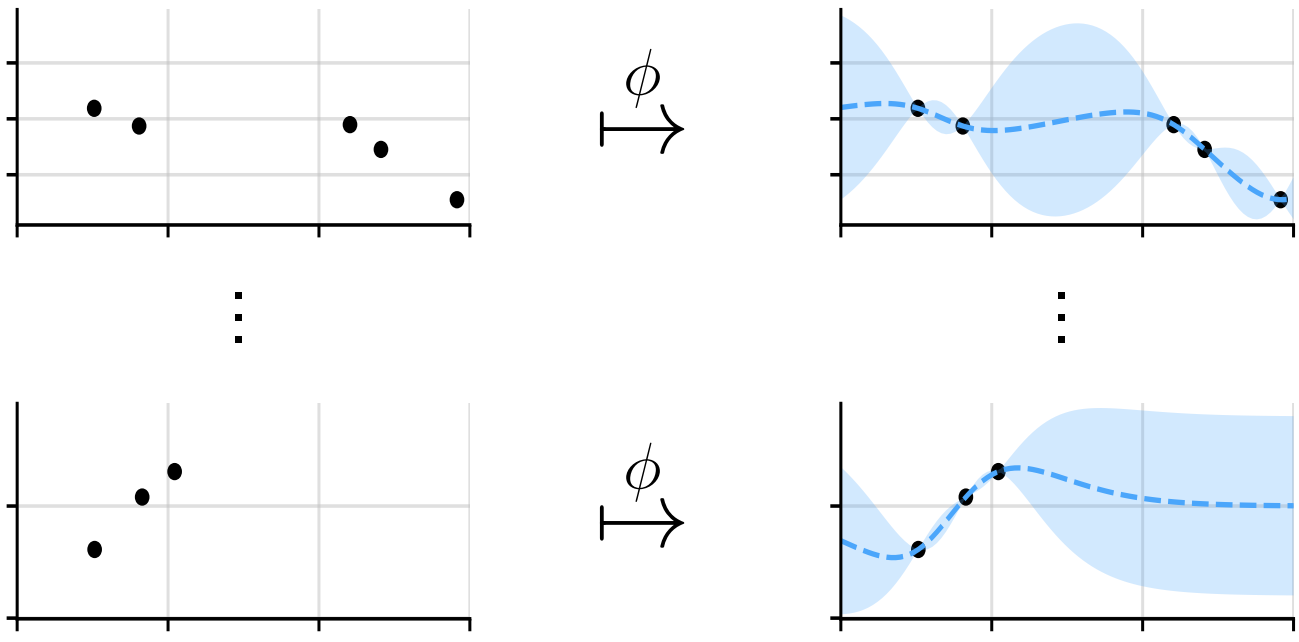


Setting: Learning to Predict

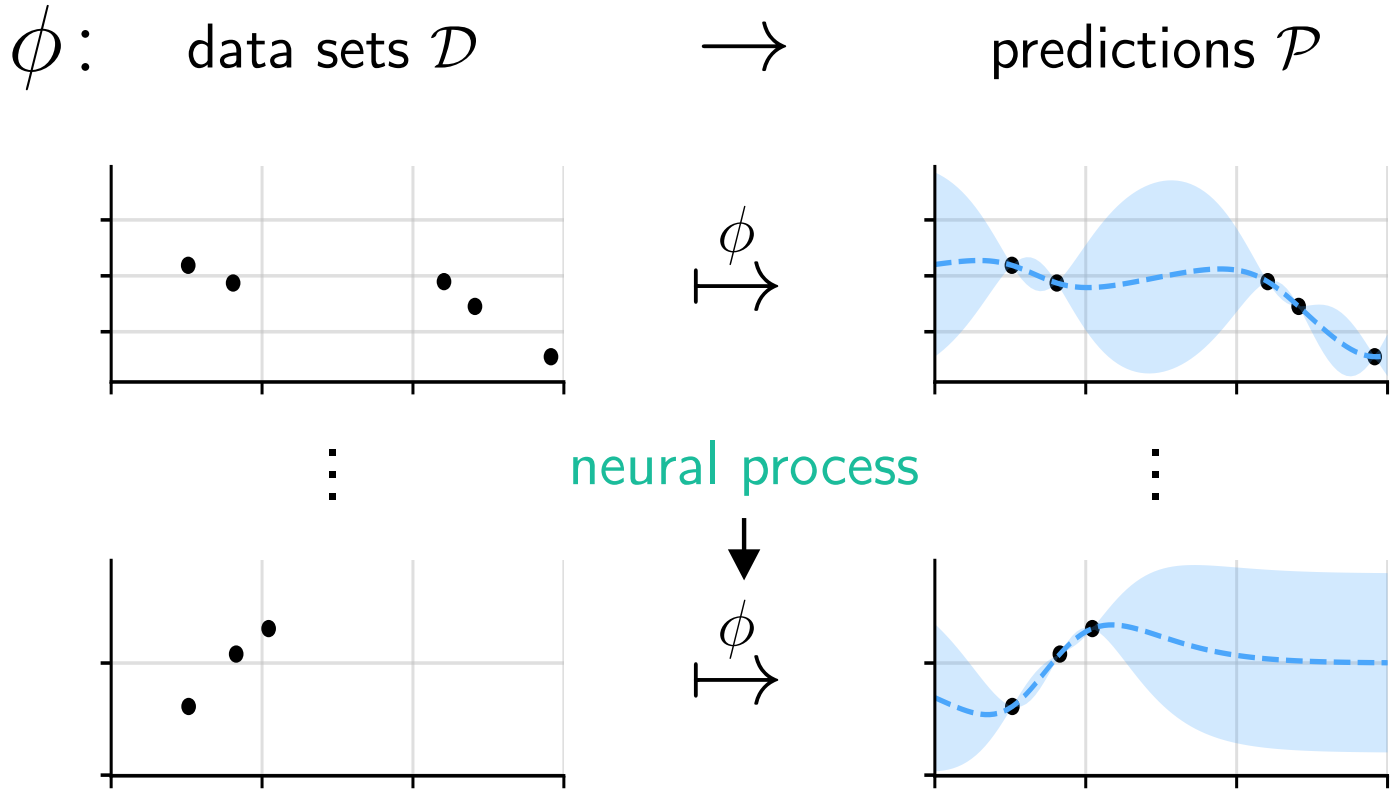


Setting: Learning to Predict

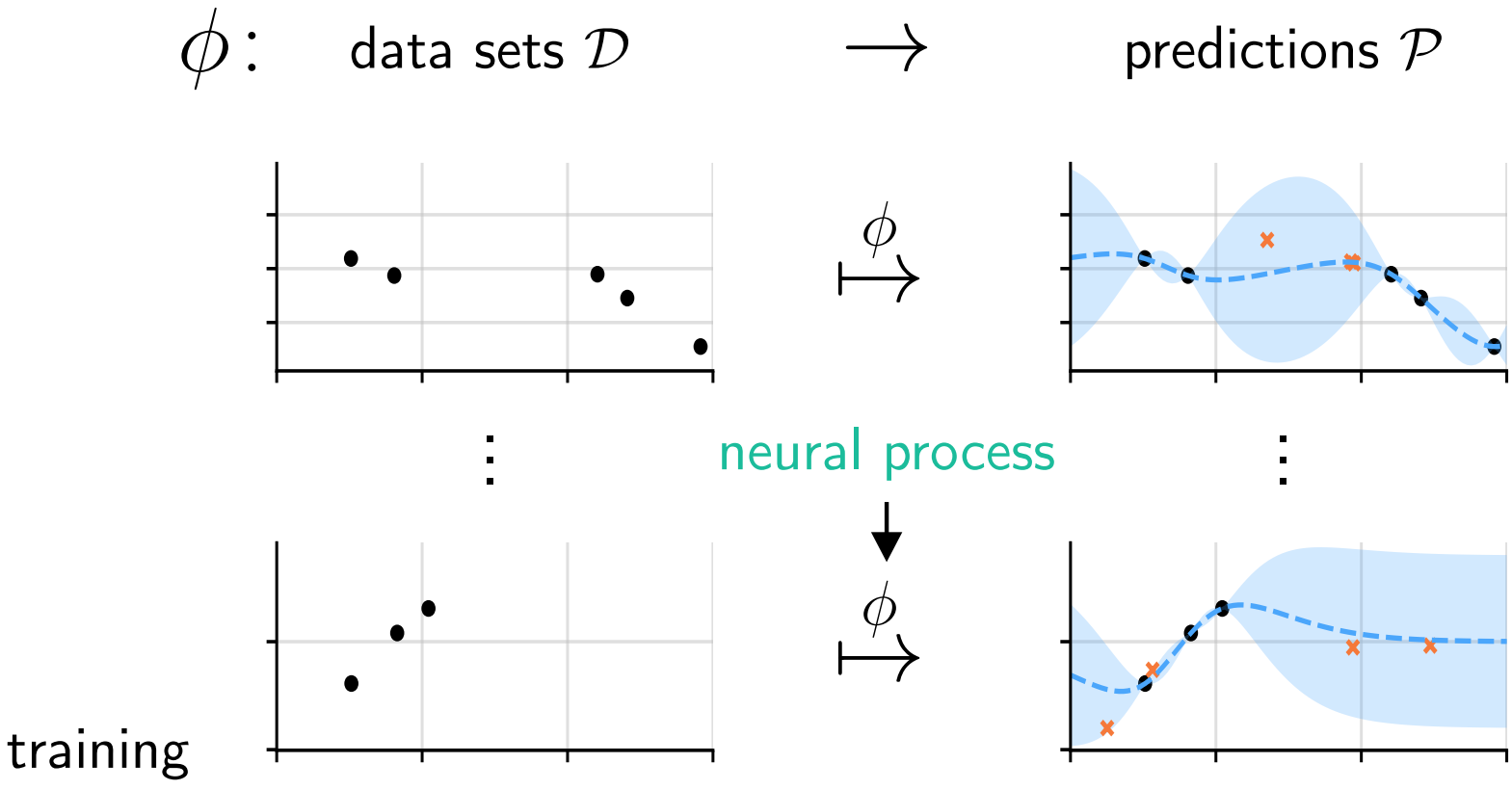
ϕ : data sets $\mathcal{D}$ $\longrightarrow$ predictions $\mathcal{P}$



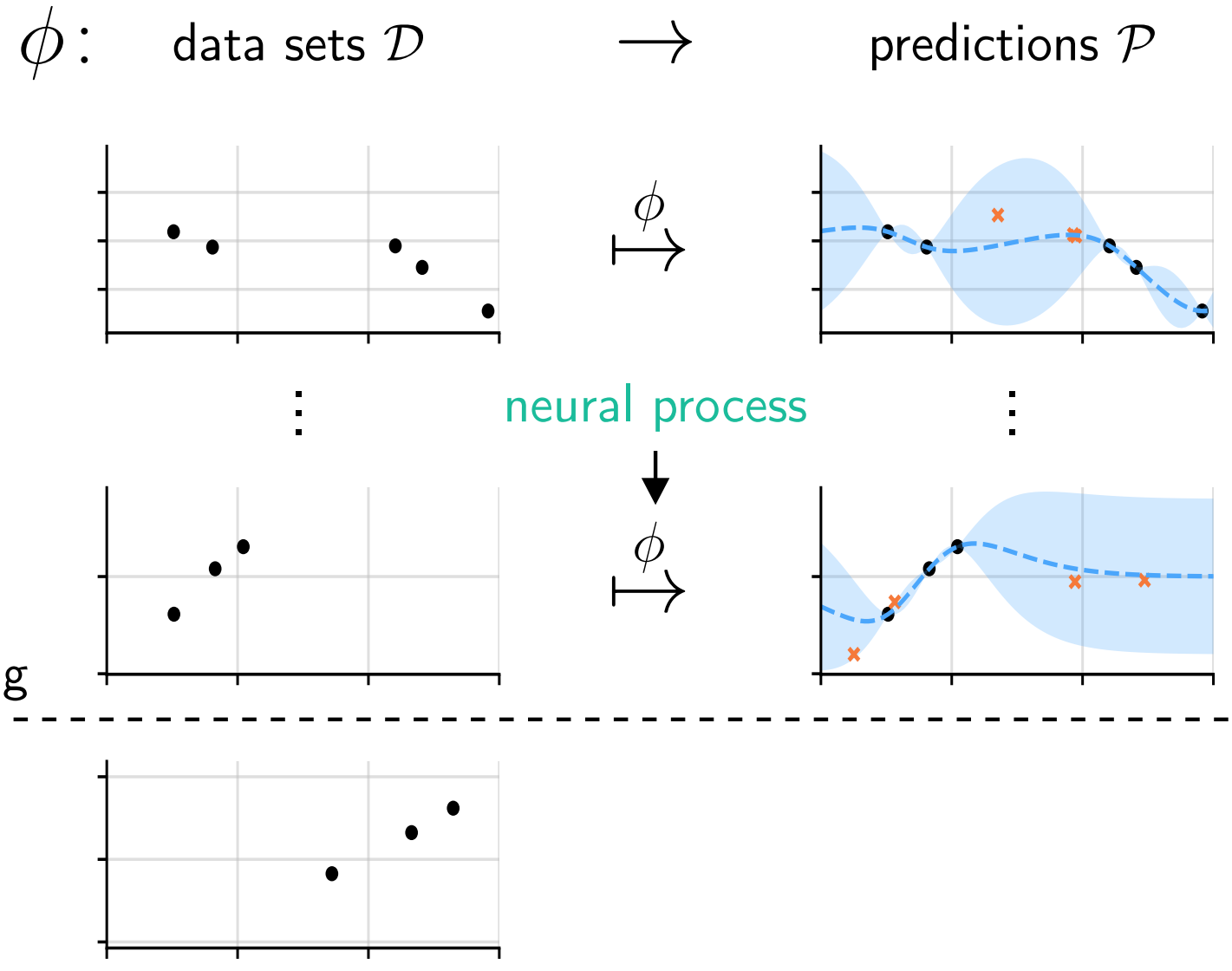
Setting: Learning to Predict



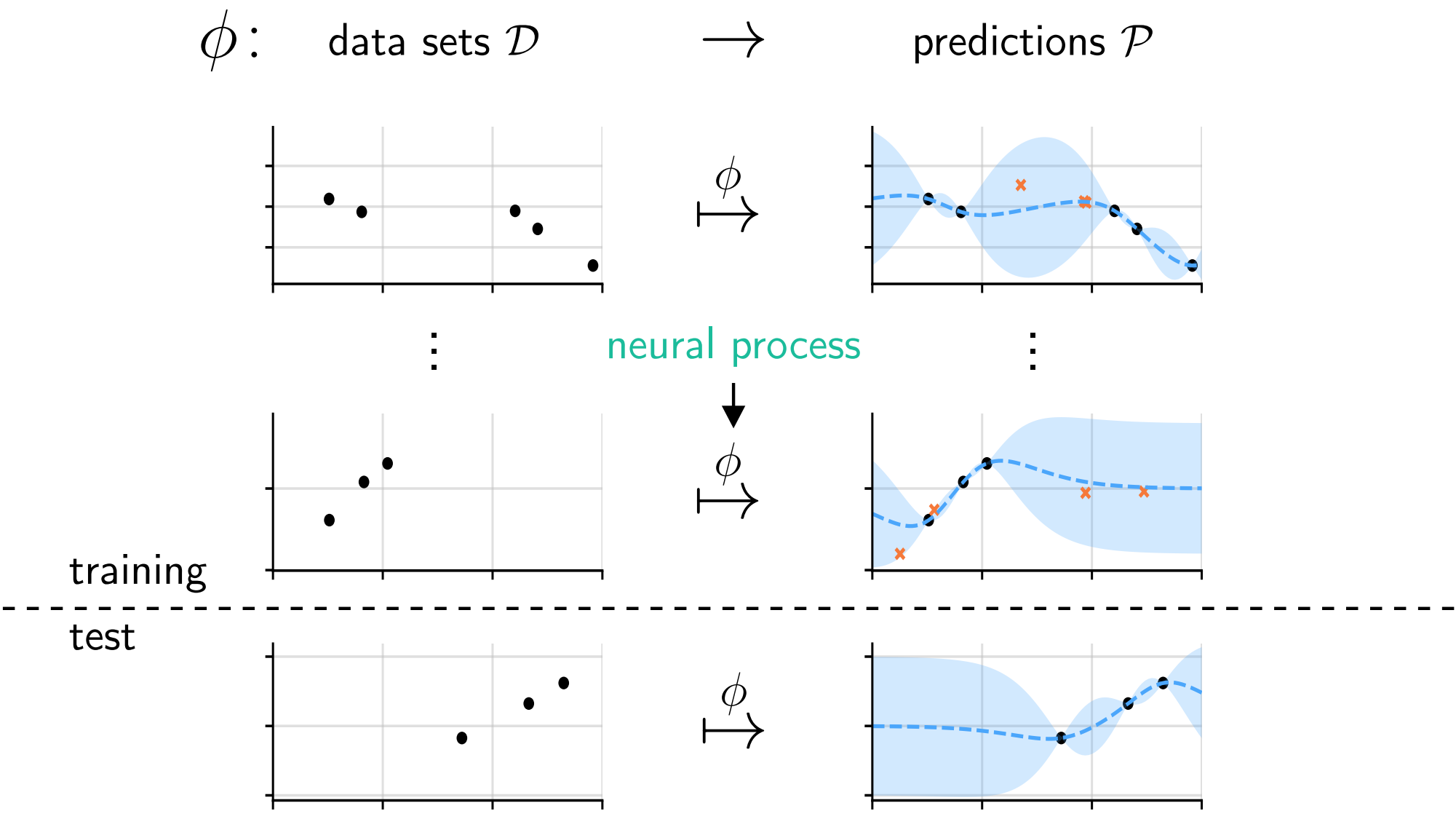
Setting: Learning to Predict



Setting: Learning to Predict



Setting: Learning to Predict



Neural Processes Are Many...

- Conditional Neural Process (Garnelo, Rosenbaum, et al., 2018)

Neural Processes Are Many...

- Conditional Neural Process (Garnelo, Rosenbaum, et al., 2018)
- Neural Process (Garnelo, Schwarz, et al., 2018)

Neural Processes Are Many...

- Conditional Neural Process (Garnelo, Rosenbaum, et al., 2018)
- Neural Process (Garnelo, Schwarz, et al., 2018)
- Attentive Neural Process (Kim et al., 2019)

Neural Processes Are Many...

- Conditional Neural Process (Garnelo, Rosenbaum, et al., 2018)
- Neural Process (Garnelo, Schwarz, et al., 2018)
- Attentive Neural Process (Kim et al., 2019)
- Functional Neural Process (Louizos et al., 2019)
- Sequential Neural Process (Singh et al., 2019)
- Convolutional Conditional Neural Process (Gordon et al., 2020)
- Convolutional Neural Process (Foong et al., 2020)

Neural Processes Are Many...

- Conditional Neural Process (Garnelo, Rosenbaum, et al., 2018)
- Neural Process (Garnelo, Schwarz, et al., 2018)
- Attentive Neural Process (Kim et al., 2019)
- Functional Neural Process (Louizos et al., 2019)
- Sequential Neural Process (Singh et al., 2019)
- Convolutional Conditional Neural Process (Gordon et al., 2020)
- Convolutional Neural Process (Foong et al., 2020)

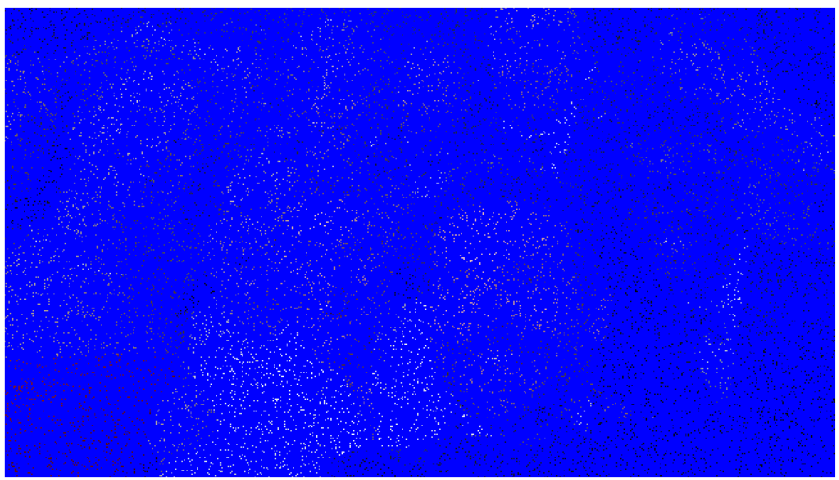


Figure from Gordon et al. (2020).

Neural Processes Are Many...

- Conditional Neural Process (Garnelo, Rosenbaum, et al., 2018)
- Neural Process (Garnelo, Schwarz, et al., 2018)
- Attentive Neural Process (Kim et al., 2019)
- Functional Neural Process (Louizos et al., 2019)
- Sequential Neural Process (Singh et al., 2019)
- Convolutional Conditional Neural Process (Gordon et al., 2020)
- Convolutional Neural Process (Foong et al., 2020)

Figure from Gordon et al. (2020).

Neural Processes Are Many... (2)

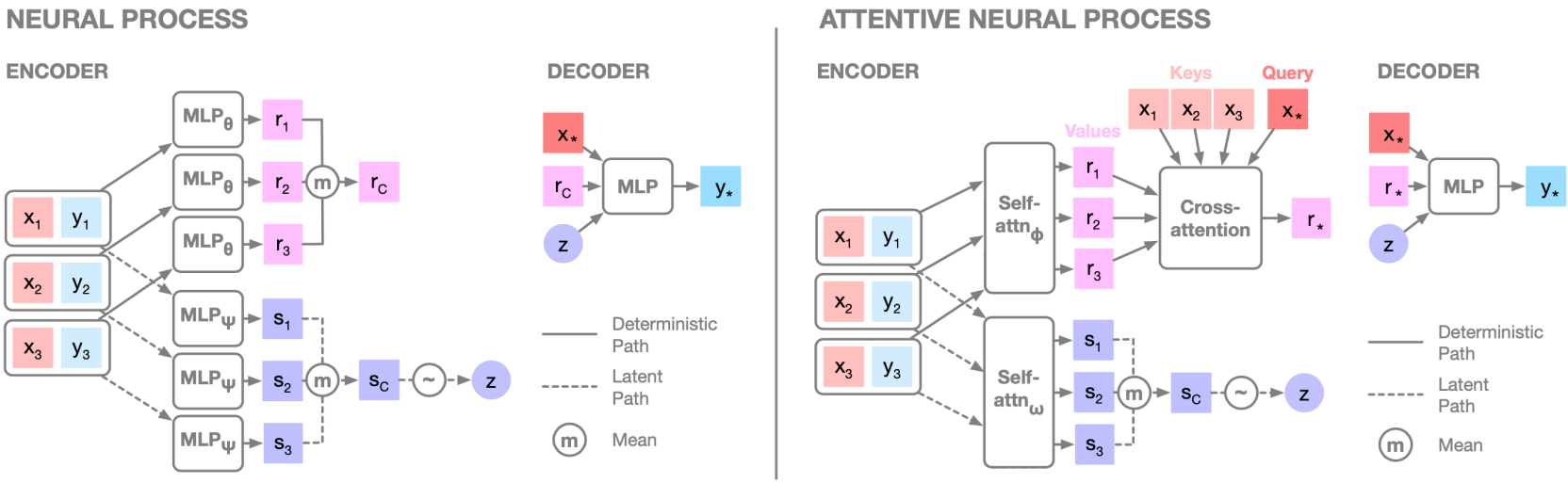


Figure from Kim et al. (2019).

Neural Processes Are Many... (2)

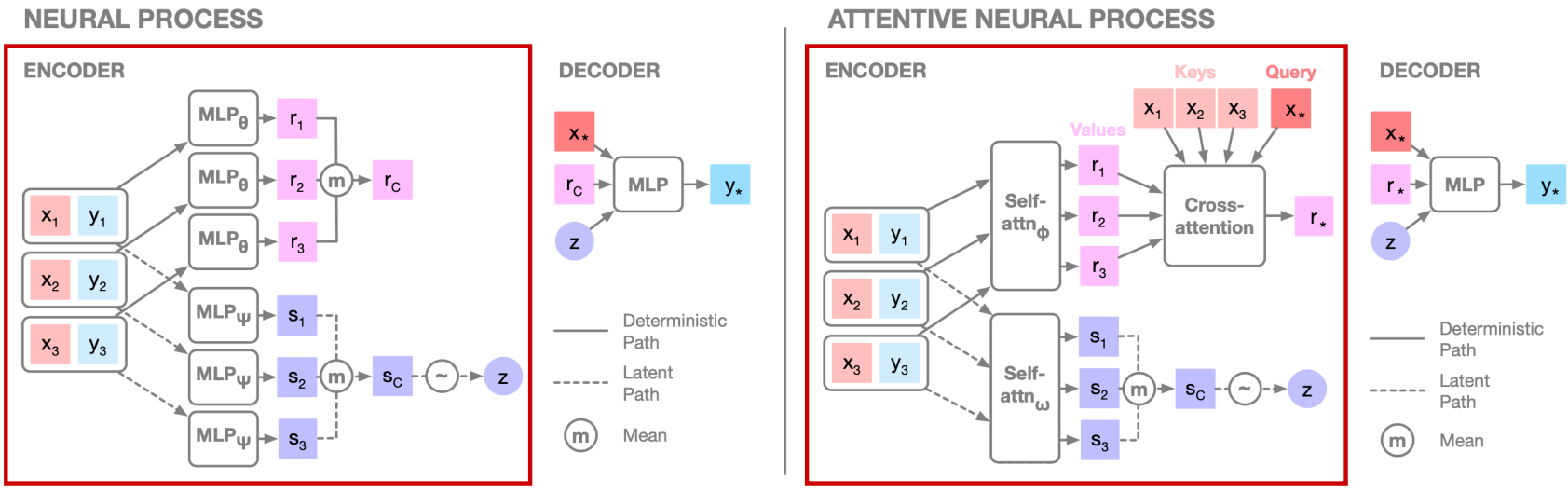


Figure from Kim et al. (2019).

Neural Processes Are Many... (2)

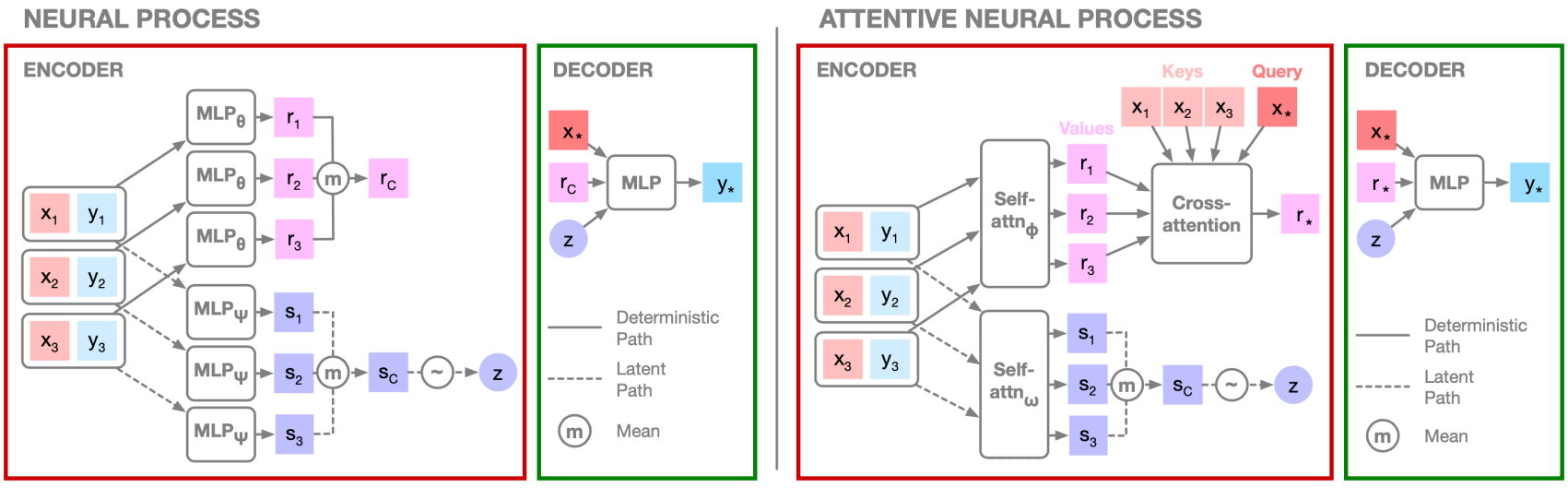


Figure from Kim et al. (2019).

Neural Processes Are Many... (2)

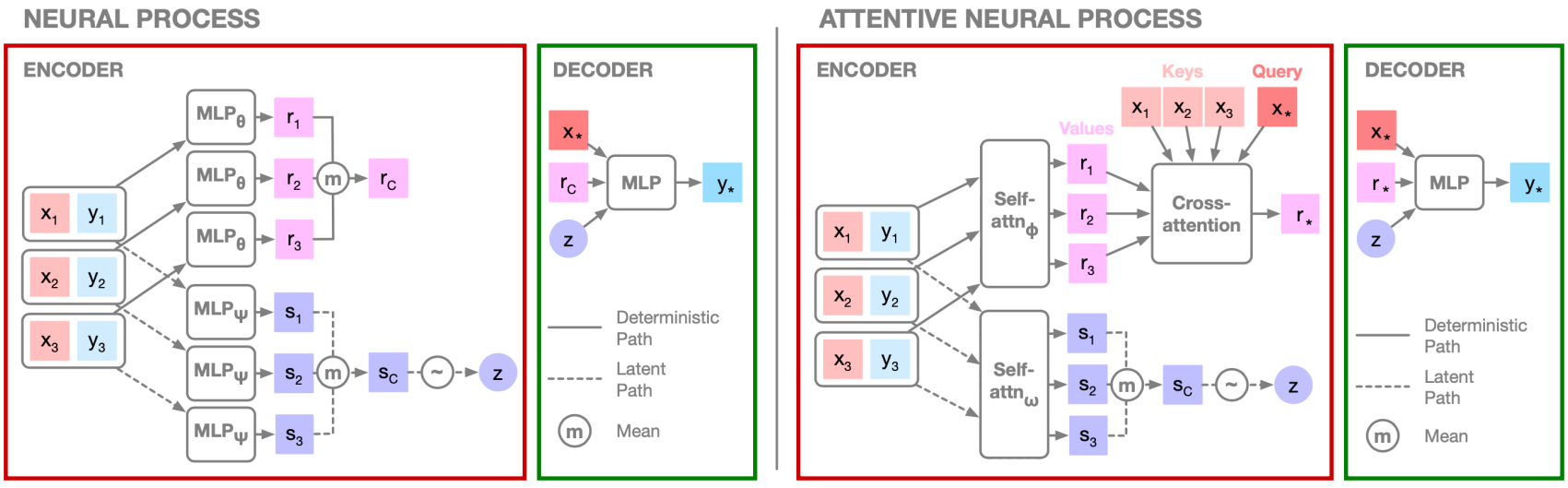


Figure from Kim et al. (2019).

- Implementations similar, but differ in details.

Neural Processes Are Many... (2)

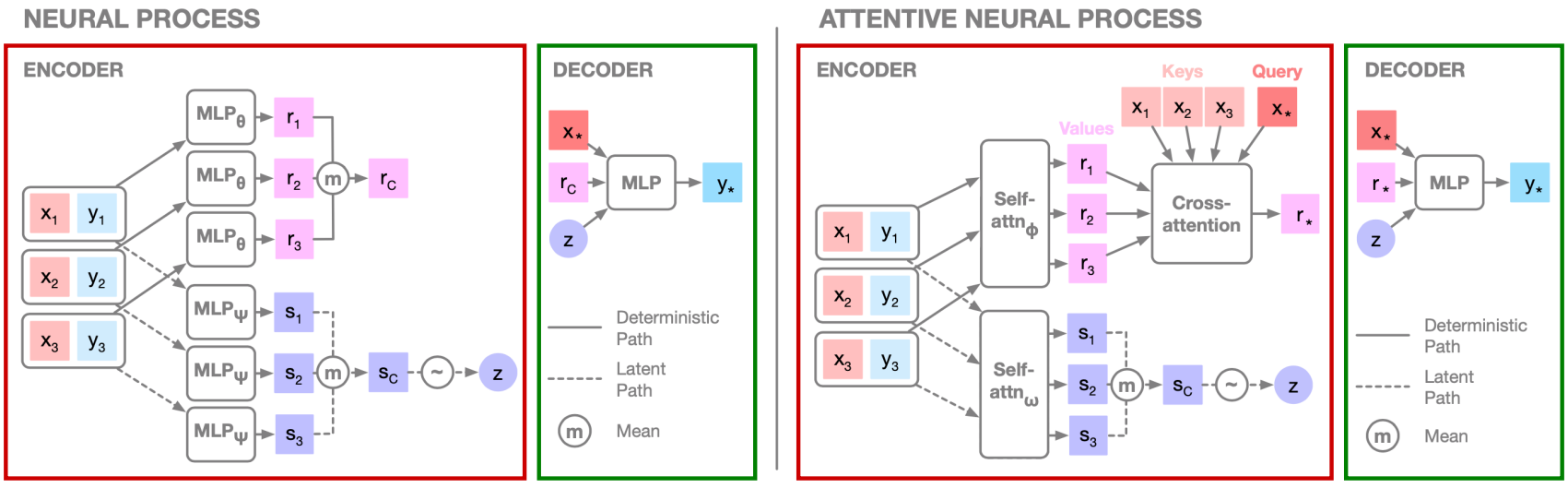
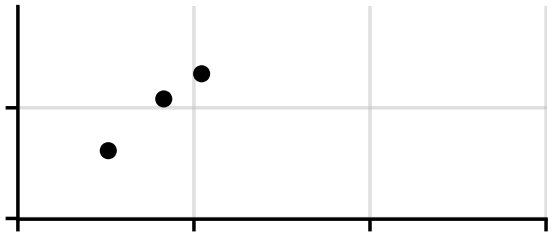


Figure from Kim et al. (2019).

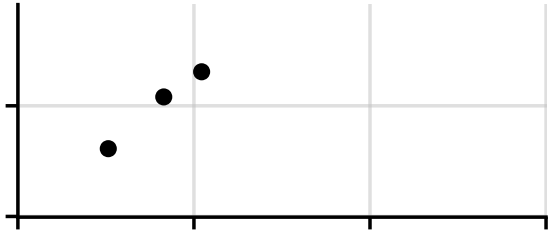
- Implementations similar, but differ in details.
- Calls for a unifying framework: `NeuralProcesses.jl`.

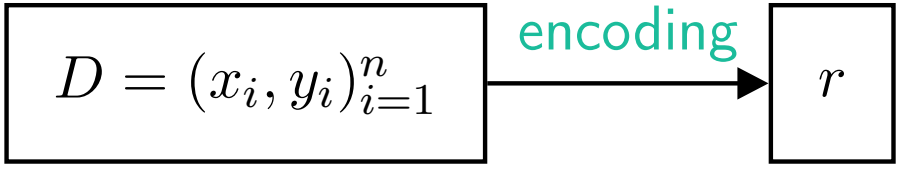
Encoder–Decoder Architectures



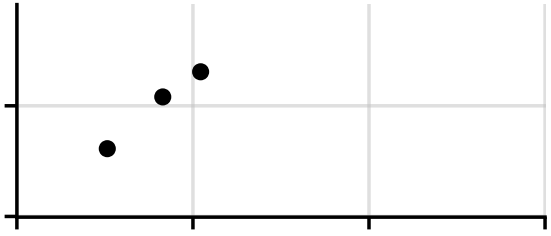
$$D = (x_i, y_i)_{i=1}^{n}$$

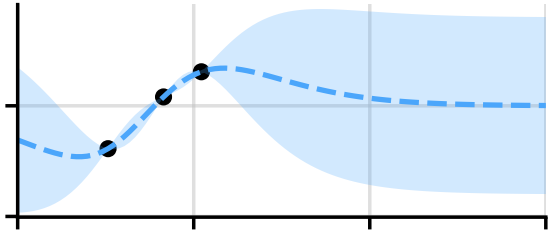
Encoder–Decoder Architectures

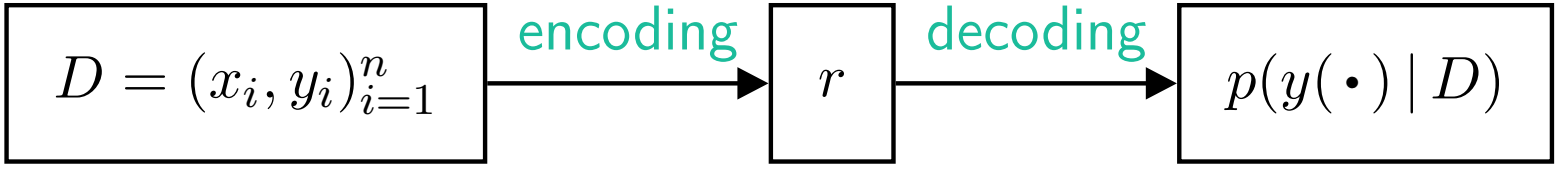




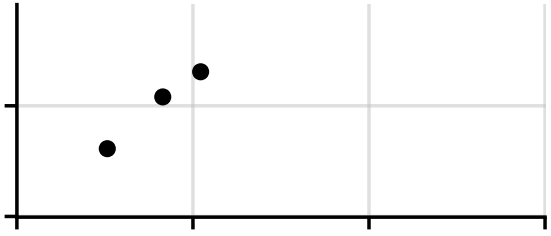
Encoder–Decoder Architectures

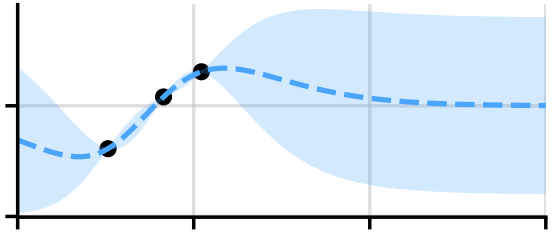


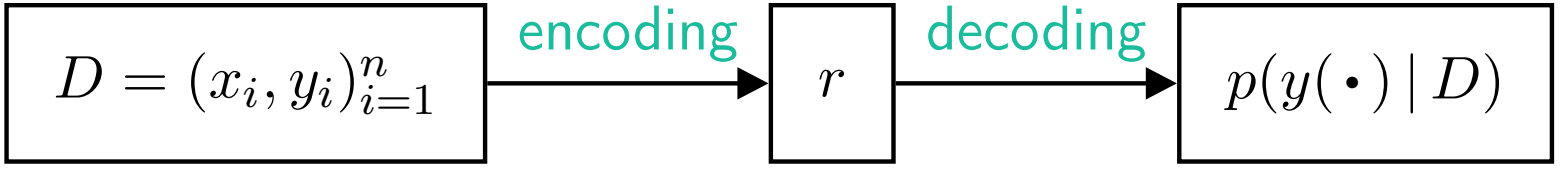




Encoder–Decoder Architectures

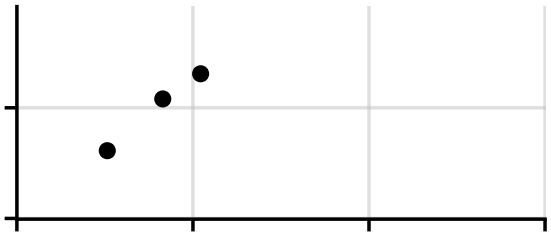


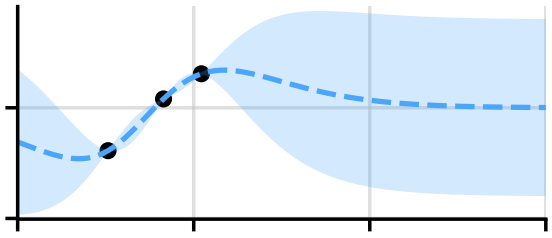


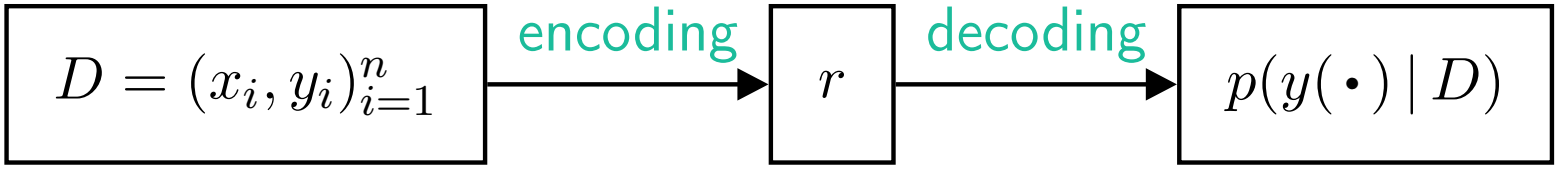


- Need a common representation.

Encoder–Decoder Architectures

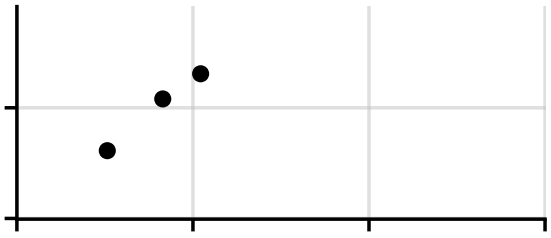


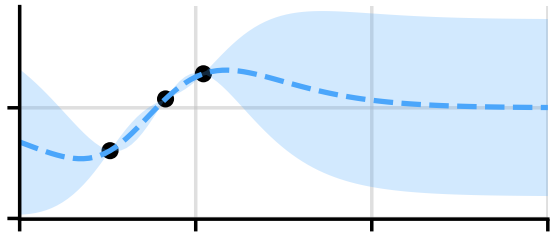


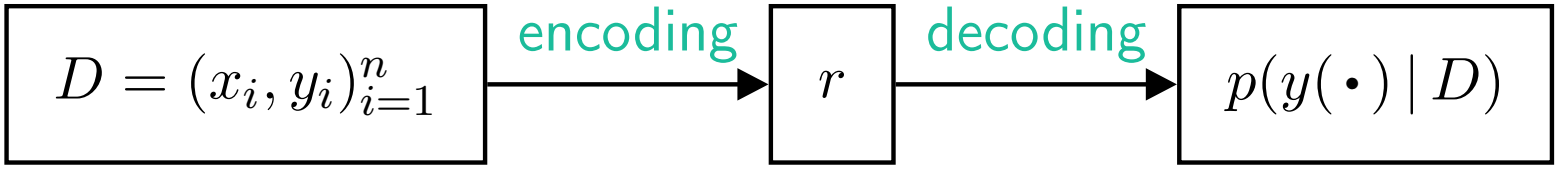


- Need a common representation.
- Data is a function: $D\colon \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_n\}$.

Encoder–Decoder Architectures

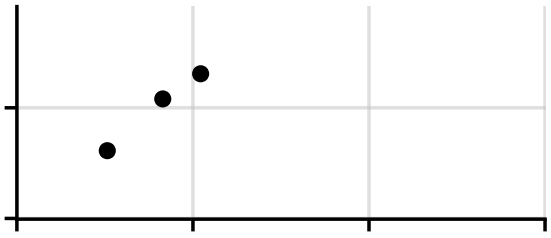


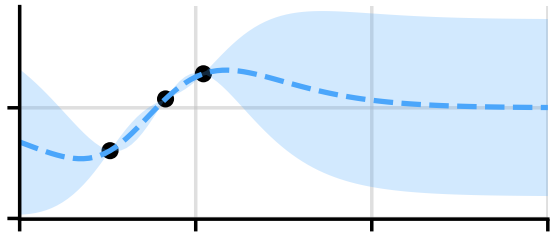


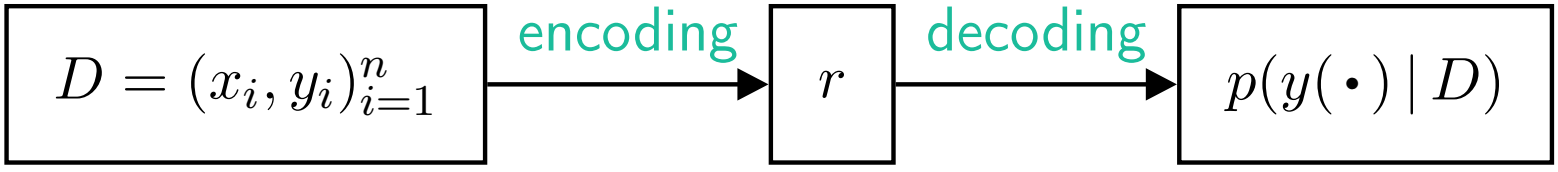


- Need a common representation.
- Data is a function: $D\colon \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_n\}$.
- Prediction is a function: $p(y(\bullet) \mid D)\colon \mathbb{R} \to \mathbb{R} \times \mathbb{R}_{>0}$ (μ and σ).

Encoder–Decoder Architectures

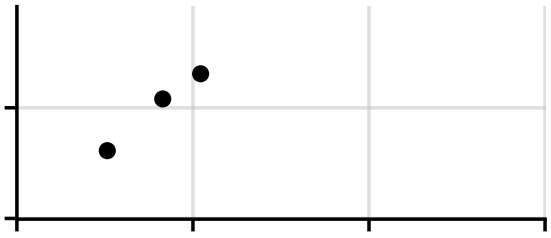


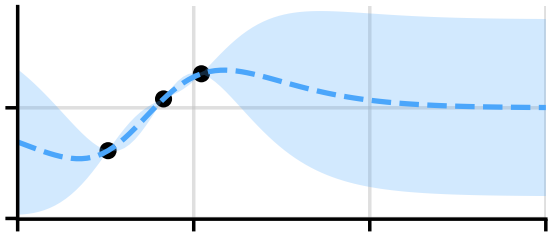


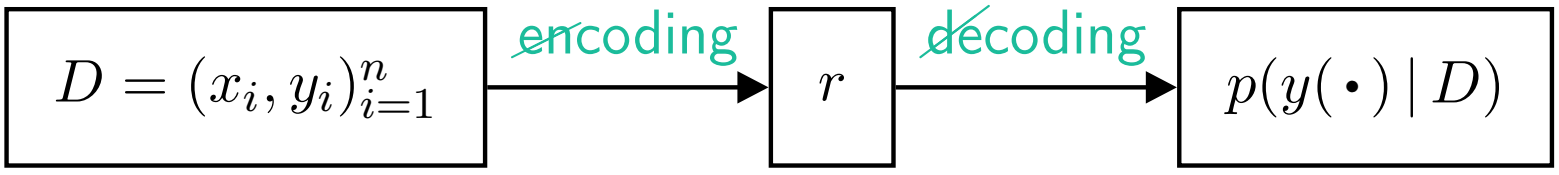


- Need a common representation: functions!
- Data is a function: $D\colon \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_n\}$.
- Prediction is a function: $p(y(\bullet) \mid D)\colon \mathbb{R} \to \mathbb{R} \times \mathbb{R}_{>0}$ (μ and σ).

Encoder–Decoder Architectures

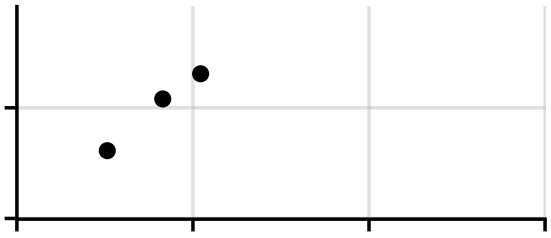


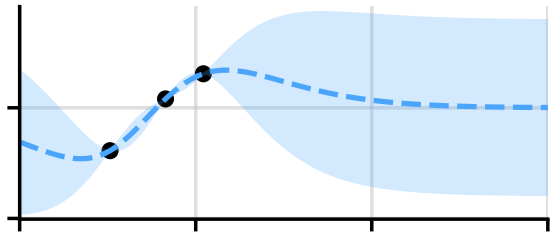


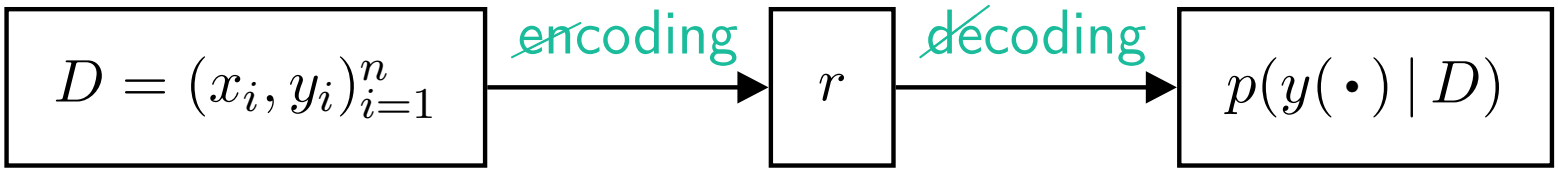


- Need a common representation: functions!
- Data is a function: $D\colon \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_n\}$.
- Prediction is a function: $p(y(\bullet) \mid D)\colon \mathbb{R} \to \mathbb{R} \times \mathbb{R}_{>0}$ (μ and σ).
- $*$coding is then a transformation of functions

Encoder–Decoder Architectures







- Need a common representation: functions!
- Data is a function: $D\colon \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_n\}$.
- Prediction is a function: $p(y(\bullet) \mid D)\colon \mathbb{R} \to \mathbb{R} \times \mathbb{R}_{>0}$ (μ and σ).
- $*$coding is then a transformation of functions

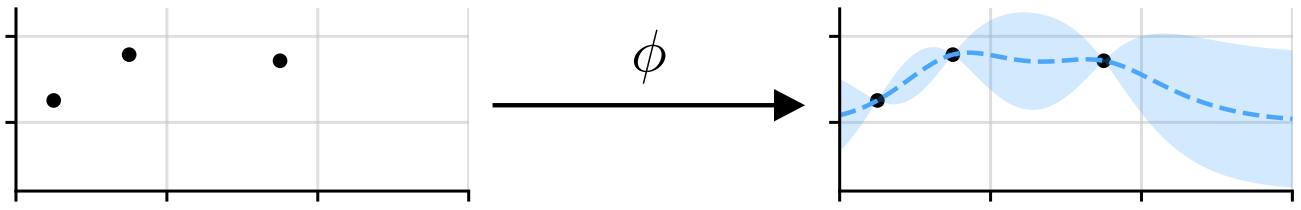
$\Rightarrow$ fundamental abstraction of `NeuralProcesses.jl`.

The Convolutional Conditional Neural Process

- A formulation that incorporates translation equivariance:

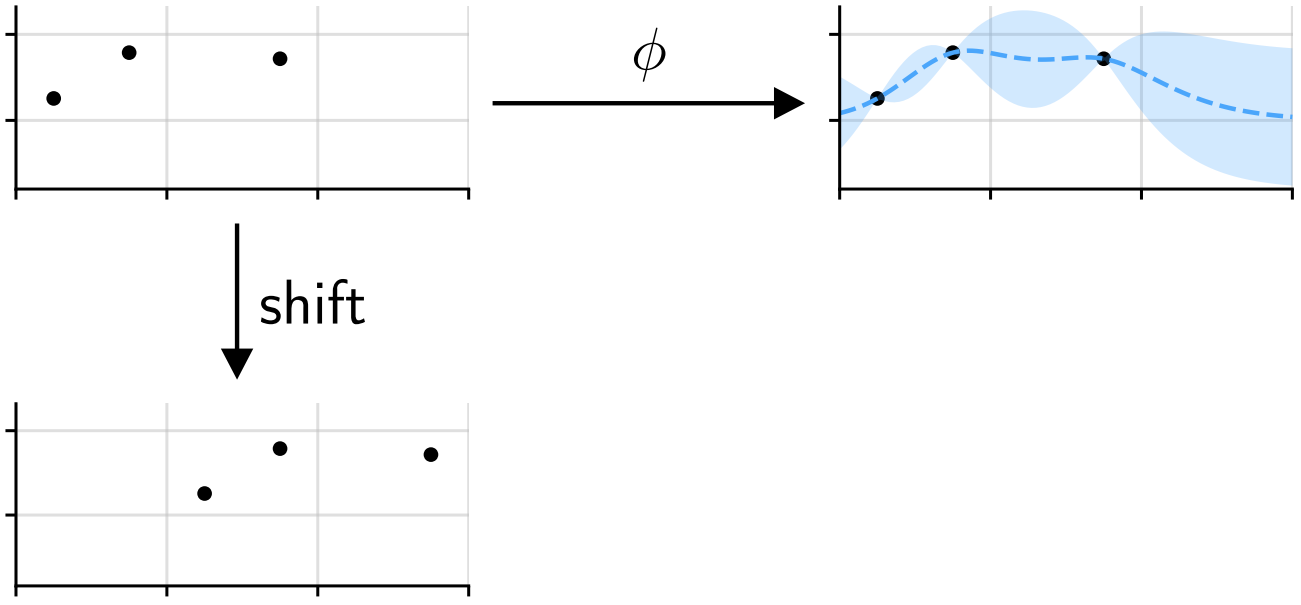
The Convolutional Conditional Neural Process

- A formulation that incorporates translation equivariance:



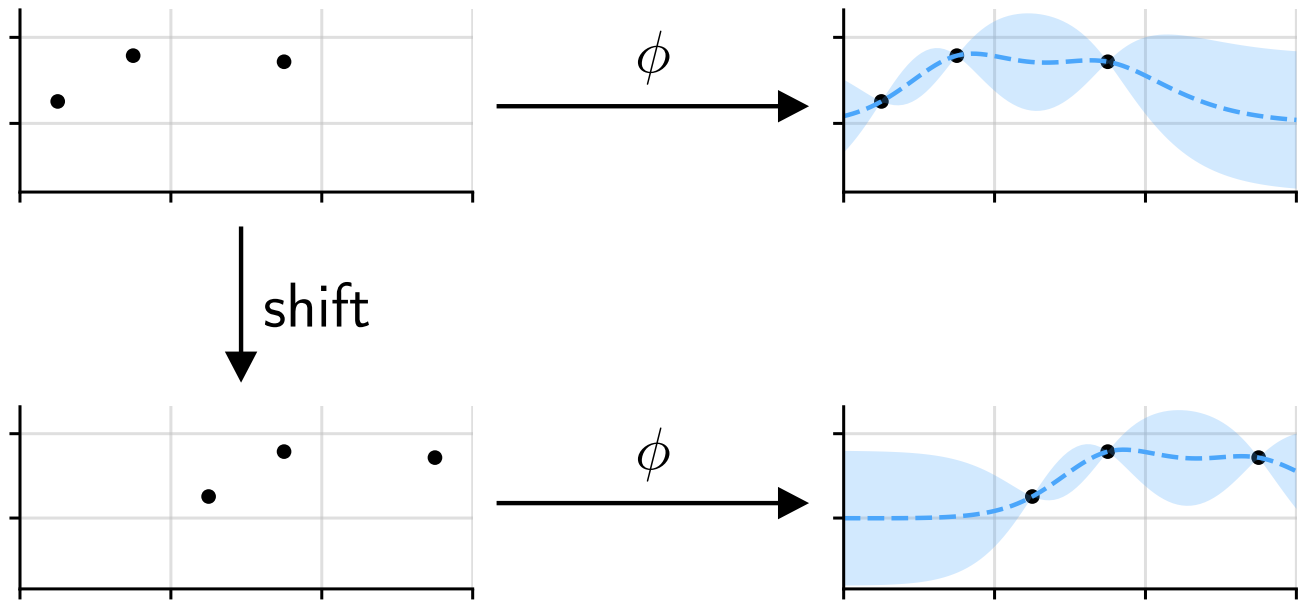
The Convolutional Conditional Neural Process

- A formulation that incorporates translation equivariance:



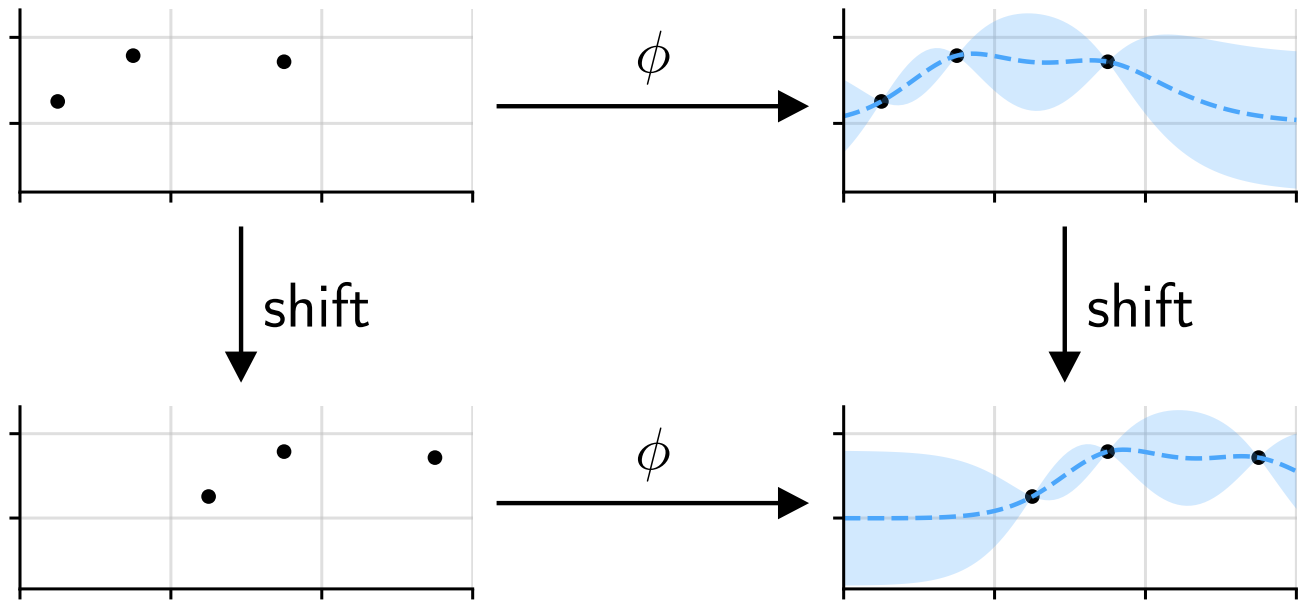
The Convolutional Conditional Neural Process

- A formulation that incorporates translation equivariance:



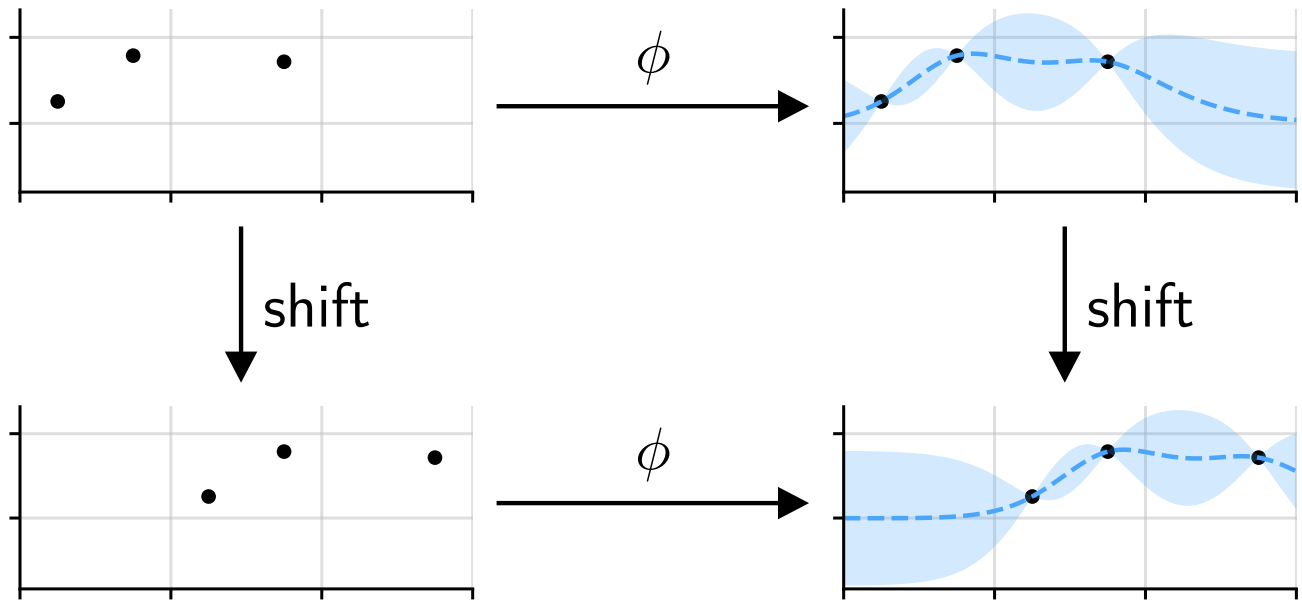
The Convolutional Conditional Neural Process

- A formulation that incorporates translation equivariance:



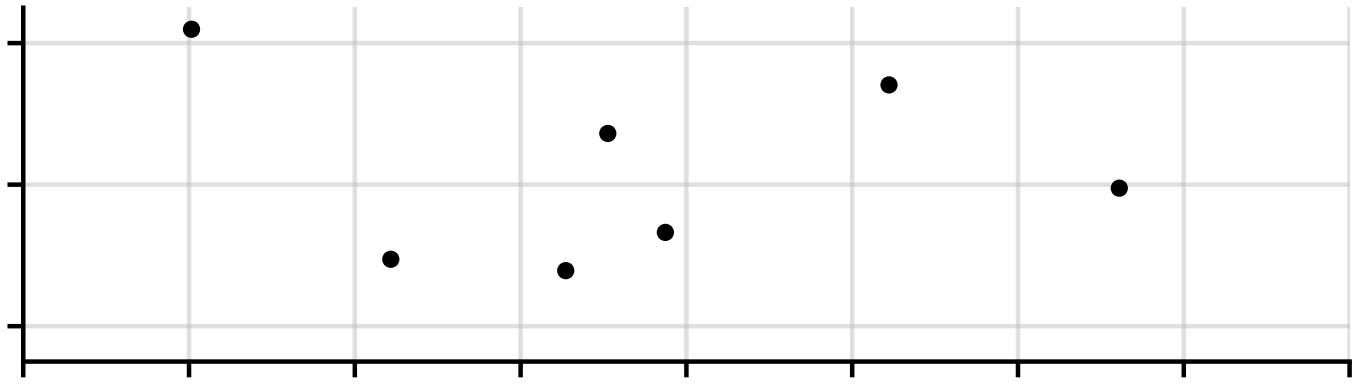
The Convolutional Conditional Neural Process

- A formulation that incorporates translation equivariance:

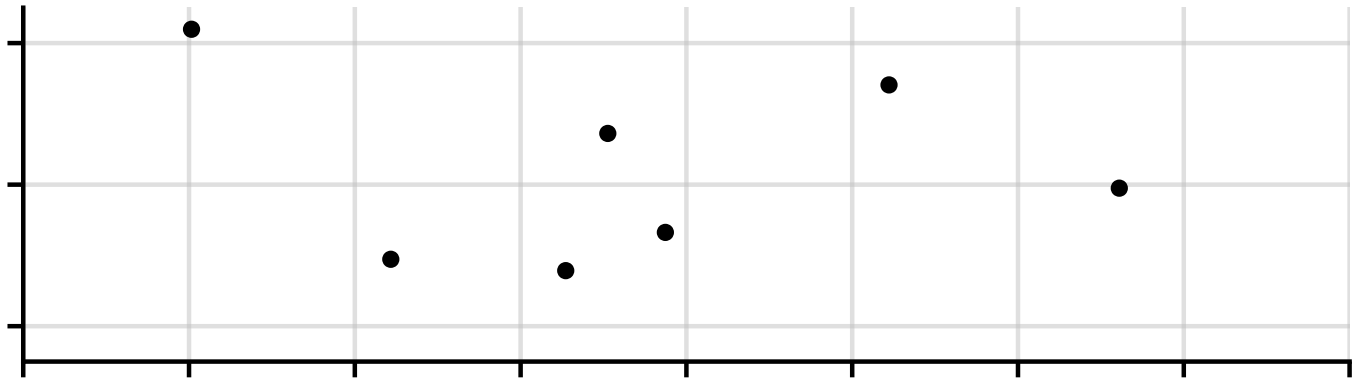


- Leverages the parameter efficiency of CNNs: `Flux.jl`!

The ConvCNP with `NeuralProcesses.jl`

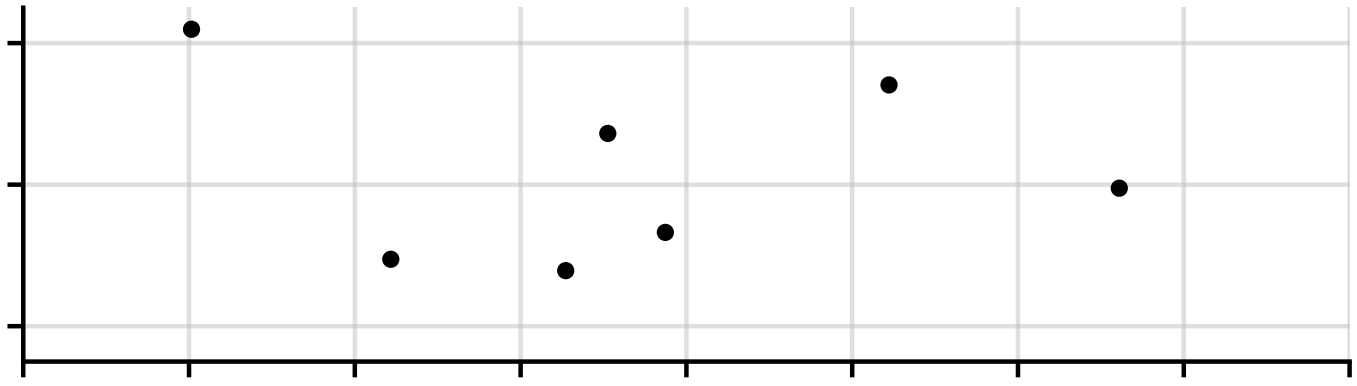


The ConvCNP with `NeuralProcesses.jl`



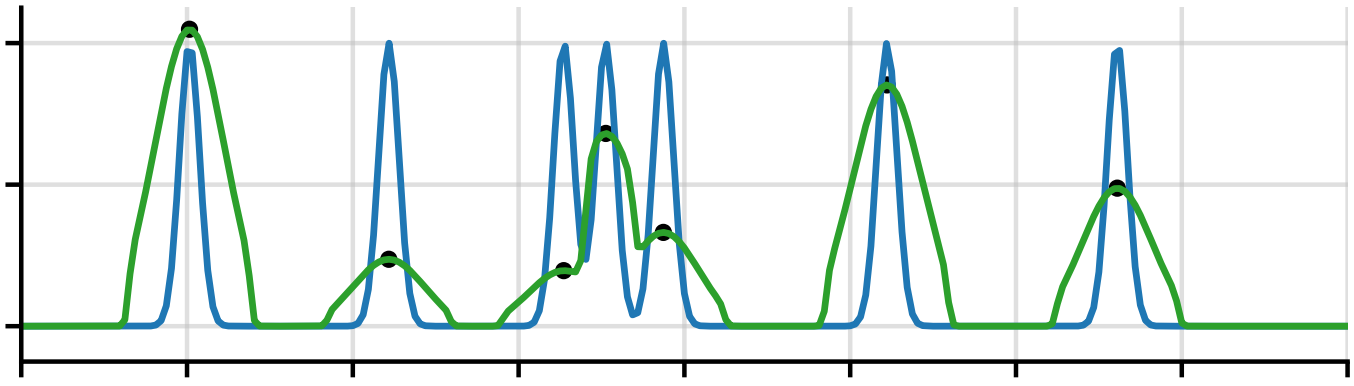
```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),

)
```

The ConvCNP with `NeuralProcesses.jl`



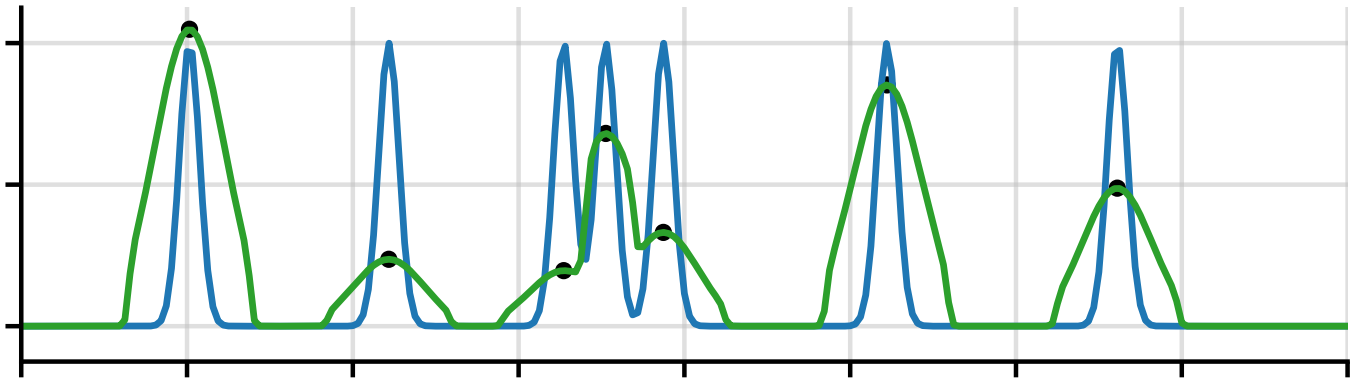
```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

The ConvCNP with `NeuralProcesses.jl`



```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

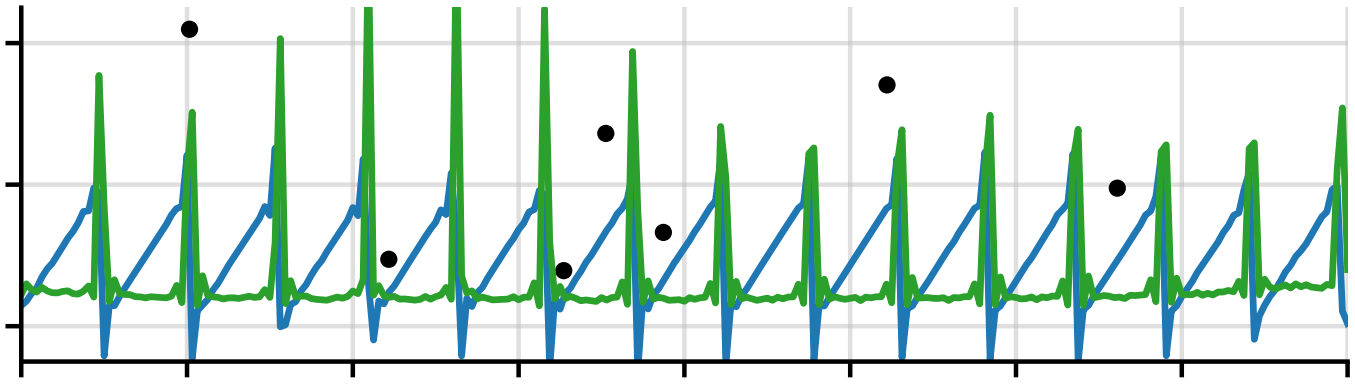
The ConvCNP with `NeuralProcesses.jl`



```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

```
decoder = Chain(
    build_conv(...),

)
```

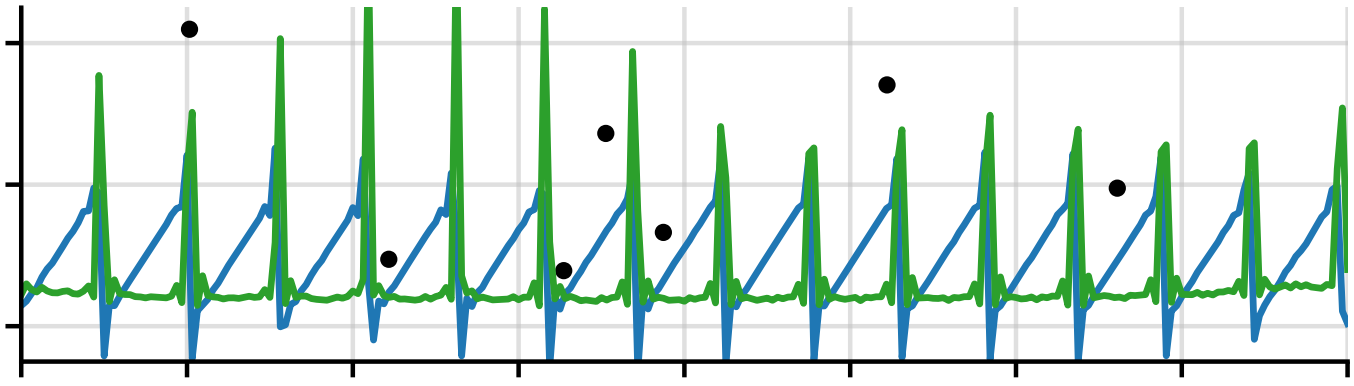
The ConvCNP with `NeuralProcesses.jl`



```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

```
decoder = Chain(
    build_conv(...),

)
```

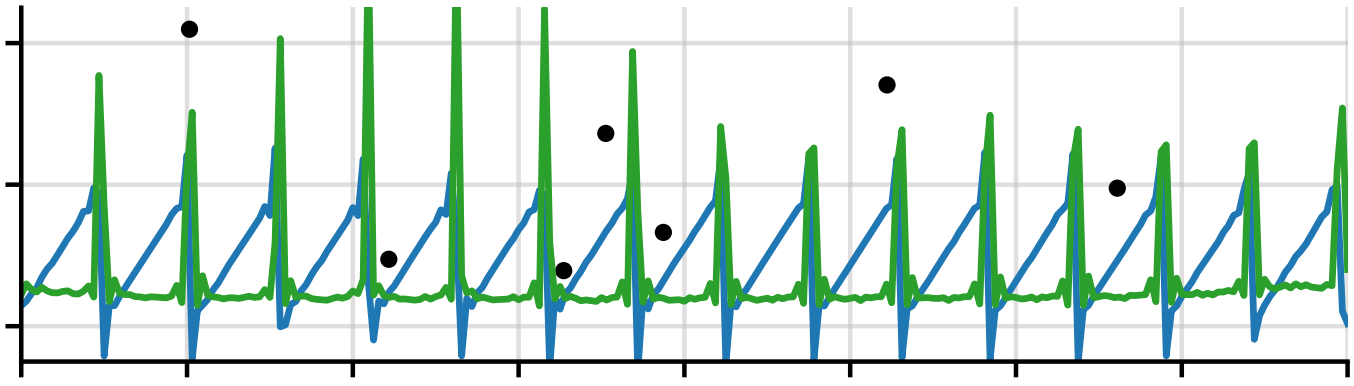
The ConvCNP with `NeuralProcesses.jl`



```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

```
decoder = Chain(
    build_conv(...),
    set_conv(...),
    HeterogeneousGaussian()
)
```

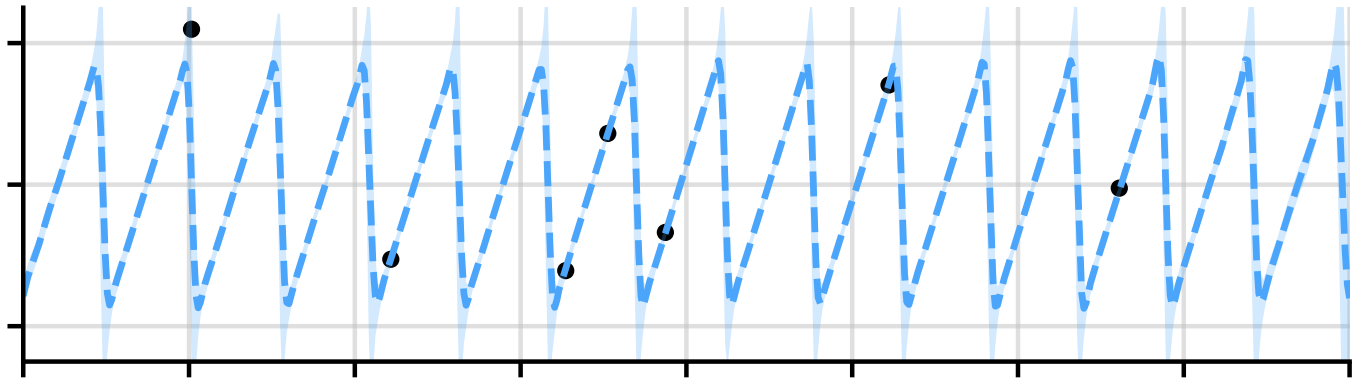
The ConvCNP with `NeuralProcesses.jl`



```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

```
decoder = Chain(
    build_conv(...),
    set_conv(...),
    HeterogeneousGaussian()
)

convcnp = Model(encoder, decoder)
```

The ConvCNP with `NeuralProcesses.jl`

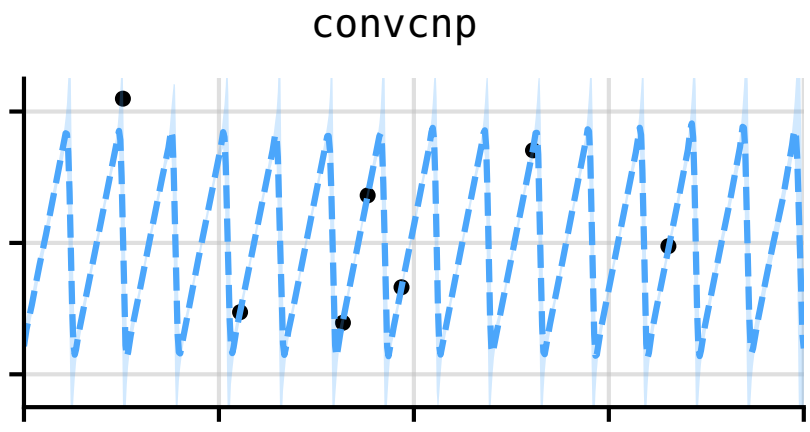


```
encoder = FunctionalCoder(
    UniformDiscretisation1D(...),
    Chain(
        # See Gordon et al. (2020).
        set_conv(...),
        Deterministic()
    )
)
```

```
decoder = Chain(
    build_conv(...),
    set_conv(...),
    HeterogeneousGaussian()
)

convcnp = Model(encoder, decoder)
```

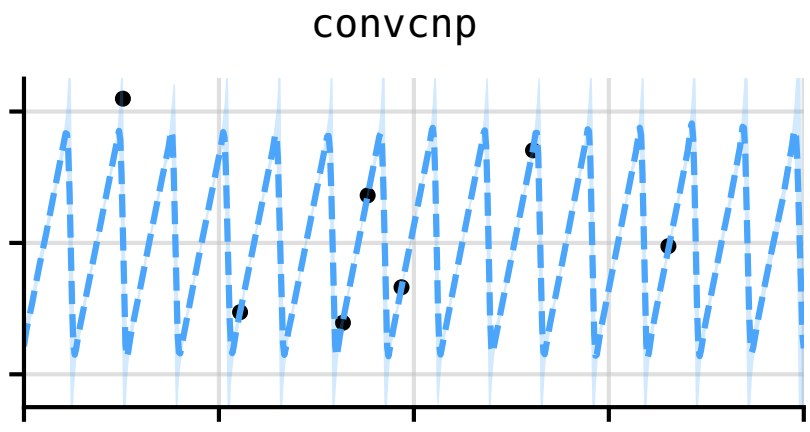
NeuralProcesses.jl


```
# Gordon et al. (2020)
convcnp = Model(
    FunctionalCoder(
        UniformDiscretisation1D(...),
        Chain(
            set_conv(...),
            Deterministic()
        )
    ),
    Chain(
       build_conv(...),
       set_conv(...),
       HeterogeneousGaussian()
    )
)
```



NeuralProcesses.jl

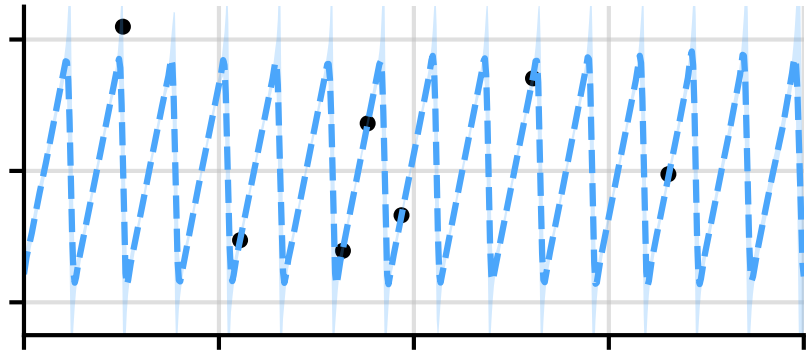

```
# Gordon et al. (2020)
convcnp = Model(
    FunctionalCoder(
        UniformDiscretisation1D(...),
        Chain(
            set_conv(...),
            Deterministic()
        )
    ),
    Chain(
       build_conv(...),
       set_conv(...),
       HeterogeneousGaussian()
    )
)
```



NeuralProcesses.jl


```
# Foong et al. (2020)
convnp = Model(
    FunctionalCoder(
        UniformDiscretisation1D(...),
        Chain(
            set_conv(...),
            build_conv(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
       build_conv(...),
       set_conv(...),
       HeterogeneousGaussian()
    )
)
```

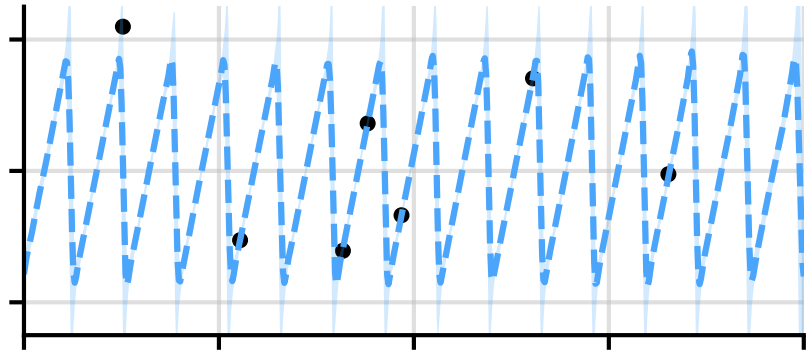
convcnp



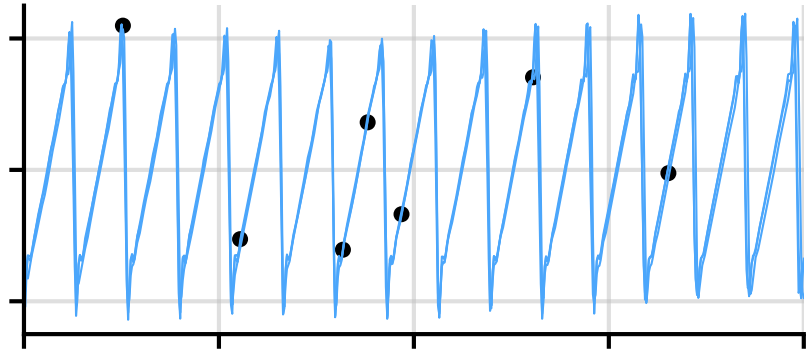
NeuralProcesses.jl


```
# Foong et al. (2020)
convnp = Model(
    FunctionalCoder(
        UniformDiscretisation1D(...),
        Chain(
            set_conv(...),
            build_conv(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
       build_conv(...),
       set_conv(...),
       HeterogeneousGaussian()
    )
)
```

convcnp



convnp



NeuralProcesses.jl


```
# Foong et al. (2020)
convnp = Model(
    FunctionalCoder(
        UniformDiscretisation1D(...),
        Chain(
            set_conv(...),
            build_conv(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
       build_conv(...),
       set_conv(...),
       HeterogeneousGaussian()
    )
)
```

- Composable building blocks

NeuralProcesses.jl


```
# Garnelo et al. (2018)
np = Model(
    Parallel(
        Chain(
           InputsEncoder(),
           Deterministic()
        ),
        Chain(
           MLPEncoder(...),
           Deterministic()
        ),
        Chain(
           MLPEncoder(...),
           HeterogeneousGaussian()
        )
    ),
    Chain(
       Materialise(),
       batched_mlp(...),
       HeterogeneousGaussian()
    )
)
```

- Composable building blocks

NeuralProcesses.jl


```
# Garnelo et al. (2018)
np = Model(
    Parallel(
        Chain(
           InputsEncoder(),
           Deterministic()
        ),
        Chain(
           MLPEncoder(...),
           Deterministic()
        ),
        Chain(
           MLPEncoder(...),
           HeterogeneousGaussian()
        )
    ),
    Chain(
       Materialise(),
       batched_mlp(...),
       HeterogeneousGaussian()
    )
)
```

- Composable building blocks
- Use `Chain` and `Parallel` to glue things together

NeuralProcesses.jl


```
# Garnelo et al. (2018)
np = Model(
    Parallel(
        Chain(
            InputsEncoder(),
            Deterministic()
        ),
        Chain(
            MLPEncoder(...),
            Deterministic()
        ),
        Chain(
            MLPEncoder(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
        Materialise(),
        batched_mlp(...),
        HeterogeneousGaussian()
    )
)
```

- Composable building blocks
- Use `Chain` and `Parallel` to glue things together
- Inference and learning are automatic

NeuralProcesses.jl


```
# Garnelo et al. (2018)
np = Model(
    Parallel(
        Chain(
            InputsEncoder(),
            Deterministic()
        ),
        Chain(
            MLPEncoder(...),
            Deterministic()
        ),
        Chain(
            MLPEncoder(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
        Materialise(),
        batched_mlp(...),
        HeterogeneousGaussian()
    )
)
```

- Composable building blocks
- Use `Chain` and `Parallel` to glue things together
- Inference and learning are automatic

NeuralProcesses.jl


```
# Kim et al. (2019)
anp = Model(
    Parallel(
        Chain(
            InputsEncoder(),
            Deterministic()
        ),
        Chain(
            attention(...),
            Deterministic()
        ),
        Chain(
            MLPEncoder(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
        Materialise(),
        batched_mlp(...),
        HeterogeneousGaussian()
    )
)
```

- Composable building blocks
- Use `Chain` and `Parallel` to glue things together
- Inference and learning are automatic

NeuralProcesses.jl


```
# Kim et al. (2019)
anp = Model(
    Parallel(
        Chain(
            InputsEncoder(),
            Deterministic()
        ),
        Chain(
            attention(...),
            Deterministic()
        ),
        Chain(
            MLPEncoder(...),
            HeterogeneousGaussian()
        )
    ),
    Chain(
        Materialise(),
        batched_mlp(...),
        HeterogeneousGaussian()
    )
)
```

- Composable building blocks
- Use `Chain` and `Parallel` to glue things together
- Inference and learning are automatic

Thank you for listening :)

Appendix

References

Foong, A. Y. K., Bruinsma, W. P., Gordon, J., Dubois, Y., Requeima, J., & Turner, R. E. (2020). Meta-learning stationary stochastic process prediction with convolutional neural processes. *arXiv preprint arXiv:2007.01332*. eprint: https://arxiv.org/abs/2007.01332

Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., ... Eslami, S. M. A. (2018). Conditional neural processes. *arXiv preprint arXiv:1807.01613*. eprint: https://arxiv.org/abs/1807.01613

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., & Teh, Y. W. (2018). Neural processes. *arXiv preprint arXiv:1807.01622*. eprint: https://arxiv.org/abs/1807.01622

References (2)

Gordon, J., Bruinsma, W. P., Foong, A. Y. K., Requeima, J., Dubois, Y., & Turner, R. E. (2020). Convolutional conditional neural processes. *International Conference on Learning Representations (ICLR), 8th*. Retrieved from https://openreview.net/forum?id=Skey4eBYPS

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., ... Teh, Y. W. (2019). Attentive neural processes. *arXiv preprint arXiv:1901.05761*. eprint: https://arxiv.org/abs/1901.05761

Louizos, C., Shi, X., Schutte, K., & Welling, M. (2019). The functional neural process. *arXiv preprint arXiv:1906.08324*. eprint: https://arxiv.org/abs/1906.08324

Singh, G., Yoon, J., Son, Y., & Ahn, S. (2019). Sequential neural processes. *arXiv preprint arXiv:1906.10264*. eprint: https://arxiv.org/abs/1906.10264