Neural Processes

Marta Garnelo [1] Jonathan Schwarz [1] Dan Rosenbaum [1] Fabio Viola [1] Danilo J. Rezende [1] S. M. Ali Eslami [1] Yee Whye Teh [1]

Abstract

A neural network (NN) is a parameterised function that can be tuned via gradient descent to approximate a labelled collection of data with high precision. A Gaussian process (GP), on the other hand, is a probabilistic model that defines a *distribution* over possible functions, and is updated in light of data via the rules of probabilistic inference. GPs are probabilistic, data-efficient and flexible, however they are also computationally intensive and thus limited in their applicability. We introduce a class of neural latent variable models which we call *Neural Processes* (NPs), combining the best of both worlds. Like GPs, NPs define distributions over functions, are capable of rapid adaptation to new observations, and can estimate the uncertainty in their predictions. Like NNs, NPs are computationally efficient during training and evaluation but also learn to adapt their priors to data. We demonstrate the performance of NPs on a range of learning tasks, including regression and optimisation, and compare and contrast with related models in the literature.

1. Introduction

Function approximation lies at the core of numerous problems in machine learning and one approach that has been exceptionally popular for this purpose over the past decade are deep neural networks. At a high level neural networks constitute black-box function approximators that learn to parameterise a single function from a large number of training data points. As such, the majority of the workload of a networks falls on the training phase while the evaluation and testing phases are reduced to quick forward-passes. Although high test-time performance is valuable for many real-world applications, the fact that network outputs cannot be updated after training may be undesirable. Meta-learning, for example, is an increasingly popular field of research that addresses exactly this limitation (Sutskever et al., 2014; Wang et al., 2016; Vinyals et al., 2016; Finn et al., 2017).

As an alternative to using neural networks one can also perform inference on a stochastic process in order to carry out function regression. The most common instantiation of this approach is a Gaussian process (GP), a model with complimentary properties to those of neural networks: GPs do not require a costly training phase and can carry out inference about the underlying ground truth function conditioned on some observations, which renders them very flexible at test-time. In addition GPs represent infinitely many different functions at locations that have not been observed thereby capturing the uncertainty over their predictions given some observations. However, GPs are computationally expensive: in their original formulation they scale cubically with respect to the number of data points, and current state of the art approximations still scale quadratically (Quiñonero-Candela & Rasmussen, 2005). Furthermore, the available kernels are usually restricted in their functional form and an additional optimisation procedure is required to identify the most suitable kernel, as well as its hyperparameters, for any given task.

As a result, there is growing interest in combining aspects of neural networks and inference on stochastic processes as a potential solution to some of the downsides of both (Huang et al., 2015; Wilson et al., 2016). In this work we introduce a neural network-based formulation that learns an approximation of a stochastic process, which we term *Neural Processes* (NPs). NPs display some of the fundamental properties of GPs, namely they learn to model distributions over functions, are able to estimate the uncertainty over their predictions conditioned on context observations, and shift some of the workload from training to test time, which allows for model flexibility. Crucially, NPs generate predictions in a computationally efficient way. Given n context points and m target points, inference with a trained NP corresponds to a forward pass in a deep NN, which scales with $\mathcal{O}(n+m)$ as opposed to the $\mathcal{O}((n+m)^3)$ runtime of classic GPs. Furthermore the model overcomes many functional design restrictions by learning an implicit kernel from the data directly.

[1]DeepMind, London, UK. Correspondence to: Marta Garnelo <garnelo@google.com>.

Presented at the ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models

Our main contributions are:

1. We introduce Neural Processes, a class of models that combine benefits of neural networks and stochastic processes.
2. We compare NPs to related work in meta-learning, deep latent variable models and Gaussian processes. Given that NPs are linked to many of these areas, they form a bridge for comparison between many related topics.
3. We showcase the benefits and abilities of NPs by applying them to a range of tasks including 1-D regression, real-world image completion, Bayesian optimization and contextual bandits.

2. Model

2.1. Neural processes as stochastic processes

The standard approach to defining a stochastic process is via its finite-dimensional marginal distributions. Specifically, we consider the process as a random function $F : \mathcal{X} \to \mathcal{Y}$ and for each finite sequence $x_{1:n} = (x_1, \dots, x_n)$ with $x_i \in \mathcal{X}$, we define the marginal joint distribution over the function values $Y_{1:n} := (F(x_1), \dots, F(x_n))$. For example, in the case of GPs, these joint distributions are multivariate Gaussians parameterised by a mean and a covariance function.

Given a collection of joint distributions $\rho_{x_{1:n}}$ we can derive two necessary conditions to be able to define a stochastic process F such that $\rho_{x_{1:n}}$ is the marginal distribution of $(F(x_1), \dots, F(x_n))$, for each finite sequence $x_{1:n}$. These conditions are: (finite) exchangeability and consistency. As stated by the Kolmogorov Extension Theorem (Øksendal, 2003) these conditions are sufficient to define a stochastic process.

Exchangeability This condition requires the joint distributions to be invariant to permutations of the elements in $x_{1:n}$. More precisely, for each finite n, if π is a permutation of $\{1, \dots, n\}$, then:

$$\begin{aligned}\rho_{x_{1:n}}(y_{1:n}) &:= \rho_{x_1,\dots,x_n}(y_1, \dots, y_n) \\ &= \rho_{x_{\pi(1)},\dots,x_{\pi(n)}}(y_{\pi(1)}, \dots, y_{\pi(n)}) =: \rho_{\pi(x_{1:n})}(\pi(y_{1:n}))\end{aligned} \quad (1)$$

where $\pi(x_{1:n}) := (x_{\pi(1)}, \dots, x_{\pi(n)})$ and $\pi(y_{1:n}) := (y_{\pi(1)}, \dots, y_{\pi(n)})$.

Consistency If we marginalise out a part of the sequence the resulting marginal distribution is the same as that defined on the original sequence. More precisely, if $1 \le m \le n$, then:

$$\rho_{x_{1:m}}(y_{1:m}) = \int \rho_{x_{1:n}}(y_{1:n}) dy_{m+1:n}. \quad (2)$$

Take, for example, three different sequences $x_{1:n}, \pi(x_{1:n})$ and $x_{1:m}$ as well as their corresponding joint distributions $\rho_{x_{1:n}}, \rho_{\pi(x_{1:n})}$ and $\rho_{x_{1:m}}$. In order for these joint distributions to all be marginals of some higher-dimensional distribution given by the stochastic process F, they have to satisfy equations 1 and 2 above.

Given a particular instantiation of the stochastic process f the joint distribution is defined as:

$$\rho_{x_{1:n}}(y_{1:n}) = \int p(f) p(y_{1:n}|f, x_{1:n}) df. \quad (3)$$

Here p denotes the abstract probability distribution over all random quantities. Instead of $Y_i = F(x_i)$, we add some observation noise $Y_i \sim \mathcal{N}(F(x_i), \sigma^2)$ and define p as:

$$p(y_{1:n}|f, x_{1:n}) = \prod_{i=1}^{n} \mathcal{N}(y_i | f(x_i), \sigma^2). \quad (4)$$

Inserting this into equation 3 the stochastic process is specified by:

$$\rho_{x_{1:n}}(y_{1:n}) = \int p(f) \prod_{i=1}^{n} \mathcal{N}(y_i | f(x_i), \sigma^2) df. \quad (5)$$

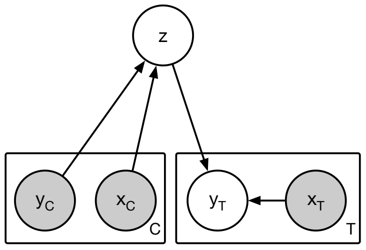
In other words, exchangeability and consistency of the collection of joint distributions $\{\rho_{x_{1:n}}\}$ impliy the existence of a stochastic process F such that the observations $Y_{1:n}$ become iid conditional upon F. This essentially corresponds to a conditional version of de Finetti's Theorem that anchors much of Bayesian nonparametrics (De Finetti, 1937). In order to represent a stochastic process using a NP, we will approximate it with a neural network, and assume that F can be parameterised by a high-dimensional random vector z, and write $F(x) = g(x, z)$ for some fixed and learnable function g (i.e. the randomness in F is due to that of z). The generative model (Figure 1a) then follows from (5):

$$p(z, y_{1:n}|x_{1:n}) = p(z) \prod_{i=1}^{n} \mathcal{N}(y_i | g(x_i, z), \sigma^2) \quad (6)$$

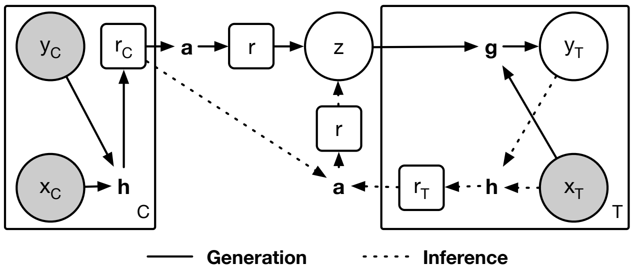
where, following ideas of variational auto-encoders, we assume $p(z)$ is a multivariate standard normal, and $g(x_i, z)$ is a neural network which captures the complexities of the model.

To learn such a distribution over random functions, rather than a single function, it is essential to train the system using multiple datasets concurrently, with each dataset being a sequence of inputs $x_{1:n}$ and outputs $y_{1:n}$, so that we can learn the variability of the random function from the variability of the datasets (see section 2.2).

Since the decoder g is non-linear, we can use amortised variational inference to learn it. Let $q(z|x_{1:n}, y_{1:n})$ be a



(a) Graphical model



(b) Computational diagram

Figure 1. **Neural process model. (a)** Graphical model of a neural process. x and y correspond to the data where $y = f(x)$. C and T are the number of context points and target points respectively and z is the global latent variable. A grey background indicates that the variable is observed. **(b)** Diagram of our neural process implementation. Variables in circles correspond to the variables of the graphical model in (a), variables in square boxes to the intermediate representations of NPs and unbound, bold letters to the following computation modules: h - encoder, a - aggregator and g - decoder. In our implementation h and g correspond to neural networks and a to the mean function. The continuous lines depict the generative process, the dotted lines the inference.

variational posterior of the latent variables z, parameterised by another neural network that is invariant to permutations of the sequences $x_{1:n}, y_{1:n}$. Then the evidence lower-bound (ELBO) is given by:

$$\log p(y_{1:n}|x_{1:n}) \qquad (7)$$
$$\geq \mathbb{E}_{q(z|x_{1:n},y_{1:n})}\left[\sum_{i=1}^{n} \log p(y_i|z,x_i) + \log \frac{p(z)}{q(z|x_{1:n},y_{1:n})}\right]$$

In an alternative objective that better reflects the desired model behaviour at test time, we split the dataset into a context set, $x_{1:m}, y_{1:m}$ and a target set $x_{m+1:n}, y_{m+1:n}$, and model the conditional of the target given the context. This gives:

$$\log p(y_{m+1:n}|x_{1:n}, y_{1:m})$$
$$\geq \mathbb{E}_{q(z|x_{1:n},y_{1:n})}\left[\sum_{i=m+1}^{n} \log p(y_i|z,x_i) + \log \frac{p(z|x_{1:m},y_{1:m})}{q(z|x_{1:n},y_{1:n})}\right] \qquad (8)$$

Note that in the above the conditional prior $p(z|x_{1:m}, y_{1:m})$ is intractable. We can approximate it using the variational posterior $q(z|x_{1:m}, y_{1:m})$, which gives,

$$\log p(y_{m+1:n}|x_{1:n}, y_{1:m}) \qquad (9)$$
$$\geq \mathbb{E}_{q(z|x_{1:n},y_{1:n})}\left[\sum_{i=m+1}^{n} \log p(y_i|z,x_i) + \log \frac{q(z|x_{1:m},y_{1:m})}{q(z|x_{1:n},y_{1:n})}\right]$$

2.2. Distributions over functions

A key motivation for NPs is the ability to represent a distribution over functions rather than a single function. In order to train such a model we need a training procedure that reflects this task.

More formally, to train a NP we form a dataset that consists of functions $f : X \to Y$ that are sampled from some underlying distribution $\mathcal{D}$. As an illustrating example consider a dataset consisting of functions $f_d(x) \sim \mathcal{GP}$ that have been generated using a Gaussian process with a fixed kernel. For each of the functions $f_d(x)$ our dataset contains a number of $(x, y)_i$ tuples where $y_i = f_d(x_i)$. For training purposes we divide these points into a set of n context points $C = \{(x, y)_i\}_{i=1}^{n}$ and a set of $n + m$ target points which consists of all points in C as well as m additional unobserved points $T = \{(x, y)_i\}_{i=1}^{n+m}$. During testing the model is presented with some context C and has to predict the target values $y_T = f(x_T)$ at target positions x_T.

In order to be able to predict accurately across the entire dataset a model needs to learn a distribution that covers all of the functions observed in training and be able to take into account the context data at test time.

2.3. Global latent variable

As mentioned above, neural processes include a latent variable z that captures F. This latent variable is of particular interest because it captures the global uncertainty, which allows us to sample at a global level – one function f_d at a time, rather than at a local output level – one y_i value for each x_i at a time (independently of the remaining y_T).

In addition, since we are passing all of the context's information through this single variable we can formulate the model in a Bayesian framework. In the absence of context points C the latent distribution $p(z)$ would correspond to a data specific prior the model has learned during training. As we add observations the latent distribution encoded by the model amounts to the posterior $p(z|C)$ over the function given the context. On top of this, as shown in equation 9, instead of using a zero-information prior $p(z)$, we condition

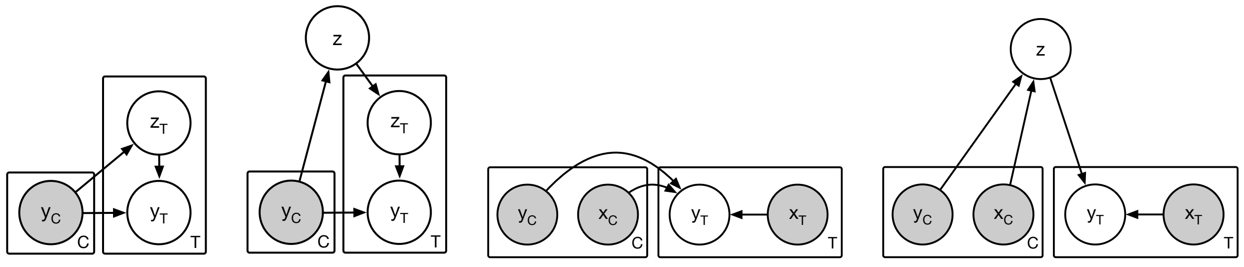


Figure 2. Graphical models of related models (a-c) and of the neural process (d). Gray shading indicates the variable is observed. C stands for context variables and T for target variables i.e. the variables to predict given C.

the prior on the context. As such this prior is equivalent to a less informed posterior of the underlying function. This formulation makes it clear that the posterior given a subset of the context points will serve as the prior when additional context points are included. By using this setup, and training with different sizes of context, we encourage the learned model to be flexible with regards to the number and position of the context points.

2.4. The Neural process model

In our implementation of NPs we accommodate for two additional desiderata: invariance to the order of context points and computational efficiency. The resulting model can be boiled down to three core components (see Figure 1b):

- An **encoder** h from input space into representation space that takes in *pairs* of $(x, y)_i$ context values and produces a representation $r_i = h((x, y)_i)$ for each of the pairs. We parameterise h as a neural network.
- An **aggregator** a that summarises the encoded inputs. We are interested in obtaining a single order-invariant global representation r that parameterises the latent distribution $z \sim \mathcal{N}(\mu(r), I\sigma(r))$. The simplest operation that ensures order-invariance and works well in practice is the mean function $r = a(r_i) = \frac{1}{n}\sum_{i=1}^{n} r_i$. Crucially, the aggregator reduces the runtime to $\mathcal{O}(n + m)$ where n and m are the number of context and target points respectively.
- A **conditional decoder** g that takes as input the sampled global latent variable z as well as the new target locations x_T and outputs the predictions $\hat{y}_T$ for the corresponding values of $f(x_T) = y_T$.

3. Related work

3.1. Conditional neural processes

Neural Processes (NPs) are a generalisation of *Conditional Neural Processes* (CNPs, Garnelo et al. (2018)). CNPs share a large part of the motivation behind neural processes, but lack a latent variable that allows for global sampling (see Figure 2c for a diagram of the model). As a result, CNPs are unable to produce different function samples for the same context data, which can be important if modelling this uncertainty is desirable. It is worth mentioning that the original CNP formulation did include experiments with a latent variable in addition to the deterministic connection. However, given the deterministic connections to the predicted variables, the role of the global latent variable is not clear. In contrast, NPs constitute a more clear-cut generalisation of the original deterministic CNP with stronger parallels to other latent variable models and approximate Bayesian methods. These parallels allow us to compare our model to a wide range of related research areas in the following sections.

Finally, NPs and CNPs themselves can be seen as generalizations of recently published generative query networks (GQN) which apply a similar training procedure to predict new viewpoints in 3D scenes given some context observations (Eslami et al., 2018). Consistent GQN (CGQN) is an extension of GQN that focuses on generating consistent samples and is thus also closely related to NPs (Kumar et al., 2018).

3.2. Gaussian processes

We start by considering models that, like NPs, lie on the spectrum between neural networks (NNs) and Gaussian processes (GPs). Algorithms on the NN end of the spectrum fit a single function that they learn from a very large amount of data directly. GPs on the other hand can represent a distribution over a family of functions, which is constrained by an assumption on the functional form of the covariance between two points.

Scattered across this spectrum, we can place recent research that has combined ideas from Bayesian non-parametrics with neural networks. Methods like (Calandra et al., 2016; Huang et al., 2015) remain fairly close to the GPs, but incorporate NNs to pre-process the input data. Deep GPs have some conceptual similarity to NNs as they stack GPs to obtain *deep* models (Damianou & Lawrence, 2013). Approaches that are more similar to NNs include for example neural networks whose weights are sampled using a GPs (Wilson et al., 2011) or networks where each unit represents a different kernel (Sun et al., 2018).

There are two models on this spectrum that are closely related to NPs: matching networks (MN, Vinyals et al. (2016)) and deep kernel learning (DKL, Wilson et al. (2016)). As with NPs both use NNs to extract representations from the data, but while NPs learn the 'kernel' to compare data points implicitly these other two models pass the representation to an explicit distance kernel. MNs use this kernel to measure the similarity between contexts and targets for few shot classification while the kernel in DKL is used to parametrise a GP. Because of this explicit kernel the computational complexity of MNs and DKL would be quadratic and cubic instead of $\mathcal{O}(n+m)$ like it is for NPs. To overcome this computational complexity DKL replace a standard GP with a kernel approximation given by a KISS GP (Wilson & Nickisch, 2015), while prototypical networks (Snell et al., 2017) are introduced as a more light-weight version of MNs that also scale with $\mathcal{O}(n+m)$.

Finally concurrent work by Ma et al introduces variational implicit processes, which share large part of the motivation of NPs but are implemented as GPs (Ma et al., 2018). In this context NPs can be interpreted as a neural implementation of an implicit stochastic process.

On this spectrum from NNs to GPs, neural processes remain closer to the neural end than most of the models mentioned above. By giving up on the explicit definition of a kernel NPs lose some of the mathematical guarantees of GPs, but trade this off for data-driven 'priors' and computational efficiency.

3.3. Meta-learning

In contemporary meta-learning vocabulary, NPs and GPs can be seen to be methods for 'few-shot function estimation'. In this section we compare with related models that can be used to the same end. A prominent example is matching networks (Vinyals et al., 2016), but there is a large literature of similar models for classification (Koch et al., 2015; Santoro et al., 2016), reinforcement learning (Wang et al., 2016), parameter update (Finn et al., 2017; 2018), natural language processing (Bowman et al., 2015) and program induction (Devlin et al., 2017). Related are generative meta-learning approaches that carry out few-shot estimation of the data densities (van den Oord et al., 2016; Reed et al., 2017; Bornschein et al., 2017; Rezende et al., 2016).

Meta-learning models share the fundamental motivations of NPs as they shift workload from training time to test time. NPs can therefore be described as meta-learning algorithms for few-shot function regression, although as shown in Garnelo et al. (2018) they can also be applied to few-shot learning tasks beyond regression.

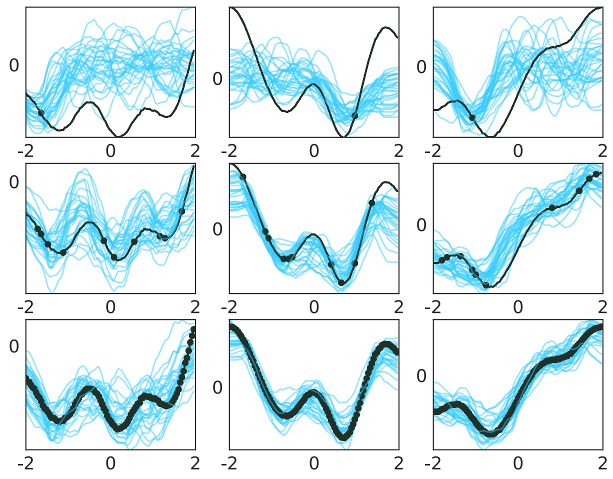


Figure 3. **1-D function regression.** The plots show samples of curves conditioned on an increasing number of context points (1, 10 and 100 in each row respectively). The true underlying curve is shown in black and the context points as black circles. Each column corresponds to a different example. With only one observation the variance away from that context point is high between the sampled curves. As the number of context points increases the sampled curves increasingly resemble the ground truth curve and the overall variance is reduced.

3.4. Bayesian methods

The link between meta-learning methods and other research areas, like GPs and Bayesian methods, is not always evident. Interestingly, recent work by Grant et al. (2018) out the relation between model agnostic meta learning (MAML, Finn et al. (2017)) and hierarchical Bayesian inference. In this work the meta-learning properties of MAML are formulated as a result of task-specific variables that are conditionally independent given a higher level variable. This hierarchical description can be rewritten as a probabilistic inference problem and the resulting marginal likelihood $p(y|C)$ matches the original MAML objective. The parallels between NPs and hierarchical Bayesian methods are similarly straightforward. Given the graphical model in Figure 2d we can write out the conditional marginal likelihood as a hierarchical inference problem:

$$\log p(y_t|C, x_t) = \log \int p(y_t|z, x_t)p(z|C)dz \qquad (10)$$

Another interesting area that connects Bayesian methods and NNs are Bayesian neural networks (Gal & Ghahramani, 2016; Blundell et al., 2015; Louizos et al., 2017; Louizos &

Welling, 2017). These models learn distributions over the network weights and use the posterior of these weights to estimate the values of y_T given y_C. In this context NPs can be thought of as amortised version of Bayesian DL.

3.5. Conditional latent variable models

We have covered algorithms that are conceptually similar to NPs and algorithms that carry out similar tasks to NPs. In this section we look at models of the same family as NPs: conditional latent variable models. Such models (Figure 2a) learn the conditional distribution $p(y_T|y_C, z)$ where z is a latent variable that can be sampled to generate different predictions. Training this type of directed graphical model is intractable and as with variational autoencoders (VAEs, Rezende et al. (2014); Kingma & Welling (2013)), conditional variational autoencoders (CVAEs, Sohn et al. (2015)) approximate the objective function using the variational lower bound on the log likelihood:

$$\log p(y_t|y_c) \geq \mathbb{E}_{q(z_T|y_c,y_t)}\Bigg[\log p(y_t|z_T, y_c) + \log \frac{p(z_T|y_c)}{q(z_T|y_c, y_t)}\Bigg] \tag{11}$$

We refer to the latent variable of CVAEs z_T as a local latent variable in order to distinguish from global latent variables that are present in the models later on. We call this latent variable local as it is sampled anew for each of the output predictions $y_{T,i}$. This is in contrast to a global latent variable that is only sampled once and used to predict multiple values of y_t. In the CVAE, conditioning on the context is done by adding the dependence both in the prior $p(z_T|y_c)$ and decoder $p(y|z_T, y_c)$ so they can be considered as deterministic functions of the context.

CVAEs have been extended in a number of ways for example by adding attention (Rezende et al., 2016). Another related extension is generative matching networks (GMNs, Bartunov & Vetrov (2016)), where the conditioning input is pre-processed in a way that is similar to the matching networks model.

A more complex version of the CVAE that is very relevant in this context is the neural statistician (NS, Edwards & Storkey (2016)). Similar to the neural process, the neural statistician contains a global latent variable z that captures global uncertainty (see Figure 2b). A crucial difference is that while NPs represent the distribution over functions, NS represents the distribution over sets. Since NS does not contain a corresponding x value for each y value, it does not capture a pair-wise relation like GPs and NPs, but rather a general distribution of the y values. Rather than generating different y values by querying the model with different x values, NS generates different y values by sampling an additional local hidden variable z_T.

The ELBO of the NS reflects the hierarchical nature of the model with a double expectation over the local and the global variable. If we leave out the local latent variable for a more direct comparison to NPs the ELBO becomes:

$$\log p(y_t, y_c) \geq \mathbb{E}_{q(z|y_c,y_t)}\Bigg[\log p(y_t|z) + \log \frac{p(z)}{q(z|y_c, y_t)}\Bigg] \tag{12}$$

Notably the prior $p(z)$ of NS is not conditioned on the context. The prior of NPs on the other hand is conditional (equation 9), which brings the training objective closer to the way the model is used at test time. Another variant of the ELBO is presented in the variational homoencoder (Hewitt et al., 2018), a model that is very similar to the neural statistician but uses a separate subset of data points for the context and predictions.

As reflected in Figure 2 the main difference between NPs and the conditional latent variable models is the lack of an x variable that allows for targeted sampling of the latent distribution. This change, despite seeming small, drastically changes the range of applications. Targeted sampling, for example allows for generation and completion tasks (e.g. the image completion tasks) or the addition of some downstream task (like using an NP for reinforcement learning). It is worth mentioning that all of the conditional latent variable models have also been applied to few-shot classification problems, where the data space generally consists of input tuples (x, y), rather than just single outputs y. The models are able to carry this out by framing classification either as a comparison between the log likelihoods of the different classes or by looking at the KL between the different posteriors, thereby overcoming the need of working with data tuples.

4. Results

4.1. 1-D function regression

In order to test whether neural processes indeed learn to model distributions over functions we first apply them to a 1-D function regression task. The functions for this experiment are generated using a GP with varying kernel parameters for each function. At every training step we sample a set of values for the Gaussian kernel of a GP and use those to sample a function $f_D(x)$. A random number of the $(x, y)_C$ pairs are passed into the decoder of the NP as context points. We pick additional unobserved pairs $(x, y)_U$ which we combine with the observed context points $(x, y)_C$ as targets and feed x_T to the decoder that returns its estimate $\hat{y_T}$ of the underlying value of y_T.

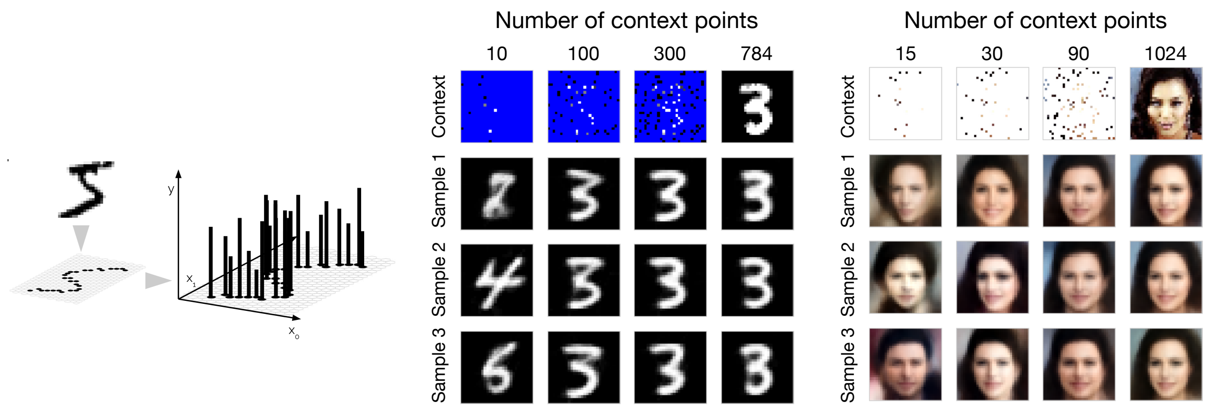


Figure 4. **Pixel-wise regression on MNIST and CelebA** The diagram on the left visualises how pixel-wise image completion can be framed as a 2-D regression task where f(pixel coordinates) = pixel brightness. The figures to the right of the diagram show the results on image completion for MNIST and CelebA. The images on the top correspond to the context points provided to the model. For better clarity the unobserved pixels have been coloured blue for the MNIST images and white for CelebA. Each of the rows corresponds to a different sample given the context points. As the number of context points increases the predicted pixels get closer to the underlying ones and the variance across samples decreases.

Some sample curves are shown in Figure 3. For the same underlying ground truth curve (black line) we run the neural process using varying numbers of context points and generate several samples for each run (light-blue lines). As evidenced by the results the model has learned some key properties of the 1-D curves from the data such as continuity and the general shape of functions sampled from a GP with a Gaussian kernel. When provided with only one context point the model generates curves that fluctuate around 0, the prior of the data-generating GP. Crucially, these curves go through or near the observed context point and display a higher variance in regions where no observations are present. As the number of context points increases this uncertainty is reduced and the model's predictions better match the underlying ground truth. Given that this is a neural approximation the curves will sometimes only approach the observations points as opposed to go through them as it is the case for GPs. On the other hand once the model is trained it can regress more than just one data set i.e. it will produce sensible results for curves generated using any kernel parameters observed during training.

4.2. 2-D function regression

One of the benefits of neural processes is their functional flexibility as they can learn non-trivial 'kernels' from the data directly. In order to test this we apply NPs to a more complex regression problem. We carry out image completion as a regression task, where we provide some of the pixels as context and do pixel-wise prediction over the entire image. In this formulation the x_i values would correspond to the Cartesian coordinates of each pixel and the y_i values to the pixel intensity (see Figure 4 for an explanation of this). It is important to point out that we choose images as our dataset because they constitute a complex 2-D function and they are easy to evaluate visually. It is important to point out that NPs, as such, have not been designed for image generation like other specialised generative models.

We train separate models on the MNIST (LeCun et al., 1998) and the CelebA (Liu et al., 2015) datasets. As shown in Figure 4 the model performs well on both tasks. In the case of the MNIST digits the uncertainty is reflected in the variability of the generated digit. Given only a few context points more than just one digit can fit the observations and as a result the model produces different digits when sampled several times. As the number of context points increases the set of possible digits is reduced and the model produces the same digit, albeit with structural modifications that become smaller as the number of context points increases.

The same holds for the CelebA dataset. In this case, when provided limited context the model samples from a wider range of possible faces and as it observes more context points it converges towards very similar looking faces. We do not expect the model to reconstruct the target image perfectly even when all the pixels are provided as context, since the latent variable z constitutes a strong bottleneck. This can be seen in the final column of the figure where the predicted images are not only not identical to the ground truth but also vary between themselves. The latter is likely a cause of the latent variance which has been clipped to a small value to avoid collapsing, so even when no uncertainty is present we can generate different samples from $p(z|C)$.

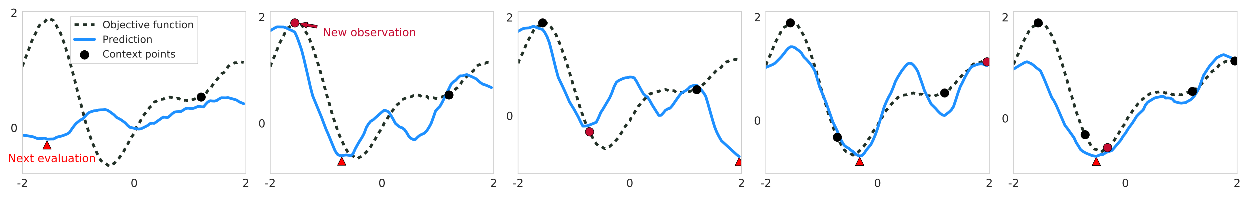


Figure 5. **Thompson sampling with neural processes on a 1-D objective function.** The plots show the optimisation process over five iterations. Each prediction function (blue) is drawn by sampling a latent variable conditioned on an incresing number of context points (black circles). The underlying ground truth function is depicted as a black dotted line. The red triangle indicates the next evaluation point which corresponds to the minimum value of the sampled NP curve. The red circle in the following iteration corresponds to this evaluation point with its underlying ground truth value that serves as a new context point to the NP.

4.3. Black-box optimisation with Thompson sampling

To showcase the utility of sampling entire consistent trajectories we apply neural processes to Bayesian optimisation on 1-D function using Thompson sampling (Thompson, 1933). Thompson sampling (also known as randomised probability matching) is an approach to tackle the exploration-exploitation dilemma by maintaining a posterior distribution over model parameters. A decision is taken by drawing a sample of model parameters and acting greedily under the resulting policy. The posterior distribution is then updated and the process is repeated. Despite its simplicity, Thompson sampling has been shown to be highly effective both empirically and in theory. It is commonly applied to black box optimisation and multi-armed bandit problems (e.g. Agrawal & Goyal, 2012; Shahriari et al., 2016).

Neural processes lend themselves naturally to Thompson sampling by instead drawing a function over the space of interest, finding its minimum and adding the observed outcome to the context set for the next iteration. As shown in in Figure 3, function draws show high variance when only few observations are available, modelling uncertainty in a similar way to draws from a posterior over parameters given a small data set. An example of this procedure for neural processes on a 1-D objective function is shown in Figure 5.

We report the average number of steps required by an NP to reach the global minimum of a function generated from a GP prior in Table 1. For an easier comparison the values are normalised by the amount of steps required when doing optimisation using random search. On average, NPs take four times fewer iterations than random search on this task. An upper bound on performance is given by a Gaussian process with the same kernel than the GP that generated the function to be optimised. NPs do not reach this optimal performance, as their samples are more noisy than those of a GP, but are faster to evaluate since merely a forward pass through the network is needed. This difference in computational speed is bound to get more notable as the dimensionality of the problem and the number of necessary function evaluations increases.

Neural process	Gaussian process	Random Search
0.26	0.14	1.00

Table 1. **Bayesian Optimisation using Thompson sampling.** Average number of optimisation steps needed to reach the global minimum of a 1-D function generated by a Gaussian process. The values are normalised by the number of steps taken using random search. The performance of the Gaussian process with the correct kernel constitutes an upper bound on performance.

4.4. Contextual bandits

Finally, we apply neural processes to the wheel bandit problem introduced in Riquelme et al. (2018), which constitutes a contextual bandit task on the unit circle with varying needs for exploration that can be smoothly parameterised. The problem can be summarised as follows (see Figure 6 for clarity): a unit circle is divided into a low-reward region (blue area) and four high-reward regions (the other four coloured areas). The size of the low-reward region is defined by a scalar δ. At every episode a different value for δ is selected. The agent is then provided with some coordinates $X = (X_1, X_2)$ within the circle and has to choose among $k = 5$ arms depending on the area the coordinates fall into. If $||X|| \leq \delta$, the sample falls within the low-reward region (blue). In this case $k = 1$ is the optimal action, as it provides a reward drawn from $r \sim N(1.2, 0.01^2)$, while all other actions only return $r \sim \mathcal{N}(1.0, 0.01^2)$. If the sample falls within any of the four high-reward region ($||X|| > \delta$), the optimal arm will be any of the remaining four $k = 2-5$, depending on the specific area. Pulling the optimal arm here results in a high reward $r \sim \mathcal{N}(50.0, 0.01^2)$, and as before all other arms receive $\mathcal{N}(1.0, 0.01^2)$ *except* for arm $k = 1$ which again returns $\mathcal{N}(1.2, 0.01^2)$.

We compare our model to a large range of methods that can be used for Thompson sampling, taking results from

Table 2. Results on the wheel bandit problem for increasing values of δ. Shown are mean and standard errors for both cumulative and simple regret (a measure of the quality of the final policy) over 100 trials. Results normalised wrt. the performance of a uniform agent.

δ	**0.5**	**0.7**	**0.9**	**0.95**	**0.99**
Cumulative regret					
Uniform	100.00 $\pm$ 0.08	100.00 $\pm$ 0.09	100.00 $\pm$ 0.25	100.00 $\pm$ 0.37	100.00 $\pm$ 0.78
LinGreedy ($\epsilon = 0.0$)	65.89 $\pm$ 4.90	71.71 $\pm$ 4.31	108.86 $\pm$ 3.10	102.80 $\pm$ 3.06	104.80 $\pm$ 0.91
Dropout	7.89 $\pm$ 1.51	9.03 $\pm$ 2.58	36.58 $\pm$ 3.62	63.12 $\pm$ 4.26	98.68 $\pm$ 1.59
LinGreedy ($\epsilon = 0.05$)	7.86 $\pm$ 0.27	9.58 $\pm$ 0.35	19.42 $\pm$ 0.78	33.06 $\pm$ 2.06	74.17 $\pm$ 1.63
Bayes by Backprob (Blundell et al., 2015)	1.37 $\pm$ 0.07	3.32 $\pm$ 0.80	34.42 $\pm$ 5.50	59.04 $\pm$ 5.59	97.38 $\pm$ 2.66
NeuralLinear	**0.95** $\pm$ 0.02	**1.60** $\pm$ 0.03	4.65 $\pm$ 0.18	9.56 $\pm$ 0.36	49.63 $\pm$ 2.41
MAML (Finn et al., 2017)	2.95 $\pm$ 0.12	3.11 $\pm$ 0.16	4.84 $\pm$ 0.22	7.01 $\pm$ 0.33	22.93 $\pm$ 1.57
Neural Processes	1.60 $\pm$ 0.06	1.75 $\pm$ 0.05	**3.31** $\pm$ 0.10	**5.71** $\pm$ 0.24	**22.13** $\pm$ 1.23
Simple regret					
Uniform	100.00 $\pm$ 0.45	100.00 $\pm$ 0.78	100.00 $\pm$ 1.18	100.00 $\pm$ 2.21	100.00 $\pm$ 4.21
LinGreedy ($\epsilon = 0.0$)	66.59 $\pm$ 5.02	73.06 $\pm$ 4.55	108.56 $\pm$ 3.65	105.01 $\pm$ 3.59	105.19 $\pm$ 4.14
Dropout	6.57 $\pm$ 1.48	6.37 $\pm$ 2.53	35.02 $\pm$ 3.94	59.45 $\pm$ 4.74	102.12 $\pm$ 4.76
LinGreedy ($\epsilon = 0.05$)	5.53 $\pm$ 0.19	6.07 $\pm$ 0.24	8.49 $\pm$ 0.47	12.65 $\pm$ 1.12	57.62 $\pm$ 3.57
Bayes by Backprob (Blundell et al., 2015)	0.60 $\pm$ 0.09	1.45 $\pm$ 0.61	27.03 $\pm$ 6.19	56.64 $\pm$ 6.36	102.96 $\pm$ 5.93
NeuralLinear	**0.33** $\pm$ 0.04	**0.79** $\pm$ 0.07	**2.17** $\pm$ 0.14	**4.08** $\pm$ 0.20	35.89 $\pm$ 2.98
MAML (Finn et al., 2017)	2.49 $\pm$ 0.12	3.00 $\pm$ 0.35	4.75 $\pm$ 0.48	7.10 $\pm$ 0.77	22.89 $\pm$ 1.41
Neural Processes	1.04 $\pm$ 0.06	1.26 $\pm$ 0.21	2.90 $\pm$ 0.35	5.45 $\pm$ 0.47	**21.45** $\pm$ 1.3

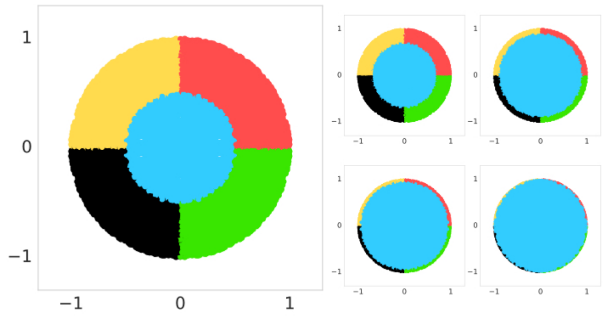


Figure 6. The wheel bandit problem with varying values of δ.

Riquelme et al. (2018), who kindly agreed to share the experiment and evaluation code with us. Neural Processes can be applied to this problem by training on a distribution of tasks before applying the method. Since the methods described in Riquelme et al. (2018) do not require such a pre-training phase, we also include Model-agnostic meta-learning (MAML, Finn et al. (2017)), a method relying on a similar pre-training phase, using code made available by the authors. For both NPs and MAML methods, we create a batch for pre-training by first sampling M different wheel problems $\{\delta_i\}_{i=1}^{M}, \delta_i \sim \mathcal{U}(0, 1)$, followed by sampling tuples $\{(X, a, r)_j\}_{j=1}^{N}$ for context X, arm a and associated reward r for each δ_i. We set $M = 64, N = 562$, using 512 context and 50 target points for Neural Processes, and an equal amount of data points for the meta- and inner-updates in MAML. Note that since gradient steps are necessary for MAML to adapt to data from each test problem, we reset the parameters after each evaluation run. This additional step is not necessary for neural processes.

Table 2 shows the quantitative evaluation on this task. We observe that Neural Processes are a highly competitive method, performing similar to MAML and the *NeuralLinear* baseline in Riquelme et al. (2018), which is consistently among the best out of 20 algorithms compared.

5. Discussion

We introduce Neural processes, a family of models that combines the benefits of stochastic processes and neural networks. NPs learn to represent distributions over functions and make flexible predictions at test time conditioned on some context input. Instead of requiring a handcrafted kernel, NPs learn an implicit measure from the data directly.

We apply NPs to a range of regression tasks to showcase their flexibility. The goal of this paper is to introduce NPs and compare them to the currenly ongoing research. As such, the tasks presented here are diverse but relatively low-dimensional. We leave it to future work to scale NPs up to higher dimensional problems that are likely to highlight the benefit of lower computational complexity and data driven representations.

Acknowledgements

We would like to thank Tiago Ramalho, Oriol Vinyals, Adam Kosiorek, Irene Garnelo, Daniel Burgess, Kevin McKee and Claire McCoy for insightful discussions and being awesome people. We would also like to thank Carlos Riquelme, George Tucker and Jasper Snoek for providing the code to reproduce the results of their contextual bandits experiments (and, of course, also being awesome people).

References

Agrawal, S. and Goyal, N. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pp. 39–1, 2012.

Bartunov, S. and Vetrov, D. P. Fast adaptation in generative models with generative matching networks. *arXiv preprint arXiv:1612.02192*, 2016.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Bornschein, J., Mnih, A., Zoran, D., and J. Rezende, D. Variational memory addressing in generative models. In *Advances in Neural Information Processing Systems*, pp. 3923–3932, 2017.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold gaussian processes for regression. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pp. 3338–3345. IEEE, 2016.

Damianou, A. and Lawrence, N. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pp. 207–215, 2013.

De Finetti, B. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pp. 1–68, 1937.

Devlin, J., Bunel, R. R., Singh, R., Hausknecht, M., and Kohli, P. Neural program meta-induction. In *Advances in Neural Information Processing Systems*, pp. 2077–2085, 2017.

Edwards, H. and Storkey, A. Towards a neural statistician. *arXiv preprint arXiv:1606.02185*, 2016.

Eslami, S. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

Finn, C., Xu, K., and Levine, S. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

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

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

Hewitt, L., Gane, A., Jaakkola, T., and Tenenbaum, J. B. The variational homoencoder: Learning to infer high-capacity generative models from few examples. 2018.

Huang, W.-b., Zhao, D., Sun, F., Liu, H., and Chang, E. Y. Scalable gaussian process regression using deep neural networks. In *IJCAI*, pp. 3576–3582, 2015.

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

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

Kumar, A., Eslami, S. M. A., Rezende, D. J., Garnelo, M., Viola, F., Lockhart, E., and Shanahan, M. Consistent generative query networks. In *CoRR*, 2018.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3290–3300, 2017.

Ma, C., Li, Y., and Hernández-Lobato, J. M. Variational implicit processes. *arXiv preprint arXiv:1806.02390*, 2018.

Øksendal, B. Stochastic differential equations. In *Stochastic differential equations*, pp. 11. Springer, 2003.

Quiñonero-Candela, J. and Rasmussen, C. E. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

Reed, S., Chen, Y., Paine, T., Oord, A. v. d., Eslami, S., J. Rezende, D., Vinyals, O., and de Freitas, N. Few-shot autoregressive density estimation: Towards learning to learn distributions. 2017.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Rezende, D. J., Mohamed, S., Danihelka, I., Gregor, K., and Wierstra, D. One-shot generalization in deep generative models. *arXiv preprint arXiv:1603.05106*, 2016.

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.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

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, 2016.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4080–4090, 2017.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pp. 3483–3491, 2015.

Sun, S., Zhang, G., Wang, C., Zeng, W., Li, J., and Grosse, R. Differentiable compositional kernel learning for gaussian processes. *arXiv preprint arXiv:1806.04326*, 2018.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pp. 3630–3638, 2016.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pp. 1775–1784, 2015.

Wilson, A. G., Knowles, D. A., and Ghahramani, Z. Gaussian process regression networks. *arXiv preprint arXiv:1110.4411*, 2011.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial Intelligence and Statistics*, pp. 370–378, 2016.