Conditional Bayesian Neural Networks for Few-Shot Learning

Miguel Jaques

Master of Science by Research
Data Science
University of Edinburgh
2018

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Miguel Jaques*)

Abstract

The recent advances in meta-learning-based few-shot learning have sparked interest in designing systems that can not only learn how to learn, but also produce uncertainty estimates conditioned on the task at hand. In this work we propose a K-shot N-way extenstion of the Learnet model, and generalise it to hierarchical Bayesian model, that we call Bayes-Learnet, which can produce a variational distribution over model parameters conditioned on the task. The model is trained by simple maximization of an evidence lower-bound using stochastic backpropagation. We show that Bayes-Learnet performs comparably to the state-of-the-art in K-shot N-way classification problems, and evaluate its ability to model uncertainty by applying it to regression, outlier detection and active learning problems. We also show, for the first time, that a single model trained for 1-shot learning generalises correctly at test time to any number of shots and ways, and propose a novel order-invariant data aggregation method.

Acknowledgements

I thank my supervisor, Tim, for his guidance and support. I thank Nick Pawlowski, Chenyang Zhao, Steven Kleinegesse, Christos Maniatis, Charlie Nash, and other colleagues for useful discussions. I thank Fidelma and Filip for keeping it positive around the flat and proof-reading this document. I thank my family for their support, and I thank Johanna, who is my light.

Contents

List of Figures

Chapter 1

Introduction

1.1 From learning to meta-learning

Neural networks have been at the center of most advances in machine learning in the past half-decade, particularly in perceptual domains such as vision, speech and text. Neural networks have leveraged the increasing availability of large scale datasets and the acceleration and scale of computing resources in order to achieve ever increasing performance in problems like image classification, object detection, speech recognition, speech generation, machine translation, computer and board games, etc.

Though notorious, these advances have relied on the assumption that large amounts of data are available, or at least easily collectible (e.g. computer simulators). What happens when we want to create a classifier with state-of-the-art performance but only have a few labeled examples per class? One option would be to use classical methods with low model complexity such as Support Vector Machines or Gaussian Processes, though these would not be able to achieve the performance desired on tasks involving images, for example. On the other hand, we know that neural networks are able to produce very good representations of these types of high dimensional perceptual data, but only when given enough data. Trying to train a neural network on a small set of images per class would result in serious overfitting, rendering the model's generalisation ability nearly null. Though there are efforts in trying to make overparametrised neural networks learn from smaller datasets (e.g. [53]), there is still a long way to go before this can find widespread application.

The way to solve this problem in the 'neural network way' is to use more data, but in a different manner. Since we do not know what priors would help a neural network be successfully trained from scratch using a small dataset, we can use many small datasets in order to learn this prior, from data. This way, we can develop a system that learns how to generalise from a small dataset, by train on a large number of small datasets. This approach, known as *meta-learning*, has been of great interest for several decades, although its potential as a tool to solve few-shot

learning problems at a large scale has only been explored in recent years.

1.2 The problem with maximum-likelihood

A model trained by maximum-likelihood aims to find the parameters $\boldsymbol{w}$ that maximise the likelihood of the dataset $\mathcal{D}$, $\boldsymbol{w}_{ML} = \text{argmax}_{\boldsymbol{w}} p(\mathcal{D}|\boldsymbol{w}) = \text{argmax}_{\boldsymbol{w}} \prod_i p(\mathcal{D}_i|\boldsymbol{w})$. Once maximum-likelihood trained neural networks became notoriously good at supervised learning problems and their use started to become widespread in performance critical systems, such as self-driving cars or medical diagnosis, obtaining correct uncertainty estimates for the networks' predictions in a scalable manner became a topic of great importance and attention [15]. A problem with maximum-likelihood estimation is that we obtain a point estimate of the parameters, which does not provide a principled way to compute uncertainty over model parameters and predictions. A natural framework for principled uncertainty estimation is Bayesian inference, and while this is a challenging problem in neural networks, many modern approaches have been proposed to model the network's parameters' posterior, $p(\boldsymbol{w}|\mathcal{D})$, such as variational inference [5], ensembling [33] and stochastic gradient Markov-chain Monte-Carlo (MCMC) [66] . We describe the Bayesian approach to neural networks in more detail in Chapter 3.

As the performance of meta-learning-based few-shot learning algorithms using neural networks on standard benchmarks quickly improves (cf. Section 2.4.1), it is likely that such systems will eventually be used in the real-world, where accurate uncertainty estimation is paramount. For example, let's say we want to teach a robot how to execute a task by providing a single human demonstration. We naturally want the robot to execute the task well even under small environment changes. Perhaps more important, however, is that the robot knows when it does not know. For example, when the environment changed too much or when the human demonstration was not enough to provide good performance on all the state space. It is also important for the robot to know when it is being given faulty input, such as contradicting human demonstrations. As such, it is clearly important to build few-shot learning systems that can reason about its own uncertainty, which will be the main problem addressed in this work.

1.3 Contributions

In this work, we use the Learnet model [4] as our base maximum-likelihood few-shot learning model and generalise it to a hierarchical Bayesian model. Specifically, we adapt the main network of Learnet to a Bayesian neural network (BNN) using the scalable variational method proposed by Blundell et al. [5], but where the mean and standard deviation of the weights are conditioned on the few-shot examples. This can be seen as a conditional Bayesian neural network (CBNN), since the distribution of the weight is now dependent on the few-shot samples used for each task.

We make the following contributions:

- Extend Learnet to frame it as a (hierarchical) variational Bayesian formulation of meta-learning and derive the corresponding evidence lower bound to be optimised;
- Provide benchmarks for the performance of the original Learnet model when extended to K-shot N-way problems (the original work only dealt with binary one-shot learning);
- Propose a product-of-experts network as a more flexible alternative to mean pooling, proposed by Edwards and Storkey [10], in order-invariant data aggregation;
- Show that the Bayes-Learnet model allows us to estimate various uncertainties in few-shot learning and propose new tasks to evaluate Bayesian few-shot learning algorithms;
- Explore the effectiveness of Bayes-Learnet at improving data efficiency relative to Learnet by acting as a meta-regularizer;
- Explore the application of Bayes-Learnet to the problem of active learning in few-shot learning.

This work is organized as follows. In Chapter 2 we give an overview of the field of few-shot learning and meta-learning, describing in detail the main competing approaches. We focus exclusively on methods that use neural network as its main computational block. In Chapter 3 we motivate in more detail the need for uncertainty estimation in few-shot learning and formulate the contributions listed above. In Chapter 4 we explore our contributions, define new benchmarks and perform the necessary ablation studies.

Chapter 2

Meta-learning for few-shot learning

2.1 Meta-learning

Meta-learning, or learning to learn, has been a topic of great interest in the machine learning community for several decades [2, 8, 34, 44, 56, 61, 62]. One hallmark of human cognition is the ability to leverage previous knowledge to learn new tasks more quickly [57]. While one could put this into the area of transfer learning, where the learned representations are used a way to warm-start learning of a new task, meta-learning has a more fundamental appeal. It refers to the fact that not only do we become more knowledgeable as we learn, we also become more familiar with the process of learning – we learn how to learn. Even for skills that seem disconnected (e.g. learning Chinese *vs* learning German), it is possible to find a common set of primitives that lead to faster learning of this new skill. Naturally, we are interested in emulating this same ability in artificial intelligence systems.

Meta-learning often overlaps with other areas such as transfer learning, multi-task learning and domain adaptation, but we will not describe them in detail in this work, since advances and challenges in each of them could easily fill a dissertation alone.

2.2 Few-shot learning

Few-shot learning refers to the problem of learning to classify new object classes being given only a few examples (*shots*) of each new class. This is a fundamental problem in machine learning since, while large supervised learning datasets might contain thousands of different classes with many thousands of examples per class, the practical demands of an intelligent system might require it to quickly acquire a new task without being given a large number of examples. This is particularly important in applications where data is costly or hard to acquire, or in

non-stationary cases where an input might change target labels depending on the environment or other context.

While classical approaches to few-shot image recognition based on hand-crafted image descriptors exist [11, 32], given the success of neural networks in nearly every computer vision problem, it is only natural to try to develop neural network-based systems for few-shot learning. However, since the image representations in neural networks are not provided but learned, we have to learn features or mechanisms that will allow the system to adapt to the few-shot dataset provided. Therefore, we need to use datasets of datasets, so that the network can learn to learn a dataset at test time, which clearly falls into the meta-learning framework. As long as the system is well designed, one should be able to perform backpropagation through the whole neural network-based model, allowing us to learn feature representations that will be useful for a few-shot learning setting.

In this work, we only study the meta-learning models that are used for few-shot classification in the standard Omniglot [32] and miniImageNet [47] datasets, which have become the main benchmarks in the vision domain. In the next section we describe the formalism used in meta-learning-based few-shot learning methods, in a general way that encapsulates all the methods that will be discussed later.

While there is also a sizeable body of work in one-shot generation (e.g. [10, 48, 49]), we do not explore it in this work.

2.3 General training protocols and definitions

In standard supervised learning, we are given a training dataset composed of input-output pairs, $\mathcal{D}^{tr} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ and train a model f with parameters $\boldsymbol{w}$ to predict $\boldsymbol{y}$ given $\boldsymbol{x}$, $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{w})$. The model is then evaluated by measuring its performance on a test dataset $\mathcal{D}^{te}$.

In the few-shot learning case, however, we want to train the model on a small set of training points (which we call the *support set*, $\mathcal{D}^S$) so that it predicts the correct labels for a set of test points (which we call the *query set*, $\mathcal{D}^Q$). This can be seen as a meta-learning procedure, since the model has to learn how to adapt to the support set, i.e. it needs to learn how to learn. To do this, we provide many sets of support and query sets – datasets of datasets. Since each support-query set pair defines a different classification/regression problem, we define a *task* as $\mathcal{T} = (\mathcal{D}^S, \mathcal{D}^Q)$. Therefore, we want to train a model with parameters $\boldsymbol{w}$ to predict $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \mathcal{D}^S; \boldsymbol{w})$, $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}^Q$. However, while in the supervised learning case f is a neural network on which feed-forward computation is performed to predict $\boldsymbol{y}$, in meta-learning f usually includes some internal adaptation procedure that allows it to adapt to $\mathcal{D}^S$. This adaptation is analogous to the training phase in standard supervised learning, so we will refer to it as *training*, and evaluation of the test set is analogous to testing. The adaptation mechanism is the main differentiating factor between approaches, and it determines the quality of the

meta-learning algorithm. The parameters that change depending on the task are the *task-specific parameters*, and the parameters that do not change but control how the task-specific parameters are produced are the *task-agnostic parameters*. In order to learn the task agnostic parameters, we sample support-query set pairs from some task distribution $p(\mathcal{T})$ and optimise them so that, after training on the support set, the performance on the query set is high. This step is called *meta-training* and the tasks used are the *meta-training tasks*. In order to evaluate the meta-learning model, we sample a set of *meta-testing tasks* and perform training and testing. In general the meta-training and meta-testing tasks are drawn from the same task distribution, though this is not necessarily the case, and it is in fact an important problem to study. We refer to it briefly later in this work.

In few-shot classification, a problem is called *K-shot N-way* when the support set contains K points for N distinct classes (so there are $K \times N$ elements) and the query set contains K' points for the same N distinct classes. Testing then corresponds to determining to which of these N classes each of the elements in the query set correspond. A special case is that of binary K-shot learning, where instead of classifying each element of the query set into one of N classes, we classify them as either belonging or not to the single class of elements given in the support set (equivalent to $K = 1$).

2.4 Modern approaches

2.4.1 Model zoo

Due to the extremely fast pace of new publications in this area, we provide here a non-extensive list of the most relevant works from the past two years, for easy reference for the reader.

- Model-Agnostic Meta-Learning [12]
- Learnet [4]
- Differentiable closed-form solvers [3]
- Memory-Augmented Neural Networks [54]
- Siamese Networks [31]
- Optimization as a model for few-shot learning [47]
- Relation networks [59]
- Meta-networks [42]
- Conditionally shifted neurons [43]
- Neural Processes [18]

- Prototypical networks [58]
- Reptile [45]
- PLATIPUS [20]
- Matching networks [64]
- Meta-SGD [37]
- Bayesian MAML [28]
- SNAIL [41]
- Graph Neural Networks [17]
- MT-Net [36]

These methods can be broadly grouped into three categories: learning an optimiser, learning an initialiser and embedding learning. We now provide an overview of each of these approaches.

2.4.2 Learning an optimiser

The approach of learning a few-shot optimiser builds on previous work in meta-learning [1, 25, 65], where the numerical optimiser of a neural network (e.g. Adam [29]) is replaced by a neural network that performs the weight updates. Typically, a recurrent neural network is used as the meta-learner that learns how to update the prediction network's weights given the update history and the true gradients of the data loss with respect to the main network's parameters. This is simply an unrolling of the usual optimisation procedure, so the RNN can be trained by standard backpropagation through time (though some care has to be taken to avoid having to compute second-order gradients during backpropagation and to avoid explosion of model parameters – c.f. [1]). When applied to few-shot learning, Ravi and Larochelle [47] proposes using such an LSTM [24] to perform T update steps to the weights of the the prediction network, given elements of the training set, so that the prediction network performs well on a test set.

Though successful when it was first proposed, this approach is the one that has been used the least in subsequent work.

2.4.3 Learning an initialisation

One of the most influential recent works that has, arguably, led to the rapid increase in few-shot learning research produced in the past couple of years is Model-Agnostic Meta Learning (MAML) [12]. In MAML, the goal is to learn a weight initialisation so that any task can be learned by taking a small number of gradient descent steps on the support set. Though in theory the method requires computation of second-order gradients, it has been shown [45] that there's no reduction in performance when using a first-order approximation. One of the main advantages

of training by fine-tuning is that it provides consistency [13], i.e. the model will eventually converge to the correct solution if given enough support data. This is particularly important when applying few-shot learning to reinforcement learning problems or to out-of-distribution data during meta-test. However, recent models that do not use fine-tuning at test time [41] have shown to obtain superior performance in few-shot learning for reinforcement learning, so there is still room for debate.

Several have built on MAML in order to also do fast adaptation of a learning rate for each weight [37], learn which weights to freeze during training [36], and build Bayesian approximations [14, 20, 28].

Another related work is that of Bertinetto et al. [3], where instead of learning a weight initialisation and taking a few gradient steps at training time, the authors propose using all but the last layer of the network as task-agnostic parameters and solving just the linear/logistic regression problem from the penultimate layer embedding to the output in closed form, in a way that makes it possible to backpropagate through it during meta-training.

2.4.4 Metric and embedding learning

The last and perhaps most popular approach is that of learning an embedding. We place into this category all the models that do not perform fine-tuning during the training stage, that is, dynamic weights or support set embeddings are computed from the support set in a feed-forward fashion. For example, Prototypical Networks [58] use a differentiable k-nearest-neighbors loss to train a network that produces embeddings such that elements belonging to the same class are close in output space. Siamese networks [31] and Matching networks [64] also learn embedding/metric spaces so that elements of the same class are close in output space. There are also lines of work using memory-augmented networks to produce fast adaptation of a prediction network [42, 43, 54], temporal convolutions [41], and relation networks [59].

One of the advantages is that, since they are typically trained to optimise a relation score (like [4]) instead of an N-way softmax output (like MAML [12]), a trained model can be meta-tested for any value of N. Another advantage is that the embedding space learned can typically be interpreted and/or visualized, and that they are typically straightforward to extend to zero-shot learning (e.g. [58, 59]). A disadvantage is that these methods tend to not provide consistency, since providing an infinitely large support set does not usually improve the query set accuracy over just the support set size K that was used during training, which renders them less applicable to reinforcement learning settings.

2.5 Learnet

Here we describe in detail the Learnet model [4] which will be the base model on which we build the remainder of this work. In the original formulation, the model only does binary 1-shot

learning, since it simply predicts whether or not an element of the query set belongs to the class of the single element in the support set, but in Chapter 3 we propose an extension to K-shot N-way learning.

In Learnet, the element in the support set is passed to a neural network, h, whose output is a set of weights $\boldsymbol{w} = h(\mathcal{D}^S; \boldsymbol{w}^h)$. These dynamic weights $\boldsymbol{w}$ are used to parametrise the layer of a prediction network f that performs the binary prediction on the elements of the query set, $y = f(x; \boldsymbol{w}^f, \boldsymbol{w})$. Here $\boldsymbol{w}^f$ correspond to the weights that parametrise the remaining layers of the prediction network and which do not depend on the task. Hence the set of task-agnostic weights is $\boldsymbol{w}^h, \boldsymbol{w}^f$ and the task-specific weights are $\boldsymbol{w}(\mathcal{D}^h)$. We omit the task-agnostic weights from our notation throughout this work. The network that produces the weights of the prediction network is referred to as an *hypernetwork* [22] .

Naive hypernetwork

In its simplest form, the hypernetwork's output will be a whole weight matrix of the main network. This results in the number of parameters of the hypernetworks scaling as $O(l_i l_o d)$, for layers with l_i input units, l_o output units and a penultimate hypernetwork layer with dimension d (assuming the hypernetwork is a single linear layer). This scaling is very poor, and we can see that even for a small weight matrix of dimension 100×100 and $d = 100$, the last layer of the hypernetwork will have at least 10^6 parameters. This becomes impractical very quickly, so it is paramount that we choose an alternative parametrisation that does not result in an explosion of model parameters.

Factorised weights

In order to alleviate the issue above, Bertinetto et al. [4] introduce a special purpose weight factorisation. For a layer with input $\boldsymbol{x} \in \mathbb{R}^{l_i}$ and output $\boldsymbol{y} \in \mathbb{R}^{l_o}$, the naive parametrisation produces:

$$\boldsymbol{y} = \boldsymbol{w}(\mathcal{D}^S) \cdot \boldsymbol{x} + \boldsymbol{b}. \tag{2.1}$$

We omit a possible dependency of the bias on the support set for simplicity. However, a simplified factorisation of the weight matrix can be used, such as:

$$\boldsymbol{y} = M' \cdot \mathrm{diag}(\boldsymbol{w}(\mathcal{D}^S)) \cdot M \cdot \boldsymbol{x} + \boldsymbol{b}, \tag{2.2}$$

where $M \in \mathbb{R}^{l_i \times l_i}$, $M' \in \mathbb{R}^{l_o \times l_i}$ and $\boldsymbol{w}(\mathcal{D}^S) \in \mathbb{R}^{l_i}$. The matrices M and M' are now task-agnostic parameters, and the hypernetwork only produces a diagonal matrix with l_i elements. With this factorisation, the last layer of the hypernetwork only has $O(l_i d)$ parameters. This results in a linear scaling with the number of hidden units, instead of quadratic, making it much more scalable to larger layers and deeper networks. A slight inconvenience of this parametrisation is

that the total number of parameters of the layer has increased from $l_o \times l_i$ to $l_o \times l_i + l_i^2 + l_i$. This can lead to wrong conclusions being drawn when comparing the factorisations since the number of parameters of the prediction network will have changed. To fix this, we propose using matrices of dimension $M \in \mathbb{R}^{k \times l_i}$, $M' \in \mathbb{R}^{l_o \times k}$ and $\boldsymbol{w}(\mathcal{D}^S) \in \mathbb{R}^k$, with $k = \frac{l_i \times l_o}{l_i + l_o + 1}$. This way we keep a total number of $l_o \times l_i$ parameters in the prediction network's layer.

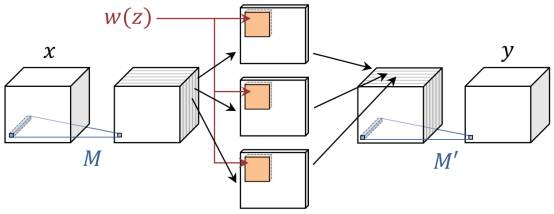


Figure 2.1: Convolution layer factorisation. Figure taken from Bertinetto et al. [4].

For convolutional layers, a similar factorisation can be formulated. For a layer with input $\boldsymbol{x} \in \mathbb{R}^{h \times w \times l_i}$ and output $\boldsymbol{y} \in \mathbb{R}^{h' \times w' \times l_o}$ (here l_i and l_o denote the number of channels), the naive parametrisation produces:

$$\boldsymbol{y} = \boldsymbol{w}(\mathcal{D}^S) * \boldsymbol{x} + \boldsymbol{b}. \tag{2.3}$$

where $\boldsymbol{w}(\mathcal{D}^S) \in \mathbb{R}^{c \times c \times l_i \times l_o}$ and c is the filter size. The factorisation proposed by Bertinetto et al. [4] takes the form:

$$\boldsymbol{y} = M' * \boldsymbol{w}(\mathcal{D}^S) *_d M * \boldsymbol{x} + \boldsymbol{b}. \tag{2.4}$$

where $M \in \mathbb{R}^{1 \times 1 \times l_i \times l_i}$, $M' \in \mathbb{R}^{1 \times 1 \times l_i \times l_o}$ and $\boldsymbol{w}(\mathcal{D}^S) \in \mathbb{R}^{c \times c \times l_i}$. This factorisation results in the matrices M and M' performing 1×1 convolutions on its respective inputs maps, while $*_d$ denotes a convolution where each kernel of $\boldsymbol{w}$ is applied to a single channel in its input. Computationally, this is implemented by making $\boldsymbol{w}$ diagonal in the third and fourth dimensions. See Figure 2.1 for a visual depiction.

Training

Since this model was formulated for binary 1-shot learning, the authors train the model simply by sampling batches of two datapoints from the meta-training set, $\{((\boldsymbol{x}^S, \boldsymbol{y}^S), (\boldsymbol{x}^Q, \boldsymbol{y}^Q))_i\}_i^N$, and minimizing a binary cross-entropy loss with respect to all the task agnostic parameters:

$$\frac{1}{N} \sum_i^N \mathcal{L}(f(\boldsymbol{x}_i^Q; \boldsymbol{w}(\boldsymbol{x}_i^S)), \mathbb{I}(\boldsymbol{y}^S = \boldsymbol{y}_i^Q)). \tag{2.5}$$

In order to guarantee that the classes are balanced, pairs of points with the same and different labels are drawn in the same proportion. In Chapter 3 we provide an extention of this loss to the K-shot N-way problem.

Chapter 3

Bayesian Learnet

In this chapter we generalise the base Learnet model [4] in order to frame it as hierarchical Bayesian inference, and introduce the various changes made in order to adapt it to K-shot N-way few-shot classification problems.

3.1 Understanding the sources of uncertainty in few-shot learning

We can see inference in a few-shot learning model as being composed of two stages. In the first stage, the model builds a representation (explicit or implicit) of the support set provided. In the second stage, this representation is used to produce the predictions for the elements in the query set. Therefore, we can identify two main areas where uncertainty can be encoded:

1. *Task uncertainty* refers to the part of the predictive uncertainty that comes from uncertainty in the representation of the support set. For example, if the task is to infer the shape of a sinusoidal curve for varying amplitude given just a K-shot support set of noisy points drawn from the curve, task uncertainty would refer to the uncertainty of the model in the prediction it makes about the curve's amplitude (as a latent variable), so we have a posterior distribution over tasks.

2. *Weight uncertainty* refers to the part of the predictive uncertainty that comes from the uncertainty in the weights of the neural network through which the query set points will be passed. For example, in the same sinusoidal task, once we have a latent task prediction, we can have many neural network parameters that yield similar curves, so it is natural to have a posterior distribution over weights, conditional or not on the task.

The most principled framework for uncertainty estimation is Bayesian inference. Below we provide an overview of Bayesian inference and the main approaches that incorporate it into neural networks, with emphasis on variational inference.

3.2 Variational Bayesian neural networks

The overwhelming majority of neural network models are trained by maximum-likelihood, i.e. the trained parameters are obtained by maximizing the likelihood of the data:

$$\boldsymbol{w}_{ML} = \text{argmax}_{\boldsymbol{w}} p(\mathcal{D}|\boldsymbol{w}) = \text{argmax}_{\boldsymbol{w}} \prod_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}). \tag{3.1}$$

This maximization can be efficiently performed by backpropagation [35]. When using a prior $p(\boldsymbol{w})$, we obtain the maximum-a-posteriori solution:

$$\boldsymbol{w}_{MAP} = \text{argmax}_{\boldsymbol{w}} p(\boldsymbol{w}|\mathcal{D}) = \text{argmax}_{\boldsymbol{w}} p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}). \tag{3.2}$$

In the case of neural networks, we want the expression above to be differentiable, so we are typically restricted to the use of smooth priors, such as zero-mean Gaussians.

Both approaches above only provide a point estimate of the parameters. However, we are interested in the posterior distribution $p(\boldsymbol{w}|\mathcal{D})$ so we can estimate the uncertainty of the model parameters, and which will allow us to perform Bayesian inference by computing the predictive distribution for a test point:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*,\mathcal{D}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*;\boldsymbol{w})p(\boldsymbol{w}|\mathcal{D})d\boldsymbol{w}. \tag{3.3}$$

Using Bayes' rule the posterior is given by

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{\int p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w}}. \tag{3.4}$$

For neural networks, the denominator of the above expression is intractable, which poses a difficulty to estimating the posterior. This means we have to resort to approximate inference. The most common approximate inference methods are Markov-chain Monte Carlo (MCMC) [52] and variational inference [26]. Here we discuss only variational inference as it is a more scalable method than MCMC so its use for estimating neural network posteriors is more widespread.

In variational inference we use a distribution $q(\boldsymbol{w}|\mathcal{D};\phi)$ to approximate the true posterior $p(\boldsymbol{w}|\mathcal{D})$. This can be done by minimizing the KL-divergence between the two distributions $KL(q(\boldsymbol{w}|\mathcal{D};\phi)\|p(\boldsymbol{w}|\mathcal{D}))$ with respect to the parameters ϕ. Noting that the KL-divergence is a positive quantity, this can be seen as maximizing an evidence lower-bound (ELBO) [26, 30, 50]:

$$p(\mathcal{D}) \geq \mathcal{L} = \mathbb{E}_{q(\boldsymbol{w}|\mathcal{D};\phi)} \log p(\mathcal{D}|\boldsymbol{w}) + KL(q(\boldsymbol{w}|\mathcal{D};\phi)\|p(\boldsymbol{w})). \tag{3.5}$$

The term $\mathbb{E}_{q(\boldsymbol{w}|\mathcal{D};\phi)} \log p(\mathcal{D}|\boldsymbol{w})$ in (3.5) is intractable, so it must be estimated by Monte-Carlo estimation. However, drawing samples from $\boldsymbol{w} \sim q(\boldsymbol{w}|\mathcal{D};\phi)$ would typically render optimisation with respect to ϕ infeasible due to the sampling step.

In order to solve this, Kingma and Welling [30], Rezende et al. [50] show that if the sample $\boldsymbol{w} \sim q(\boldsymbol{w}|\mathcal{D};\phi)$ can be written as a deterministic function of random noise, $\boldsymbol{w} = g(\boldsymbol{\epsilon};\phi)$, for some random variable $\boldsymbol{\epsilon} = p(\boldsymbol{\epsilon})$, is possible to backpropagate gradients through the samples, which allows deep neural networks to be easily incorporated into variational inference. While this *reparametrisation trick* was originally proposed in the context of latent variable models like variational autoencoders (where one wants to approximate the posterior over hidden variables $p(\boldsymbol{z}|\boldsymbol{x})$), [5] used it to obtain unbiased estimates of the gradient of (3.5) w.r.t ϕ, where the weight posterior of the neural network is parametrised as Gaussian with diagonal convariance, $q(\boldsymbol{w}|\mathcal{D};\phi) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$, so a sample can be written as $\boldsymbol{w} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}$, where $\boldsymbol{\epsilon} \sim N(0, 1)$.

Earlier works [21, 23] propose similar variational approximations to the weights' posterior, without making explicit use of the stochastic backpropagation formulation. Using recent developments in scalable variational inference, many works have been proposed to build more accurate and flexible variational posteriors [38, 39, 46, 51].

3.3 Few-shot learning as hierarchical Bayes

Since variational BNN's are able to approximately model the posterior over weights of a neural network, they are a natural fit for estimating the posterior over weights of a neural network that is being trained for few-shot learning. However, we need to incorporate task information into the model so that we obtain a neural network that is conditional on the task, ideally without causing a large increase in the number of model parameters.

We formulate meta-learning as a hierarchical Bayesian model with a joint distribution of the form:

$$p(\mathcal{D}, \boldsymbol{w}, \boldsymbol{z}) = p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z}), \tag{3.6}$$

where $\boldsymbol{z}$ are the hidden variables corresponding to a task embedding/descriptor, which will be inferred by the model, and $\boldsymbol{w}$ are the task-specific weights of the prediction network, $P(\mathcal{D}|\boldsymbol{w})$ (we will omit task-agnostic weights of the prediction network throughout). We note that this formulation also provides a way to incorporate known task embeddings or task embedding distributions, treating $\boldsymbol{z}$ as a visible variable on which the weights $\boldsymbol{w}$ are dependent. This would make this formulation transfer trivially to zero-shot learning, where support sets are not provided, but task embeddings are. See Fig. (*make graphical model*) for the graphical model depiction in plate notation.

By modeling the joint $p(\mathcal{D}, \boldsymbol{w}, \boldsymbol{z})$ we can do Bayesian inference on a new datapoint $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ by computing the predictive distribution:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \mathcal{D}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*; \boldsymbol{w})p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D})d\boldsymbol{w}d\boldsymbol{z}. \tag{3.7}$$

To compute the expression above we need to model the posterior:

$$p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z})}{\int p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{w}d\boldsymbol{z}}. \tag{3.8}$$

Since its denominator is intractable, in order to approximate this posterior we resort to structured variational inference with an approximating distribution of the form $p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D}) \approx q(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D}) = q(\boldsymbol{w}|\boldsymbol{z})q(\boldsymbol{z}|\mathcal{D})$.

As is standard in variational inference, we want to optimise q's parameters by minimizing the KL-divergence between the approximate and the true posterior:

$$\phi^*, \theta^* = \text{argmin}_{\phi,\theta}\text{KL}(p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D})\|q(\boldsymbol{w}|\boldsymbol{z}; \theta)q(\boldsymbol{z}|\mathcal{D}; \phi)). \tag{3.9}$$

Minimizing this KL-divergence is equivalent to maximizing an ELBO given by:

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D};\phi)}\left[\mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z},\mathcal{D};\theta)} \log p(\mathcal{D}|\boldsymbol{w}) - \text{KL}(q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D}; \theta)\|p(\boldsymbol{w}|\boldsymbol{z}))\right] - \text{KL}(q(\boldsymbol{z}|\mathcal{D}; \phi)\|p(\boldsymbol{z})) \tag{3.10}$$

where the evidence (or marginal likelihood) of a dataset is given by:

$$p(\mathcal{D}) = \int p(\boldsymbol{z}) \left[\prod_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{z})d\boldsymbol{w}\right] d\boldsymbol{z}. \tag{3.11}$$

Proof can be found in Appendix 6.1. We simplify this ELBO by assuming $q(\boldsymbol{w}|\boldsymbol{z}; \mathcal{D}) = q(\boldsymbol{w}|\boldsymbol{z})$ and $p(\boldsymbol{w}|\boldsymbol{z}) = p(\boldsymbol{w})$:

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D},\phi)}\left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z};\theta)} \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) - \text{KL}(q(\boldsymbol{w}|\boldsymbol{z}; \theta)\|p(\boldsymbol{w}))\right] - \text{KL}(q(\boldsymbol{z}|\mathcal{D}; \phi)\|p(\boldsymbol{z})). \tag{3.12}$$

For the posteriors we use Gaussian distributions with diagonal covariance,

$$q(\boldsymbol{z}|\mathcal{D}; \phi) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{z}}(\mathcal{D}), \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{z}}(\mathcal{D}))) \quad , \quad q(\boldsymbol{w}|\boldsymbol{z}; \theta) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}(\boldsymbol{z}), \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{w}}(\boldsymbol{z}))). \tag{3.13}$$

For numerical stability, following [5] the standard deviations are parametrised as $\boldsymbol{\sigma} = \log(1 + \exp(\boldsymbol{\rho}))$, where the $\boldsymbol{\mu}$'s and $\boldsymbol{\rho}$'s of each distribution are the output of neural networks with parameters ϕ and θ.

If we use Gaussian priors $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0}, \gamma^2\boldsymbol{I})$ and $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})$, we can write the KL terms in closed form [30]:

$$\text{KL}(q(\boldsymbol{w}|\boldsymbol{z}; \theta)\|p(\boldsymbol{w})) = \frac{1}{2}\sum_j \left(1 + 2\log\sigma_{\boldsymbol{w},j} - \frac{(\mu_{\boldsymbol{w},j})^2 + (\sigma_{\boldsymbol{w},j})^2}{\gamma^2}\right) \tag{3.14}$$

$$\text{KL}(q(\boldsymbol{z}|\mathcal{D}; \phi)\|p(\boldsymbol{z})) = \frac{1}{2}\sum_j \left(1 + 2\log\sigma_{\boldsymbol{z},j} - (\mu_{\boldsymbol{z},j})^2 - (\sigma_{\boldsymbol{z},j})^2\right). \tag{3.15}$$

However, since the expectation terms in (3.5) are once again intractable, we use a single Monte-Carlo sample from the corresponding Gaussian posterior to estimate it. This results in the approximate ELBO for a single dataset:

$$\tilde{\mathcal{L}} \approx \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}) - \frac{1}{2}\sum_j \left(1 + 2\log\sigma_{\boldsymbol{w},j} - \frac{(\mu_{\boldsymbol{w},j})^2 + (\sigma_{\boldsymbol{w},j})^2}{\gamma^2}\right) - \\ - \frac{1}{2}\sum_j \left(1 + 2\log\sigma_{\boldsymbol{z},j} - (\mu_{\boldsymbol{z},j})^2 - (\sigma_{\boldsymbol{z},j})^2\right), \tag{3.16}$$

where $\boldsymbol{w} \sim q(\boldsymbol{w}|\boldsymbol{z};\theta)$ and $\boldsymbol{z} \sim q(\boldsymbol{z}|\mathcal{D};\phi)$. Since all the samples drawn are from Gaussian distributions, using the reparametrisation trick this equation can be optimised end-to-end with stochastic backpropagation. While Bayes-by-Backprop (BbB) [5] use a scale mixture of two Gaussians with mean zero as the weight prior, we opt for a simple Gaussian prior. We note, however, that using a Gaussian prior in our case is not completely equivalent to Gaussian prior in BbB, since in our case the prior will influence not only the distribution over weights for each dataset, but the distribution over weights across *all* datasets. This means that the prior must be weak enough so that the conditional mean and variance can change across datasets and not default to a single constant that satisfies the latent cost. We note that, though possible, we do not make a Bayesian treatment of the variational parameters ϕ and θ.

3.3.1 Adapting the ELBO for few-shot learning

So far we have used ELBO equation where all the elements of a dataset $\mathcal{D}$ are both passed to the task network, $q(\boldsymbol{z}|\mathcal{D};\phi)$, and evaluated by the likelihood function of the BNN, $\log p(\mathcal{D}|\boldsymbol{w})$. However, this does not correspond to a few-shot learning setting, since the same data would be used for inferring the task and making predictions. Instead, we adapt (3.5) so that the task network only takes as input the support set $\mathcal{D}^S$ and the BNN only takes as input the query set, $\mathcal{D}^Q$:

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D}^S;\phi)}\left[\mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z};\theta)} \log p(\mathcal{D}^Q|\boldsymbol{w}) - \mathrm{KL}(q(\boldsymbol{w}|\boldsymbol{z};\theta)\|p(\boldsymbol{w}))\right] - \mathrm{KL}(q(\boldsymbol{z}|\mathcal{D}^S;\phi)\|p(\boldsymbol{z})), \tag{3.17}$$

which yields an approximate ELBO:

$$\tilde{\mathcal{L}} \approx \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}^Q} \log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}) - \frac{1}{2}\sum_j \left(1 + 2\log\sigma_{\boldsymbol{w},j} - \frac{(\mu_{\boldsymbol{w},j})^2 + (\sigma_{\boldsymbol{w},j})^2}{\gamma^2}\right) - \\ - \frac{1}{2}\sum_j \left(1 + 2\log\sigma_{\boldsymbol{z},j} - (\mu_{\boldsymbol{z},j})^2 - (\sigma_{\boldsymbol{z},j})^2\right), \tag{3.18}$$

where $\boldsymbol{w} \sim q(\boldsymbol{w}|\boldsymbol{z};\theta)$ and $\boldsymbol{z} \sim q(\boldsymbol{z}|\mathcal{D}^S;\phi)$ This way our model complies with the standard few-shot learning practices and we can use the training protocol outlined in Chapter 2 to train

it.

We see that there are three distinct parts of this model: the Bayesian neural network, $p(\mathcal{D}|\boldsymbol{w})$, the hypernetwork, $q(\boldsymbol{w}|\boldsymbol{z},\theta)$, and the task network, $q(\boldsymbol{z}|\mathcal{D},\theta)$. Together, they form a conditional Bayesian neural networks (CBNN).

3.3.2 Full-data ELBO and KL weighing

The ELBO in (3.17) is for one dataset/task only. The full-data ELBO is obtained by summing the ELBO's of all the tasks in our meta-training set. However, since the datasets used to compute the likelihood term in (3.18) are relatively small, the KL terms will be orders of magnitude greater than the likelihood term. If optimised in the current form, these KL terms quickly overpower the likelihood term, pushing the means $\boldsymbol{\mu}_{\boldsymbol{w}}$ and $\boldsymbol{\mu}_{\boldsymbol{z}}$ to zero and preventing any type of learning from happening. In order to prevent this, we add a weighing factor β to the KL terms:

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D}^S;\phi)}\left[\mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z};\theta)} \log p(\mathcal{D}^Q|\boldsymbol{w}) - \beta \mathrm{KL}(q(\boldsymbol{w}|\boldsymbol{z};\theta)\|p(\boldsymbol{w}))\right] - \beta \mathrm{KL}(q(\boldsymbol{z}|\mathcal{D}^S;\phi)\|p(\boldsymbol{z})), \tag{3.19}$$

which can be used to control the strength of the latent regularization.

3.3.3 Decomposing the model

Since by stochastic backpropagation the gradients of the BNN will flow through the hypernetwork and into the task network, in order to make this system work it is necessary that all parts work as intended independently. This means that the BNN and hypernetwork must be able to get high likelihood on the training data given the ground truth task descriptors $\boldsymbol{z}$, and the task network must be able to infer good task descriptors that are highly correlated with ground-truth descriptors, if such exist. If either of these components does not work on its own, the end-to-end trained model will not be able to learn as intended.

An advantage of this model is that, due to its clean variational formulation, we can test individual parts of the model by assuming the others are known. For example, if we have some ground-truth task descriptors $\boldsymbol{z}_{\mathcal{D}}$, we can check whether our BNN + hypernetwork architecture choice is appropriate by removing the $\boldsymbol{z}$ posterior in (3.5):

$$\hat{\mathcal{L}} = \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z}_{\mathcal{D}};\theta)} \log p(\mathcal{D}|\boldsymbol{w}) - \mathrm{KL}(q(\boldsymbol{w}|\boldsymbol{z}_{\mathcal{D}};\theta)\|p(\boldsymbol{w})), \tag{3.20}$$

and optimising this simplified lower bound.

Conversely, we can check whether our task network architecture is appropriate by doing maximum likelihood estimation on the ground-truth descriptors:

$$\hat{\mathcal{L}} = \log q(\boldsymbol{z}_{\mathcal{D}}|\mathcal{D};\phi). \tag{3.21}$$

Besides being able to check that each part of the model is working as intended, we can also assess the influence of the probabilistic formulation by replacing some of the distribution by point estimates that come from a neural network output. We can compare how using a deterministic/stochastic variable affects training by removing/not removing the corresponding expectations and KL terms in (3.5). For example, in order to see how the stochasticity in $\boldsymbol{z}^{(i)} \sim q(\boldsymbol{z}|\mathcal{D};\phi)$ affects training, we can compare it with using a deterministic function $\boldsymbol{z} = f_\phi(\boldsymbol{z}|\mathcal{D})$ and optimising the simplified ELBO:

$$\hat{\mathcal{L}} = \mathrm{KL}(q(\boldsymbol{w}|f(\boldsymbol{z}|\mathcal{D};\phi);\theta)\|p(\boldsymbol{w})) - \mathbb{E}_{q(\boldsymbol{w}|f(\boldsymbol{z}|\mathcal{D};\phi);\theta)} \log p(\mathcal{D}|\boldsymbol{w}). \tag{3.22}$$

Since using stochastic variables introduces a number of difficulties both in terms of design (choosing an appropriate prior, re-weighting the KL term, etc.) and training (more unstable learning) it is useful to first make sure the model works correctly when using deterministic variables. A fully deterministic model defaults to the Learnet architecture.

3.4 The task network architecture

We want to compute the approximate task posterior given a dataset, $q(\boldsymbol{z}|\mathcal{D}^S)$, in a way that is invariant to the ordering of the datapoints in $\mathcal{D}^S$. Despite its apparent simplicity, choosing an architecture that is able to accurately predict this posterior is far from trivial.

3.4.1 Mean aggregation

The simplest approach is that proposed by Edwards and Storkey [10]: each datapoint $\mathcal{D}_i^S$ is passed through the same neural network in order to create a point embedding $\boldsymbol{e}_i$. These embeddings are then pooled together by computing the mean across embeddings, $\hat{\boldsymbol{e}} = \sum_i \boldsymbol{e}_i$. This is then be passed to another neural network to compute the sufficient statistics of $q(\boldsymbol{z}|\mathcal{D}^S)$, such as mean and variance in the case of a Gaussian posterior:

$$q(\boldsymbol{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}(\hat{\boldsymbol{e}}), \mathrm{diag}(\boldsymbol{\sigma^2}(\hat{\boldsymbol{e}}))). \tag{3.23}$$

The Neural Statistician is an unsupervised model, so the datasets fed to $q(\boldsymbol{z}|\mathcal{D}^S)$ are only composed of input samples without labels, $\mathcal{D}^S = \{\boldsymbol{x}_i\}_i^N$. This can be adapted for few-shot classification tasks, but a way to use it for regression tasks has not been proposed. Since we want our model to work both for classification and regression tasks, the dataset passed to the task network contains both the inputs and the labels, $\mathcal{D}^S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_i^N$. This adds another degree of complexity to the design of the task network since the network must be able to, for the same sets of inputs, infer different tasks depending on the corresponding labels. In Sections 3.6 and 3.7 we will propose ways to deal with this issue for regression and classifcation problems,

respectively.

3.4.2 Product-of-experts posterior

Though simple, the aggregation function proposed by Edwards and Storkey [10] fails to capture how much each point contributes to the posterior distribution. Ideally, we want to be able to evaluate the posterior belief of each datapoint, $q(\boldsymbol{z}|\mathcal{D}_i)$, and pool this information from each datapoint to construct the final posterior. As such, in this work we use a product of experts model as used by Vedantam et al. [63], where each datapoint instantiates an expert. The posterior is then written as $q(\boldsymbol{z}|\mathcal{D}) \propto \prod_i q(\boldsymbol{z}|\mathcal{D}_i)$. For Gaussian experts, $q(\boldsymbol{z}|\mathcal{D}_i) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_i(\mathcal{D}_i), \boldsymbol{C}_i(\mathcal{D}_i))$, the dataset posterior can be written in closed form:

$$q(\boldsymbol{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{C}), \quad \boldsymbol{\mu} = \boldsymbol{C}\left(\sum_i \boldsymbol{C}_i^{-1}\boldsymbol{\mu}_i\right), \quad \boldsymbol{C}^{-1} = \sum_i \boldsymbol{C}_i^{-1}. \tag{3.24}$$

This formulation allows us not only to quantify the uncertainty coming from each datapoint, but also to use a dataset of any size during training or testing in a principled way. Due to the contribution of further $\boldsymbol{C}_i$ terms to the posterior covariance $\boldsymbol{C}$, the more datapoints we have, the narrower the posterior will be. Since we want the posterior to be invariant to the order of the datapoints, we use a shared expert network for all datapoints, as before. We note that mean aggregation corresponds to the limit case where every expert has identity covariance, $\boldsymbol{C}_i = \boldsymbol{I}, \forall i$. For simplicity and in order to maintain end-to-end differentiability we use experts with diagonal covariance.

3.4.3 Normalized product-of-experts

One drawback of this product-of-experts model is that if the same point is provided twice, the posterior in $\boldsymbol{z}$ will become sharper than if the point was given just once, even though we did not gain any information about the distribution of $\boldsymbol{z}$ because of the repeated point. We can easily see this for a 1D case with 2 datapoints $\mathcal{D} = \mathcal{D}_1, \mathcal{D}_2$, where $\mathcal{D}_1 = \mathcal{D}_2$. If $q(z|\mathcal{D}_1) = q(z|\mathcal{D}_2) = \mathcal{N}(z|\mu, \sigma^2)$, the product of experts will be $q(z|\mathcal{D}) = \mathcal{N}(z|\mu, \sigma^2/2)$. Therefore, the uncertainty of the posterior was reduced even though there was no real gain in information.

In order to deal with this we normalize the multiplicative contribution of the experts by taking the n-root of the product, $q(\boldsymbol{z}|\mathcal{D}) \propto (\prod_i^n q(\boldsymbol{z}|\mathcal{D}_i))^{1/n}$. This results in a corrected product of experts of the form:

$$q(\boldsymbol{z}|\mathcal{D}) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, n\,\boldsymbol{C}), \tag{3.25}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{C}$ are the same as given in (3.24).

3.4.4 Other aggregation methods

A pooling operation is not the only way to aggregate information about the support set. One other option is to use a recurrent neural network [60] or temporal convolution [41] that at each time step receives a datapoint from $\mathcal{D}^S$ and updates a recurrent state. However, order invariance is not built into the architecture and must be learned by the model. For this reason we do not apply these methods in our model.

Further experiments using relation networks [55] can be found in Appendix 6.2

3.5 The hypernetwork architecture

Following the Bayes-By-Backprop formulation [5], we model the task-specific weights of the prediction network as a Gaussian variable with diagonal covariance matrix, conditional on $\boldsymbol{z}$. Therefore, we can see the weights as conditionally independent given $\boldsymbol{z}$:

$$q(\boldsymbol{w}|\boldsymbol{z};\theta) = \prod_i q_\theta(w_i|\boldsymbol{z}) = \prod_i \mathcal{N}(w_i|\mu_i(\boldsymbol{z}), \sigma_i^2(\boldsymbol{z})), \tag{3.26}$$

where $[\boldsymbol{\mu}(\boldsymbol{z}), \boldsymbol{\rho}(\boldsymbol{z})]$ is the output of a neural network with input $\boldsymbol{z}$. In Bayes-By-Backprop, since the mean and variance are not dependent on $\boldsymbol{z}$, there is only in a $2\times$ increase in the number of model parameters, though in our case the increase in parameters will depend on how we choose to build the hypernetwork.

Naive hypernetwork

In the simplest form the mean and variances of all the weights in a layer are produced by the hypernetwork. Though this scales very poorly, we will still experiment with this approach in Chapter 4 since it allows us to draw interesting conclusions.

factorised weights

As described in Section 2.5, Bertinetto et al. [4] proposed a weight factorisation in order to prevent an explosion of parameters in the hypernetwork. In this parametrisation, only a fraction of the means and variances of the weights are produced by the hypernetwork, with the remaining parameters of the layer being task-agnostic. This leaves a question as to how we treat task-agnostic weights in our CBNN model. We will refer to weights produced by the hypernetwork as dynamic weights, and the remaining as static weights. Regarding the static weights, we can either parametrise them with a static mean and variance associated with each weight just like in Bayes-by-Backprop [5], or just use deterministic weights. While the latter partially breaks the Bayesian formulation of the neural network, since now we are not modeling the posterior $q(\boldsymbol{w}|\boldsymbol{z};\theta)$ over the full set of weights, we found that this results in much better training stability, while providing satisfactory posterior distributions. In fact, in Bayes-by-Backprop the authors

show that the vast majority of weights in a standard Bayesian neural network become redundant due to low signal to noise ratio (i.e. most of the weights can be removed without loss of accuracy). This supports the idea that, in practice, we do not need to make every single weight stochastic, which aligns with our weight factorisation approach.

As in Learnet, only a single layer is parametrised by a hypernetwork, which results in a neural network with one layer of the form $M' \cdot \text{diag}(\boldsymbol{w}) \cdot M \cdot \boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{w} \sim q(\boldsymbol{w}|\boldsymbol{z}; \theta)$, and M', M and $\boldsymbol{b}$ are task-agnostic parameters, and the remaining layers of the standard form $\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}$, where both $\boldsymbol{w}$ and $\boldsymbol{b}$ are task-agnostic parameters. The design for convolutional networks is analogous. This means that only a very small fraction of the total number of weights of the neural network is actually stochastic and conditional on the task, though we found that this is enough to provide good uncertainty estimates while maintaining accuracy competitive with the fully deterministic model.

3.6 Setup for regression problems

For regression problems, implementation is relatively straightforward. Since we only use regression for 1D regression tasks, the input to each expert in the task network is simply the concatenation of the input and target, $[\boldsymbol{x}, \boldsymbol{y}]$. In the case that $\boldsymbol{x}$ and $\boldsymbol{y}$ have different dimensionality, passing them to the task network might require extra manipulation, though we do not investigate that case here.

Using a prediction network with linear output layer, $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{w}(\mathcal{D}^S))$, we use a simple Gaussian likelihood for training:

$$\log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}(\mathcal{D}^S)) = -\frac{1}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma} \|\boldsymbol{y} - \hat{\boldsymbol{y}}_j\|^2, \tag{3.27}$$

for $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}^Q$. We can choose to use a constant residual or produce it as an additional output of the prediction network as $[\hat{\boldsymbol{y}}, \log \hat{\sigma}] = f(\boldsymbol{x}; \boldsymbol{w}(\mathcal{D}^S))$. In this work, we always use a constant residual to simplify analysis of the other variance estimations in the model.

At test time, in order to get Bayesian predictions, we compute the predictive distribution via Monte-Carlo estimation:

$$p(\hat{\boldsymbol{y}}) = \frac{1}{T} \sum_{t=1}^{T} \mathcal{N}(\hat{\boldsymbol{y}}|\hat{\boldsymbol{y}}^{(t)}, \sigma^2), \tag{3.28}$$

where $\hat{\boldsymbol{y}}^t = f(\boldsymbol{x}; \boldsymbol{w}^{(t)})$, $\boldsymbol{w}^{(t)} \sim q(\boldsymbol{w}|\boldsymbol{z}^{(t)})$ and $\boldsymbol{z}^{(t)} \sim q(\boldsymbol{z}|\mathcal{D}^S)$. The expected value of this predictive distribution is:

$$\mathbb{E}(\hat{\boldsymbol{y}}) = \frac{1}{T} \sum_{t=1}^{T} \hat{\boldsymbol{y}}^{(t)}, \tag{3.29}$$

and the variance can be estimated, according to Kendall and Gal [27], as:

$$\text{var}(\hat{\boldsymbol{y}}) \approx \sigma^2 + \frac{1}{T}\sum_{t=1}^{T}(\hat{\boldsymbol{y}}^{(t)})^T\hat{\boldsymbol{y}}^{(t)} - \mathbb{E}(\hat{\boldsymbol{y}})^T\mathbb{E}(\hat{\boldsymbol{y}}). \tag{3.30}$$

3.7 Setup for classification problems

For classification problems, there are several choices we can make for the type of likelihood function used, depending on the task at hand. In classification models it is common to have the network's output be a softmax over the N classes, for an N-way problem. This means that an output unit can correspond to any class depending on the input-label correspondence of the classes used on a given support set. However, in the context of our Bayesian Learnet, this would require passing both the support images and the 1-hot vector of its classes to the task network, which would have to find a 1-hot, permutation invariant representation of the image-label association, and produce weights that respond accordingly. This is an extremely hard pattern for the task network to learn, so instead we pass the K-shots of each of the N classes through the Learnet and obtain a binary relation score (like in [4, 59]) as the output of the main network, that indicates whether an image in the query set belongs to the class on the support set. This is a simple extension of the binary one-shot approach used originally in the Learnet model.

At training time, the model is trained to predict the correct label binary l for tuples $(\boldsymbol{x}, \mathcal{D}^{S,c})$, where all elements in the support set $\mathcal{D}^S$ belong to the same class c, and $l = 1$ if the query element $\boldsymbol{x}$ belongs to c and 0 otherwise. We use a prediction network $f(\boldsymbol{x}; \boldsymbol{w}(D^{S,c}))$ with sigmoid output and logistic loss. In order to prevent classification bias, during training we always sample the same number of tuples with $l = 1$ and $l = 0$.

At test time, given a support set with K items in each the N classes, for each element in query set we compute a distribution over classes as:

$$\hat{\boldsymbol{p}} = \text{softmax}(\text{logit}(\hat{\boldsymbol{l}})) \tag{3.31}$$

where $\hat{l}_c = f(\boldsymbol{x}; \boldsymbol{w}(D^{S,c})) \equiv p(l_c = 1|\boldsymbol{x}; \boldsymbol{w}(D^{S,c}))$, and $\text{logit}(\cdot)$ denotes the logit of the sigmoid output.

For the Bayesian model, when computing the predictive distribution we use Monte-Carlo estimation to produce an average over softmaxes for T samples:

$$\hat{p}_c = \frac{1}{T}\sum_{t=1}^{T}\hat{p}_c^{(t)} \tag{3.32}$$

where now $\hat{l}_c^{(t)} = p(l = 1|\boldsymbol{x}; \boldsymbol{w}^{(t)})$, $\boldsymbol{w}^{(t)} \sim q(\boldsymbol{w}|\boldsymbol{z}^{(t)})$ and $\boldsymbol{z}^{(t)} \sim q(\boldsymbol{z}|\mathcal{D}^{S,c})$,

Because of this training procedure, a model trained for a particular K can be used for N-way

classification with any N, which is an advantage it shares with other embedding learning methods. Gradient-based methods, on the other hand, have a fixed softmax output layer, so can only perform classification for the N-way problem it was trained on.

3.7.1 Measuring uncertainty in classification

The standard way to compute uncertainty in a softmax distribution is the entropy:

$$\mathbb{H} = -\sum_c \hat{p}_c \log \hat{p}_c. \tag{3.33}$$

The higher the entropy, the higher the uncertainty. Whereas in regression a maximum-likelihood model does not provide a measure of uncertainty in the output, for classification the output softmax is always available, so an uncertainty measure can be computed. However, in ML models this uncertainty is known to be unreliable, such as having low uncertainty in wrong predictions. By incorporating the output of several model samples in the predictive distribution, a Bayesian model can produce more reliable uncertainty estimates.

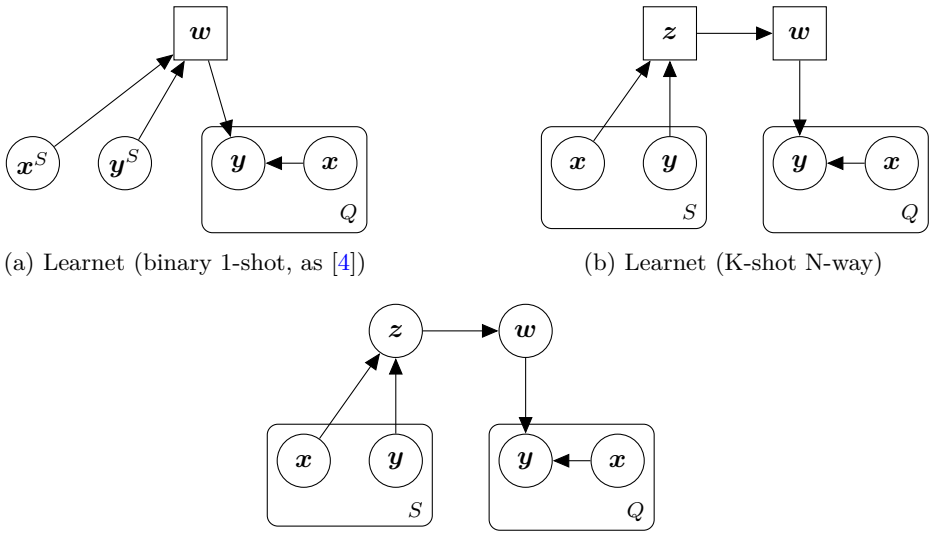
The entropy is not the only measure we can use for uncertainty. Particularly, for Bayesian models we can use the BALD estimator, which measures the mutual information between predictions and model posterior. This estimator, which takes into account not only the entropy the predictions, but also the variance in the predictions across model samples, can be approximated as [16]:

$$\begin{aligned}\mathbb{I} &= -\sum_c \left(\sum_{t=1}^T \hat{p}_c^t\right) \log \left(\sum_{t=1}^T \hat{p}_c^{(t)}\right) + \frac{1}{T}\sum_c \sum_{t=1}^T \hat{p}_c^{(t)} \log \hat{p}_c^{(t)} \\ &= \mathbb{H} + \frac{1}{T}\sum_c \sum_{t=1}^T \hat{p}_c^{(t)} \log \hat{p}_c^{(t)}\end{aligned} \tag{3.34}$$

3.8 Model summary and naming conventions

We now understand why a Learnet-like model fits particularly well to the hierarchical Bayesian formulation from (3.6). The representation of the support set is represented by the post-aggregation vector $\boldsymbol{z}(\mathcal{D}^S)$ (first stage), which is then passed to the hypernetwork to produce the dynamic weights $\boldsymbol{w}(\boldsymbol{z})$ (second stage), which will be used by the prediction network to produce predictions for the elements of the query set (third stage). A graphical model depiction of the original Learnet, our K-shot N-way extension, and Bayes-Learnet can be seen in Figure 3.1.

Since the original Learnet model [4] was used just in the context of binary one-shot learning, the task network receives a single support exemplar. This means that, in our naming convention, the hypernetwork and the task network are the same, as there is no clear difference between their functionality. In our formulation, however, $\mathcal{D}$ contains several datapoints, so task inference is a problem of its own, since information from the datapoints of the support set must be aggregated



(a) Learnet (binary 1-shot, as [4])

(b) Learnet (K-shot N-way)

(c) Bayes-Learnet

Figure 3.1: Graphical models of the original version of Learnet, our extension to K-shot N-way, and Bayes-Learnet.

in an order-invariant way. In other few-shot learning methods this distinction is not as clear. For example, in gradient-based method (e.g. MAML [12]) the support set representation is the obtained by updating all of the network's parameters by taking one gradient step to minimise the loss of the support set, so there is no explicit representation to, for example, compare support sets, whereas in metric or embedding learning (e.g. Prototypical Networks [58]) the representation is obtained by passing the support set through a feed-forward network to obtain a support set embedding, which will not change the representation of the elements of the query set.

Stripping all the KL costs and stochastic variables from (3.19) results in an of the original Learnet to K-shot N-way. Risking an abuse of notation, we will refer to this version simply as Learnet from here on, and to the full Bayesian model as Bayes-Learnet.

3.9 Evaluation metrics

Besides classification accuracy, we are interested in evaluating the Bayes-Learnet model in problems where the advantage of the Bayesian formulation over the standard Learnet might become clear, particularly in problems that involve reasoning about the model's uncertainty. To this end, we propose several evaluation tasks.

3.9.1 Junk task detection

One important aspect to measure in a few-shot learning system is the reliability of the support set provided. For example, when training on the Omniglot dataset [32], we want to be able to tell whether all the elements of the support set with a given label actually belong to the

same character class. This falls into the framework of outlier detection, which is a standard benchmark for deep Bayesian models.

Intuitively, the predictive uncertainty of the model for all elements in the query set should be higher the more junk there is in the support set, or the more incoherent it is. Formally, we compute the area under the curve (AUC) of the receiver operating characteristic (ROC) using the entropy of the outputs as the binary classifier. This gives us an estimate of how good the models are at distinguishing between good and bad support sets. The AUC-ROC is a reliable metric as long as we have as many positive and negative examples at test time, so we make sure this is the case during the experiments.

3.9.2 Query outlier detection

A related task is that of detecting a query element which does not belong to any of the classes provided in the support set. We can measure the model's performance in this task in exactly the same way as the junk task detection above, but now instead of providing good and bad support sets, we provide elements to the query set that do not belong to any of the (correct) support classes in an N-way problem. This is essentially a few-shot version of the standard outlier detection tasks used to benchmark Bayesian neural network methods (see e.g. [46]).

3.9.3 Active few-shot learning

Recent works [14, 19] have shown good performance of probabilistic meta-learning models on uncertainty estimation in 1-D regression, Bayesian optimisation and simple contextual bandits tasks. This stems from the ability of the models to learn the uncertainty in input space given a number of datapoints. However, the input space in this task is very low dimensional, and these examples were performed on regression tasks, where, as we have seen before, a straightforward uncertainty measure is not available in maximum-likelihood models. This leaves unanswered the question of how such models would perform in high-dimensional, classification problems, where uncertainty estimates are available for maximum-likelihood models. If our argument that Bayesian models provide better uncertainty estimates is correct, then a Bayesian model should perform better than an ML model at uncertainty-based tasks.

In order to test this we apply our model in the setting of active learning in K-shot N-way classification problems. The term active few-shot learning encompasses any problem where we need to sequentially decide which points to add to the support set in order to reduce the error in the query set as much as possible. It is clear that both Bayesian optimisation and contextual bandits fall under this term. The acquisition function used to determine which point to add to the support set is typically one that expresses the uncertainty of the model at each input $\boldsymbol{x}$. As before, we use the predictive entropy or BALD as our acquisition function.

We define our evaluation protocol as follows. Having a model trained for K-shot learning,

we start with a support set with $1 \times N$ elements, a tentative set with $(K-1) \times N$ elements and a query set with $K' \times N$ elements (in both sets the elements are equally distributed among the same N classes), at each step we compute the acquisition function for every element in the tentative set, given the current support set, and add the element of the tentative set that maximises the acquisition function to the support set. We continue until every element in tentative set has been added to the support set, and performance is evaluated on the query set at every step. This process is repeated for many support/query sets drawn from the meta-test set. This setting is similar to that used by Kim et al. [28].

Recent work on active few-shot learning include Woodward and Finn [67] and Boney and Ilin [6], though in the former the authors focus on learning a query policy through reinforcement learning (whereas we use an uncertainty-based policy), and in the latter on the integration with semi-supervised learning.

3.10 Improving sample efficiency

Bayesian methods are particularly suitable for training models where data is limited, since computing the posterior $p(\boldsymbol{w}|\mathcal{D})$ depends on a user-defined prior $p(\boldsymbol{w})$, the influence of which is larger for smaller training sets. In the limit of infinite data the prior's influence will vanish, and in the limit of no data, the posterior is the prior. Though priors play an important role on what type of functions are being modelled in classical models (i.e. not neural network-based) where their influence is well understood and there is a wide variety to choose from, weight priors in deep learning are usually restricted to L_2 or L_1 regularization, and they are not guaranteed to improve performance even for small data regimes. There are many reasons why this might be the case. Due to the high number of non-linear interactions between layers, it is hard to intuitively understand what the effect of shrinking any particular weight is on the output function, so it is hard to argue that the network is being restricted to producing a particular type of output functions when a prior is used (unlike what happens in linear regression where this is easy to analyse), and even if it is we cannot describe it.

For example, even the relatively small Omniglot dataset, which is a standard benchmark for few-shot learning, contains 1200 training character classes[1], which means that there are over 2^{13} class training combinations for 5-way learning (without even taking into account the fact that each class contains 20 different elements to choose from). However, practical applications might benefit from quick adaptation through meta-learning, such as imitation learning for robotics [9], acquiring annotated data for hundreds of tasks might be extremely time consuming. However, it is well known that neural network-based systems that are trained on small datasets tend to overfit. In this case, a system trained on a small number of tasks will meta-overfit, i.e. it will show poor performance on unseen tasks.

[1] As per the standard training/test splits.

Therefore, we are interested in making meta-learning systems more sample efficient and less prone to overfitting, so that they can learn from a smaller amounts of annotated tasks. In this work we explore whether the use of a variational Bayesian formulation of a few-shot learning method like Learnet improves sample efficiency, not at the level of datapoints, but at the level of tasks. That is, we want to improve the few-shot learning performance when we do not have access to tens of thousands of training tasks. This has been investigated by Kim et al. [28], who showed very good results when using a Bayesian formulation of MAML.

3.11 Related work

Though meta-learning has gained popularity in recent years, there are still very few works that attempt to combine meta-learning with Bayesian inference in neural networks.

Our work is most similar to the recently proposed Neural Processes (NP's) [19]. In NP's, the support set is also aggregated into a stochastic task variable $\boldsymbol{z}$, but this is then passed as input to a deterministic prediction network, whereas we use $\boldsymbol{z}$ to parametrise a Bayesian neural network through the variable $\boldsymbol{w}$. In fact, the graphical models of the Neural Processes is the same as ours, though we have the additional treatment of the prediction network's weights $\boldsymbol{w}$ as a stochastic variable that can be conditioned on $\boldsymbol{z}$, which allows us to interpret our model as conditional Bayesian neural network, in the Bayes-by-Backprop sense [5]. The neural processes architecture uses mean aggregation to pool the embeddings of the support set, while we try to improve aggregation by using more flexible product-of-experts and normalized product-of-experts. Crucially, in neural processes the authors focus on regression tasks (1-D regression, contextual bandits, and face prediction), while we tackle mainly classification tasks.

In terms of the experiments performed, our work bears more resemblance to PLATIPUS [14] where the authors explore uncertainty at the level of tasks by using ambiguous (but correct) support sets at meta-test time. We try to explore a greater breadth of settings by constructing outlier task detection problems and exploring the suitability of this type of Bayesian approach to few-shot outlier detection and active few-shot learning, while also verifying the benefit of a Bayes-Learnet over just Learnet.

Other related works include [20], which frames MAML as hierarchical Bayesian inference, and approximates the posterior using a Laplace approximation with K-FAC [40], though the predictive distribution is approximated by a point estimate. [28] builds on this work and implements a more flexible and efficient posterior approximation using Stein Variational Gradient Descent [38]. Like us, [28] tackle the problem of learning with reduced meta-training sets and active few-shot learning.

Chapter 4

Experiments

4.1 Toy dataset: sinusoidal regression

For sinusoidal data, we follow closely the training setup from MAML. We have a set of meta-training tasks, where each task is an amplitude-phase tuple, (a, ϕ). We use 500 meta-training tasks and 200 meta-testing tasks sampled uniformly from $a \in [0.1, 5]$ and $\phi \in [0, \pi]$. For each of these tasks, we train on K points and evaluate (for computing meta-gradients during meta-training or obtaining performance during meta-testing) on 20 points, all of which are uniformly sampled from the range $x \in [-5, 5]$. While the tasks are fixed at the beginning of training, the training and evaluation datapoints for each task are sampled again every time a task is seen. During meta-testing, we resample the datapoints 5 times for each task, in order to obtain lower-variance error estimates. Unlike the experiments reported in MAML, we use noisy datapoints with $\sigma = 0.3$, so each point is drawn according to $y \sim \mathcal{N}(a \cdot \sin(x + \phi), 0.3^2)$. As in MAML, we use a prediction network with 2 hidden layers with 40 units and Relu activation, for the task network we use 2 hidden layers with 100 units and tanh activation, and for the hypernetworks we use 1 hidden layer with 10 units and tanh activation.

In all the experiments reported in the remainder of this section, we used Adam as our optimiser, with learning rate 0.0005.

4.1.1 Hypernetworks design choice

Using hypernetworks gives us countless model design choices: how many layers should be produced by a hypernetwork? Should hypernetworks share the task representation $\boldsymbol{z}$? If we use a single hypernetwork, at which layer should we use it?

For the sinusoidal problem, since the prediction network is quite small, we use hypernetworks at every layer, with the input to each hypernetwork being the same task embedding $\boldsymbol{z}$. In later sections when we deal with larger datasets and networks, we will reduce the number of hypernetworks to match the design choices of Learnet [4].

4.1.2 Testing the task network: deterministic

In our experiments, the task network was the main bottleneck in terms of achieving good performance. Using the correct architecture to incorporate the information from the points in a training set was crucial to achieve acceptable results.

Here we compare the ability of a mean aggregation and product of experts to regress from data to ground-truth task parameters, which in this case are amplitude and phase. We use deterministic networks in all cases, and use non-noisy data, $y = a \cdot \sin(x + \phi)$. For this task we used a task network with an output layer with 2 units (one for amplitude, one for phase). The 2-dimensional output is then aggregated across all points, and we use a mean squared error loss. The input to each task network is the concatenated $[x, y]$ pair. Results are shown in Table 4.1.

	Training loss	Test loss
Mean aggregation	0.117 ± 0.008	0.141 ± 0.014
Product of experts	0.033 ± 0.002	0.042 ± 0.006

Table 4.1: MSE of mean aggregation vs product-of-experts for sinusoidal parameter regression.

As we can see, product-of-experts aggregation provides better results then a simple mean aggregation. This supports our hypothesis that giving equal weight to all points during aggregation might be suboptimal, and producing not only a mean but also a variance allows the network to assign different importance to different points, depending on how informative they are to the task. During experiments we found it important to use a task network with many hidden units (100, as we use), even when the input to each task network is just 2-dimensional. There was no additional benefit on using even larger number of units. We note that here we did not compare to the normalized product-of-experts since expression for the output mean is the same to the product-of-experts.

Despite the performance gains of product-of-experts over mean aggregation, we find it slightly worrying that even though the model is highly overparametrised and we used noiseless data, it was not possible to obtain 0 training error. One would expect that a network with enough capacity would trivially regress to 0 on such simple data, even accounting for the fact that the x points are drawn randomly every time. However, even letting the network train for a very long time did not result in overfitting. We also tried using an LSTM though this did not perform better than the above methods.

4.1.3 Testing the task network: stochastic

We now compare the mean aggregation (Mean), product-of-experts (PoE) and normalized product-of-experts (nPoE), in terms of their ability to estimate uncertainty. As we have mentioned, one characteristic of the product-of-experts is that the output variance always reduces with the number of experts. In our case, this means that even if we pass the same $[x, y]$ pair 5 times to the network, its uncertainty in the output parameters will decrease even though

the 4 extra points did not bring any extra information about the curve. However, this is not necessarily a bad thing, since one would also argue that is the data distribution is such that 5 points fall on exactly the same coordinates, whatever belief the network had with a single point should be concentrated. We will compare the methods here for the task network alone, and we evaluate their impacts on the predictive uncertainty in a larger problem on Section 4.2.

We train the models as before, but now we use datapoints drawn from the sinusoidal with noise $\epsilon \sim \mathcal{N}(0, 0.3^2)$, and we use a Gaussian negative loglikelihood loss from (3.24) and (3.25) instead of just MSE loss. As before we use $\boldsymbol{z} = [a, \phi]$ as the ground-truth parameters to regress.

To evaluate the results, we plot the output distribution $q(\boldsymbol{z}|\mathcal{D})$ for an increasing number of points in $\mathcal{D}$, for a mean aggregation, product-of-experts and nomralized product-of-experts model, as shown in Figure 4.1. We can see that while all models produce predictions with similar accuracy at the mean, there is some disparity in the variances estimated. Specifically, using product-of-experts model shows a steady reduction in output variance as more points as given, which is in line with what we had discussed in Section 3.4.2, whereas the mean aggregation and normalized product-of-experts seem to have a more well calibrated variance, producing small variances that can increase or decrease as more points are given. However, we note that, when only 1 point is given, the mean aggregation model estimates a small variance, which is incorrect since a single point provides no information about the amplitude of the underlying sinusoidal curve.

From these plots, it becomes particularly clear that the use of a diagonal covariance on the task posterior is a limiting factor, as the model is unable to capture the correlation between amplitude and phase for each point. In later sections, when $\boldsymbol{z}$ becomes a hidden variable, we will use a much higher dimensional $\boldsymbol{z}$ in order to alleviate this limitation.

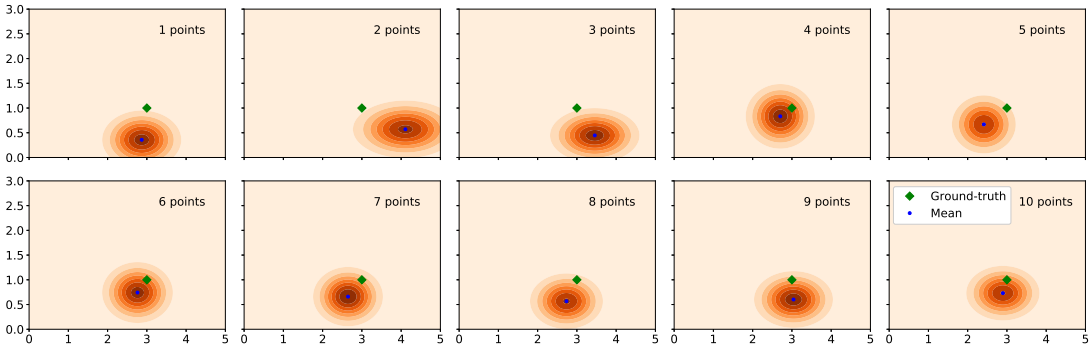
4.1.4 Testing the hypernetwork: deterministic

Having studied the ability of the task network to regress the ground-truth task parameters, we now perform the reverse experiment for the hypernetwork + prediction network combination: we will test the ability to regress the y values of a query set when the ground-truth task parameter is given as input to the hypernetwork. Here we use deterministic networks, i.e. $\boldsymbol{w}$ is not a stochastic variable.

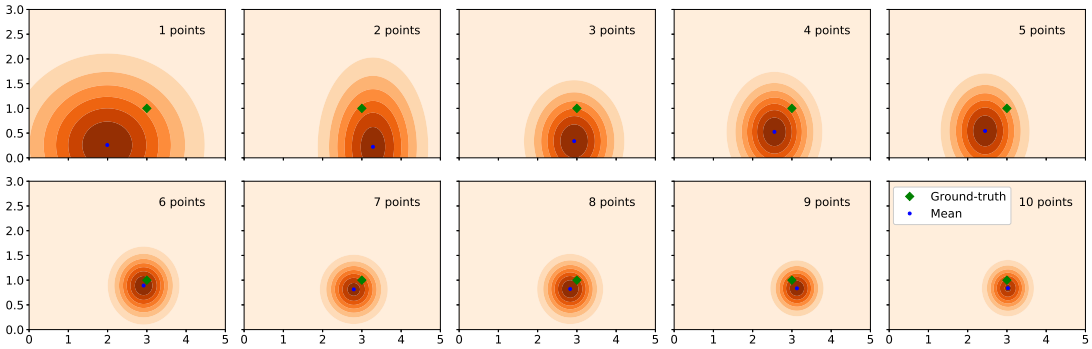
	Training loss	Test loss
Naive	0.002 ± 0.001	0.002 ± 0.001
Factored	0.041 ± 0.001	0.042 ± 0.001

Table 4.2: MSE of sinusoidal point prediction with ground-truth amplitude and phase given as input to the hypernetwork.

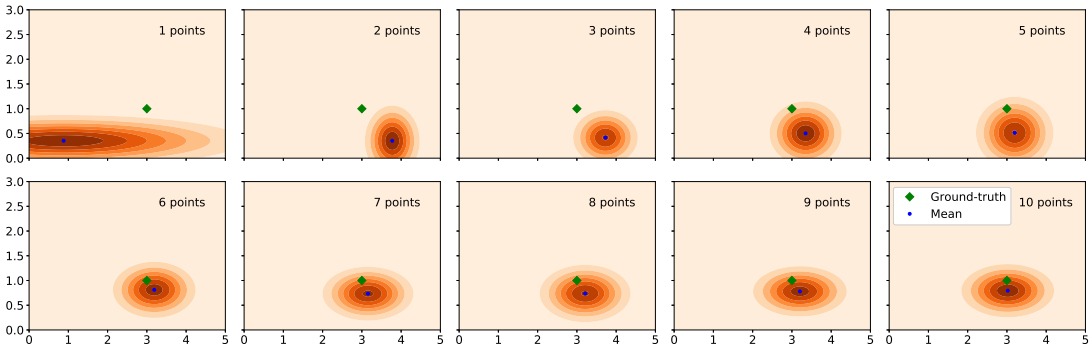
In Table 4.2 we can see that both the naive and factored hypernetworks are able to achieve essentially zero error (visually both achieve perfect fits) when predicting the query set's y



(a) Mean aggregation



(b) Product-of-experts



(c) Normalized product-of-experts

Figure 4.1: 2-D density map of task posterior distribution $q(\boldsymbol{z}|\mathcal{D})$ with increasing datapoints. Darker colour indicate higher density. We note that the colourmaps are normalized to each individual plot, so equal colours do not correspond to equal density values across plots.

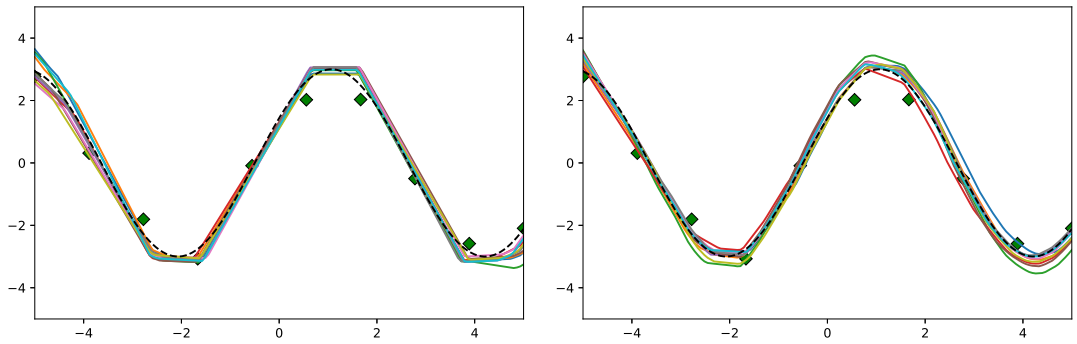
values using the input to the hypernetwork $\boldsymbol{z}$ as the ground-truth amplitude and phase that generated the curve. This is naturally unsurprising, but we ran this as a sanity check to make sure that a prediction network with a hypernetwork attached has enough training stability and representation power to regress to zero in a noiseless data setting.

4.1.5 Testing the hypernetwork: stochastic

We now study the behaviour of the hypernetwork + prediction network combination in the more interesting case when $\boldsymbol{w}$ is treated as a stochastic variable and the data is noisy. Initially we use

a fixed output residual $\sigma = 1$, so the likelihood term is again just a MSE loss, but since $\boldsymbol{w}$ is stochastic we are now optimising the variational objective in (3.20).

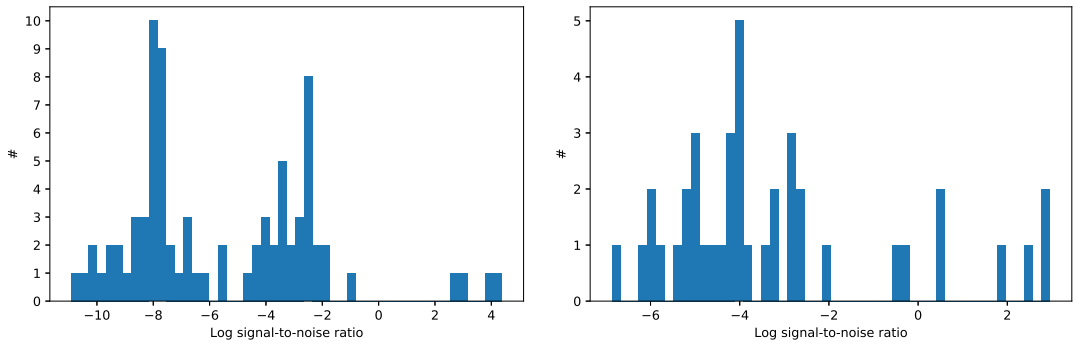
Training models with naive and factored weights, we can draw samples from the weight posterior $q(\boldsymbol{w}|\boldsymbol{z}_{\mathcal{D}})$ and plot the corresponding prediction curves, as shown in Figure 4.2.



(a) Naive (b) Factored

Figure 4.2: Samples drawn from the weight posterior, for naive and factored hypernetwork parametrisation. Green diamonds represent noisy point sampled from the sine and the black dashed curve represents the ground-truth sinusoidal.

While the factored version draws reasonable samples, we see that the curves drawn from the naive version are are piecewise linear, and not smooth like a sinusoid. While one could think that this is caused by the fact that we are using Relu activations in the prediction network, using tanh activations yields a similarly piecewise linear behaviour. In order to better understand this difference in behaviour between naive and factored weights, we plot the histogram of the log signal-to-noise ratio of the stochastic weights, $\log|\mu_{\boldsymbol{w}}/\sigma_{\boldsymbol{w}}|$, shown in Figure 4.3.



(a) Naive (b) Factored

Figure 4.3: Histograms of log signal-to-noise ratio of the stochastic weights of the network, for a fixed task, for naive and factored hypernetwork parametrisation.

We can see that though both parametrisations result in a large number of weights having a very low SNR, in the naive parametrisation only 6.5% of the weights have a log-SNR above -1, whereas in the factored parametrisation that number is 20%. This means that having a naive parametrisation is undesirable not only because of the scalability issues, but also because when

the weights are stochastic, there is too much noise in the weights in order to allow the model to learn appropriate (smooth) prediction functions. Since in the naive version all the weights are stochastic, a low SNR might mean that the network has to "factor-out" the weights with low SNR, which in the case of a Relu activation can be done by placing most of the activations in the saturating regime, which can lead to a piecewise linear function in the output since very few weights in the network actually contribute to the output function. This is alleviated in a factored parametrisation because of the presence of task-agnostic weights that can learn without noise.

Through this experiment, we conclude that the naive parametrisation is not suitable for use in our Bayes-Learnet, both in terms of scalability and probabilistic modeling. Therefore, in the remainder of this work we always use factored weights.

Using learnable residuals

The samples drawn in Figure 4.2b show that even when the ground-truth task parameters $\boldsymbol{z}$ are known, it makes sense to have stochastic $\boldsymbol{w}$ in order to obtain different plausible output functions. However, one could argue that this uncertainty could be captured by the model not by having a stochastic $\boldsymbol{w}$, but by learning the output residual σ in addition to the prediction y, in order to account for the output noise. Unfortunately, we found that in the full model it becomes hard to train the model with a Gaussian likelihood with learnable residual because it tends to get stuck in a local minimum of outputing a constant value of y and accounting for all the variation in the data by predicting a very large σ. For this reason, we choose to always use a fixed residual $\sigma = 1$ in regression experiments in this work. While this is the behaviour we observed for conditional Bayesian neural networks based on Bayes-by-Backprop, this does not seem to happen for non-conditional, Dropout MC Bayesian neural networks [27].

4.1.6 Testing the full model

We now finally explore the full model, trained according to (3.19), with both stochastic $\boldsymbol{w}$ and $\boldsymbol{z}$. Here we use a 10-dimensional $\boldsymbol{z}$, and as before we use $\beta = 0.1$ for training, with a learning rate of 0.0005, and the points are drawn with noise $\epsilon \sim \mathcal{N}(0, 0.3^2)$.

Training a 10-shot model and sampling curves when given 3, 5, 7 and 10 points we obtain the plots in Figure 4.4. The plots show that the samples become increasingly concentrated around the true curve as more points are given, with appropriate uncertainty estimates. It is particularly interesting to notice that when only 3 points are given most curves are around a "curve prior" with low amplitude, though one of the curve samples actually follows the true sinusoidal. This shows that the approximate posterior is able to explore a wide region of parameters, which cover both an uninformed curve and an plausible curve.

These curves can be improved by following the training protocol used by [19], where the

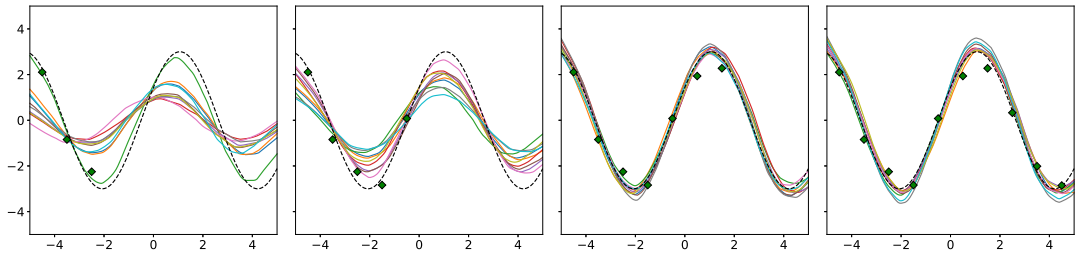


Figure 4.4: Posterior samples from Bayes-Learnet, using 3, 5, 7 and 10 support points. Model trained for $K = 10$. Samples not cherry picked.

model is trained using a random K between 2 and 10 at every draw. Plots are shown in Figure 4.5. We can see that now even for a small number of samples the model draws samples close to the true curve, while retaining the property that uncertainty reduces with the number of samples. This hints that our model would be suitable for the Bayesian optimisation and contextual bandit tasks benchmarked in Garnelo et al. [19], though we did not evaluate that in this work.

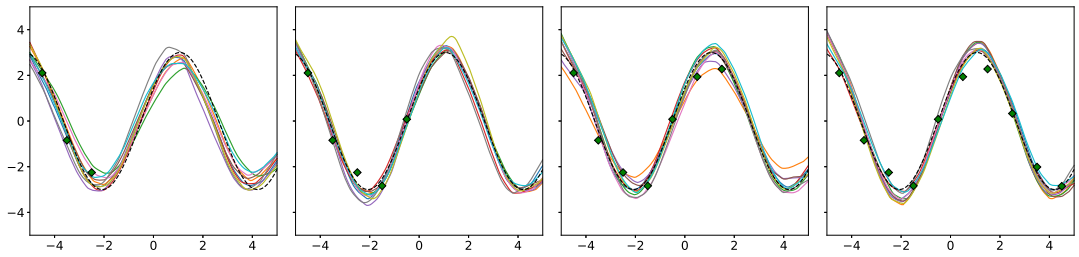


Figure 4.5: Posterior samples from Bayes-Learnet, using 3, 5, 7 and 10 support points. Model trained with varying $K = 2...10$. Samples not cherry picked.

These results are encouraging and show that a well tuned Bayes-Learnet is able to correctly estimate uncertainty and produce sensible samples from the posterior, even when the number of shots given at meta-test time is smaller than those given at meta-training time. We will explore the ability of our models to generalise to untrained K and N settings in later section.

Do we need stochasticity in the weights and in the task?

While the experiments performed above were done using stochastic $\boldsymbol{z}$ and $\boldsymbol{w}$, we observed that very similar results could be obtained when using stochasticity in only one of these variables. While this could hint to the fact that we do not actually need two stochastic variables in our model, experiments in the next section will show that there is a benefit to using stochastic $\boldsymbol{z}$ and $\boldsymbol{w}$ when it comes to uncertainty estimation in larger scale classification problems.

4.2 Omniglot

Having explored the properties of Bayes-Learnet and its individual components on the simple sinusoidal regression, we now run our model on more challenging Omniglot Lake et al. [32] dataset. Omniglot is a dataset of handwritten characters, containing 1623 character classes from 50 different alphabets, with each class containing 20 samples drawn by different people. We follow the convention proposed by [54], where 1200 classes plus its 90-degree multiple rotations are used for meta-training and the remaining 423 plus 90-degree multiple rotations are used for meta-testing. This yields a total of 4800 classes for meta-training and 1692 classes for meta-testing. We also follow the convention of downscaling the images from the original 105×105 resolution to 28×28.

This dataset will allow us to apply Bayes-Learnet to several classification problems, from character recognition to outlier detection, and compare it to Learnet in order to assess in which cases the Bayesian formulation is beneficial.

In all experiments that follow in this section we used a task network whose expert network has with 3 hidden layers with 64 channels, 3×3 filters, 2×2 max-pooling and Relu activation at every layer except the last where we use global mean-pooling and linear activation. After mean-pool, we either perform mean aggregation across experts followed by a fully connected layer to produce the mean and variance of $\boldsymbol{z}$, or produce the mean and variance of each expert using a shared fully connected layer followed by product-of-experts aggregation. In all experiments $\boldsymbol{z}$ is of dimension 100. The prediction network has the same architecture as each the expert networks, with the last layer having only 1 output channel and sigmoid activation after global mean-pooling. The penultimate layer of the prediction network has factored weights, and the hypernetwork has a single linear layer from $\boldsymbol{z}$ to $\boldsymbol{w}$. When results for Learnet and Bayes-Learnet are shown in tables, without further remarks, it is assumed that the normalized product-of-experts task network is being used. The results are typically reported for 5000 samples from the meta-test set using the same random seed (which was not searched over).

4.2.1 K-shot N-way performance

Following the choice reported by Bertinetto et al. [4], we use a hypernetwork at the penultimate layer in order to report the results for our K-shot N-way extension of Learnet. For the results reported in Table 4.3 we trained Learnet and Bayes-Learnet for 120000 meta-iterations with a learning rate of 0.0003, and $\beta = 0.0002$ for Bayes-Learnet. Normalized product-of-experts was used for the task network.

We can see that both our Learnet extension to K-shot N-way problems and Bayes-Learnet perform slightly under, though competitively with the state-of-the-art.

	1-shot, 5-way	5-shot, 5-way	1-shot, 20-way	5-shot, 20-way
MANN	82.8	94.9	—	—
Siamese Nets	97.3	98.4	88.1	97.0
Matching Nets	97.9	98.7	93.5	98.7
Neural Statistician	98.1	99.5	93.2	98.1
Meta Nets	99.0	—	97.0	—
Prototypical Nets	98.8	99.7	96.0	98.9
MAML	98.7 ± 0.4	99.9 ± 0.1	95.8 ± 0.3	98.9 ± 0.2
Relation Nets	99.6 ± 0.2	99.8 ± 0.1	97.6 ± 0.2	99.1 ± 0.1
Learnet	98.8	99.6	95.8	98.4
Bayes-Learnet	98.7	99.6	95.7	98.2

Table 4.3: K-shot N-way classification on Omniglot. Results sourced from [59].

4.2.2 Hyperparameter choice and difficulties during training

When training Bayes-Learnet on the Omniglot dataset there were several design choices that were critical to make the model work and train correctly. Firstly, it was necessary to choose a value of β small enough to prevent the likelihood term to be dominated by the latent KL terms. This is a known issue in variational autoencoders (see e.g. [7]), since in the beginning of training is it much easier for the network to minimise the KL cost, which might produce an initial latent space from which the rest of the network cannot learn. One alternative to this is to anneal β as training progresses (we tried this and we were able to reach larger values of β than when fixed), though we preferred to use a fixed but small enough β that would train stably while still providing the appropriate probabilistic properties to the model. While the benchmarks in Table 4.3 where obtained using $\beta = 0.0002$, most of the results below were obtained using $\beta = 0.001$, though the same conclusions would be reached with different values of β within a reasonable range.

One other issue was that when producing the mean and variance of $q(\boldsymbol{w}|\boldsymbol{z})$ as outputs of a neural network with split heads at the last layer, the latent cost on the variance term would tend to dominate the mean, preventing the hypernetwork from training at all. To fix this, we block the gradient of the variance term flowing through the layer shared with the mean. This allows the variance cost to control its own single layer, though the gradients computed through the hypernetwork and down to the task network only take the mean into account. In the task network $p(\boldsymbol{z}|\mathcal{D}^S)$, however, it was not necessary to block gradients.

Regarding the choice of choice of prior variance in $p(\boldsymbol{w})$, we observed that using small variances resulted in vanishing of the posterior mean, which renders the network untrainable. We instead used a prior variance of 1. This is somewhat at odds with the prior variances used by [5] which are orders of magnitude smaller. During early experiments, we noticed that while this type of very narrow prior on the weights might work for static weights, it puts too strong of a constrain on weights coming from a hypernetwork, which by itself already adds enough difficulty to the training of the full model. Using a large prior, provided β is small, allows us to put a more uninformative prior on the dynamic weights, which allows the weight means to vary

more across tasks.

4.2.3 Junk task detection

We now show the ability of several versions of our model in junk task detection. We are interested in assessing the impact of the probabilistic formulation *vs* deterministic Learnet in outlier detection, and comparing the use of the various aggregation methods. We also want to verify whether there is a need for both $\boldsymbol{w}$ and $\boldsymbol{z}$ being stochastic variables, or if we can remove stochasticity in $\boldsymbol{z}$.

To do this, we train the following models for 10-shot learning: a deterministic Learnet, a Bayes-Learnet with probabilistic $\boldsymbol{w}$ only (with normalized product-of-experts), a Bayes-Learnet with product-of-experts, and a Bayes-Learnet with normalized product-of-experts. Once trained, the models are given single-class support sets containing either 10 shots from the same character or 10 shots from random characters. We use a binary 10-shot setting, where the query sets contain 5 elements corresponding to the support set class (when not random) and 5 random elements. However, the particular choice of query set is not important, since the predictive entropy is what is going to be evaluated, not accuracy. The AUC-ROC of the predictive entropy is then calculated for 5000 correct and 5000 junk support sets. The results are shown in Table 4.4.

	10-shot AUC-ROC
Learnet	0.67
Bayes-Learnet (Mean; prob. $\boldsymbol{w}$ only)	0.77
Bayes-Learnet (PoE)	0.86
Bayes-Learnet (nPoE)	0.86
Bayes-Learnet (Mean)	0.88
Bayes-Learnet (Mean; 1 sample)	0.79

Table 4.4: AUC-ROC for binary 10-shot task outlier detection. For all Bayes-Learnet settings except the last one the results are obtained by computing the posterior predictive using 10 samples from the approximate posterior. Mean, PoE and nPoE correspond to mean aggregation, product-of-experts and normalized product-of-experts, respectively.

Firstly, we see that all the probabilistic versions beat the deterministic model, which shows that a Bayesian formulation of few-shot learning is indeed helpful to obtain correct uncertainty estimates at the level of tasks. Secondly, we see that using only stochastic $\boldsymbol{w}$ (= deterministic $\boldsymbol{z}$) reduces the model's ability to capture the uncertainty in the task, which motivates the use of a Bayesian model with two random variables instead of just one. Also, we see that all aggregation methods perform equally well, with mean aggregation having a slight advantage. Finally, measuring uncertainty using a single sample from the posterior (instead of 10 as in the other results) significantly reduces outlier detection ability, which is a strong indicator that the model is capturing the task uncertainty at the stochastic variables and not only at the softmax output, as we desired.

Though informative, the above results are reported just on a single metric. We now perform further experiments to better understand the underlying differences between models.

4.2.4 Finding the sources of uncertainty, testing on fewer shots, and other outlier metrics

One of the main arguments for using a normalized product-of-experts was that the basic product-of-experts has the undesirable property that the variance of the aggregation reduces with the number of experts. Though one might associate the variance of the task posterior $q(\boldsymbol{z}|\mathcal{D})$ with the uncertainty the model has about the task, we have to verify whether this is the case, or whether we should take the predictive entropy as the only reliable measure of uncertainty.

In order to test a wide range of behaviour, we use the same models as trained above, and test them in the following settings: 10-shot, 5-shot, 3-shot, 1-shot, 10-shot where all 10 elements of the support set are repeated, and 10-shot where all the elements of the support set are random (as in the previous experiment). For all these settings we report the uncertainty as measured by the predictive entropy, the standard deviation of the task posterior $q(\boldsymbol{z}|\mathcal{D})$ and the accuracy in binary classification (same setting as before, 5 elements of the query set correspond to the support set class and 5 others do not). The results for Learnet, Bayes-Learnet with mean aggregation, Bayes-Learnet with product-of-experts and Bayes-Learnet with normalized product-of-experts are shown in Table 4.5. We emphasize that the models were trained for 10-shot learning, and the setting of the first column is the only one seen during training.

		10-shot	5-shot	3-shot	1-shot	10-shot, repeat	10-shot, random
Learnet (nPoE)	PU	0.048	0.055	0.065	0.112	0.112	0.103
	TPS	0.299	0.301	0.303	0.311	0.311	0.266
	ACC (%)	98.4	97.9	97.4	91.0	91.0	50.0
Bayes-Learnet (Mean)	PU	0.146	0.154	0.163	0.211	0.210	0.380
	TPS	0.910	0.910	0.910	0.911	0.911	0.910
	ACC (%)	98.0	97.8	97.6	95.2	95.2	50.0
Bayes-Learnet (PoE)	PU	0.143	0.261	0.409	0.229	0.206	0.342
	TPS	0.91	1.285	1.660	2.881	0.911	0.907
	ACC (%)	98.1	97.4	90.5	50.6	95.4	49.9
Bayes-Learnet (nPoE)	PU	0.137	0.145	0.155	0.199	0.199	0.343
	TPS	0.916	0.916	0.916	0.916	0.916	0.915
	ACC (%)	98.0	97.8	97.6	95.0	95.1	49.8

Table 4.5: Performance of various methods on binary K-shot metrics, for models trained on 10-shot learning. See Figure 4.6 for a visualization of the tasks. PU - Predictive uncertainty; TPS - Task posterior standard deviation; ACC - Accuracy.

There are many conclusions to draw from this table:

- **Normalized vs unnormalized product-of-experts**: We see that the predictive uncertainty increases only slightly when provided fewer shots than those seen during training, and accuracy decrease is also minimal. Further, we notice that the task posterior standard

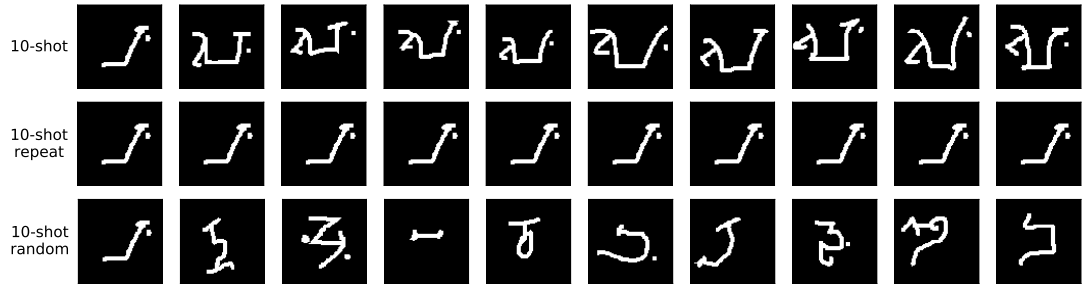


Figure 4.6: Character distribution example for 10-shot, 10-shot repeat and 10-shot random evaluation.

deviation is the same regardless of how many shots are given. This is a sign that in this normalized version all the experts tend to have similar average variance (across the dimensions of $\boldsymbol{z}$). Nonetheless, uncertainty in the 10-shot, random case is clearly higher than in all the other cases, which shows that model uncertainty does not need to be represented in the variance of the task posterior. Instead, the variations in mean and the relative contributions of the variances are enough to propagate through the hypernetwork $q(\boldsymbol{w}|\boldsymbol{z})$ and provide good uncertainty estimates when measured using the predictive entropy. On the other hand, the unnormalized model clearly incurs in large predictive uncertainty when given fewer shots than in training, even though these shots are correct (i.e. not junk). We can also see that the task posterior standard deviation is clearly tied to the number of shots (= number of experts). The main flaw of the unnormalized product-of-experts can be seen in the accuracy difference between when the model is given 1-shot or 10-shots of the same element. This means the representation learned by each expert is not reliable enough to provide good accuracy when a single expert is used at test time, and it shows the clear advantage of the normalized product-of-experts over the simple product-of-experts.

- **Mean aggregation vs normalized product-of-experts**: The simple mean aggregation produces identical results to the normalized product-of-experts, which indicates that the nPoE does not actually provide any significant modelling advantage over mean aggregation in a larger scale classification task.

- **Bayes-Learnet and Learnet**: The results show that even though accuracy of Learnet for 10-shot testing is slightly higher than Bayes-Learnet, accuracy decrease is larger with decreasing number of shots. We also see that predictive uncertainty increases faster with decreasing number of shots in Learnet, though it is the same for 10-shot task with repeated elements and random elements. Though one could argue that this shows a weakness of Learnet compared to Bayes-Learnet, it is not the only interpretation. If one considers an application where being given the input repeatedly is considered a faulty behaviour, a deterministic Learnet might be able to perform adequately at detecting such outlier task.

These results provide further insight into the advantages of Bayes-Learnet over Learnet, and

show that it performs well even when provided fewer shots than those used for training, with appropriate uncertainty estimates for these cases.

Dependence on the number of junk instances For completeness, in Figure 4.7 we show the change in accuracy and predictive entropy as we increase the fraction of the 10-shots that are random from 0, where all shots are from the same character, to 10, where all shots are from random characters (for example, 3 would correspond to 7 elements of the support set being drawn from the same character, and the remaining 3 are random). Clearly, the accuracies decrease very similarly, though the uncertainty in Bayes-Learnet increases faster.

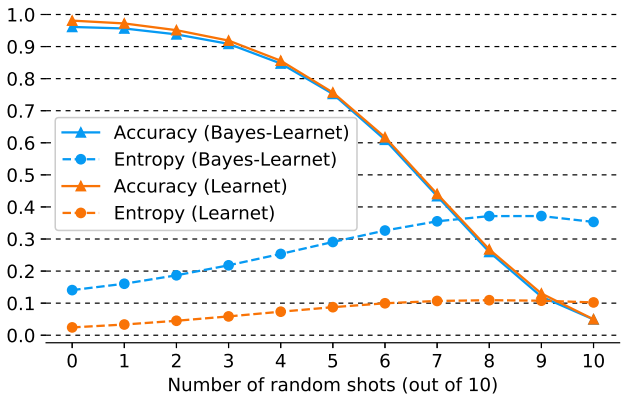


Figure 4.7: Accuracy and predictive entropy as a function of the fraction of random elements in the support set.

From this point on, reference to Bayes-Learnet always assumes use of normalized product-of-experts task network.

4.2.5 Query outlier detection

We now test the ability of Learnet and Bayes-Learnet to determine the existence of an element of the query set that does not belong to any of the classes of the (now correct) support set. Whereas in the previous section we were working in a binary (1-*vs*-all) 10-shot setting, we now work on a 1-shot 5-way setting.

The evaluation protocol is a follows. After training the models on 1-shot learning , we evaluate on 5000 meta-test sets, where each set is composed by a 1×5 support set, a 5×5 inlier query set (5 elements for each of the 5 classes of the support set) and a 5×5 outlier query set (5 elements belonging for each of 5 classes not in the support set). The inlier and outlier query sets are concatenated to form the query set, and we use the predictive entropy of each element of the query set, across meta-test sets, to obtain an AUC-ROC measure as before. The results are shown in Table 4.6, where we see that Learnet and Bayes-Learnet both perform very well, though comparably.

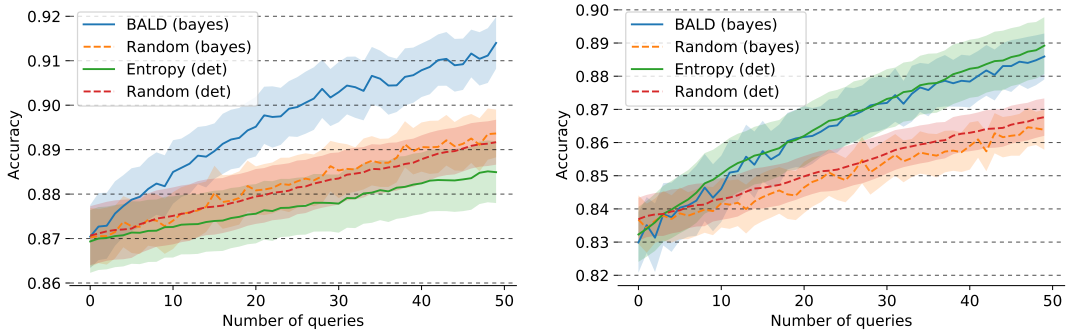
	1-shot 5-way AUC-ROC
Learnet	0.93
Bayes-Learnet	0.92

Table 4.6: Query outlier detection AUC-ROC for 1-shot 5-way classification.

We can think of a few reasons why the probabilistic model does not provide an advantage in these tasks. Due to the fact that the model is trained for within-class logistic regression, and not for N-way classification using a softmax (like [3, 12]), neither version is taught to explicitly solve a N-way task. This means that Learnet might be less prone to incorrectly classify an outlier character into one of the inlier classes by construction, although verifying such claim would require more in-depth comparisons, which we do not carry out here. We have also ran the same experiments but for a very reduced meta-train set (using only 100 character classes in total), though we found that the performance of Bayes-Learnet in that case was more severely affected than that of Learnet.

4.2.6 Active few-shot learning

For active few-shot learning we use the procedure described in Section 3.9.3, with $N = 100$, $K = 3$ and $K' = 5$. We use a 100-way setting in order to make the task more challenging and ambiguous. We perform 50 queries on the tentative set, and each experiment is ran for 100 meta-test datasets. For Bayes-Learnet we used the BALD acquisition function, but we observed the same results when using just the entropy. The results are shown in Figure 4.8.



(a) Model trained for 1-shot learning.

(b) Model trained for 5-shot learning.

Figure 4.8: Active learning on a 3-shot 100-way task. In the plots, "bayes" corresponds to Bayes-Learnet and "det" corresponds to Learnet. The acquisition functions used were BALD, entropy and random. Shaded regions indicate 95% confidence interval.

The results for a 5-shot trained model (Figure 4.8b) show that though predictive entropy for both Learnet and Bayes-Learnet perform better than random selection (in line with Figure 3, right, from [18]), there seems to be no advantage in using Bayesian predictive entropy in Bayes-Learnet versus just the output predictive entropy of Learnet. However, for a 1-shot trained model (Figure 4.8a), Learnet performs not better than random, whereas Bayes-Learnet

significantly outperforms it. In light of the experiments that are done in Section 4.2.8, it is not clear why this is the case[1], and we were not able to find a good explanation for this behaviour. One possibility is that in the 100-way setting, the results from Section 4.2.8 do not quite hold, and 1-shot Bayes-Learnet starts to show an advantage in that regime.

Though Kim et al. [28] show in Figure 2 c) of their work that a Bayesian formulation of MAML improves active learning over the deterministic model, we note that the two models they compare have different performances to start with (see Figure 2 a) of Kim et al. [28]), which makes it hard to disentangle the benefit of better uncertainty for active learning with the benefit of the overall model. They also do not compare the methods proposed to a random acquisition policy. In contrast, our models can be compared because they have, from the start, comparable performance on the K-shot N-way classification benchmarks.

4.2.7 Working with limited training tasks

In order to test whether Bayes-Learnet improves sample efficiency over Learnet, we train 1-shot 5-way model on 30, 50, 100, 500 and 4800 (all) character sets from the augmented Omniglot dataset (i.e. after rotations included), and tested on a meta-test set of 5000 tasks. The results can be seen in Table 4.7.

	1-shot 5-way accuracy				
Training set characters	30	50	100	500	All
Learnet	66.9	79.0	88.7	97.5	98.7
Bayes-Learnet	66.7	75.8	87.5	97.4	98.7
Bayes-Learnet (prob. $\boldsymbol{w}$ only)	70.8	81.9	87.6	—	—

Table 4.7: Test-set accuracy when training with a reduced number of training tasks. 'All' corresponds to 4800 characters sets.

The results show that Bayes-Learnet generalises slightly worse than Learnet when trained on limited data. However, we see that a version of Bayes-Learnet where only $\boldsymbol{w}$ is stochastic ($\boldsymbol{z}$ is deterministic) actually provides better generalisation with decreasing data. While we do not have conclusive explanations for why this is the case, one theory is that the KL cost on $\boldsymbol{z}$ when applied to small datasets causes task representations to not generalise correctly to unseen support sets. Removing stochasticity on $\boldsymbol{z}$, we are left with stochasticity on $\boldsymbol{w}$, which in this case behaves more as random noise on the weights of the prediction network, which might help prevent overfitting.

[1] We suspected this was from a computation error, but the results found on Section 4.2.8 were obtained using this exact same model.

4.2.8 Do we really need to train a model for each shot-way combination?

As a final experiment, we explore whether we can train a single model to perform well across all the K-shot N-way tasks. In general, few-shot learning models are trained for a particular K-shot N-way setting and tested for the same K and N. This means that, for examples, to perform the standard benchmarks on Omniglot, 4 models need to be trained. This seems undesirable, as in practice one would like to train a single model and have it transfer across number of shots and number of ways.

In the uncertainty estimation for regression tasks, as in Section 4.1.6, and as done by Garnelo et al. [19], this is overcome by passing a varying number of elements to the support set, so that the model learns to handle a variable K. For larger scale classification, models like MAML, which are trained with a softmax output, can only be tested for the same N they were trained on. This constraint is lifted in metric- or embedding-based models where a recognition score is computed for training, which is the case of Learnet. This allows the model to be tested for any N, for the particular K trained. However, we would like the model to generalise not only across N but also across K. While intuitively one could believe that models trained on large K will work well when tested on smaller K (such as what we see on Table 4.5), here we show that Learnet actually performs just as well on a larger K even when trained on just $K = 1$.

		Test task			
	Train task	1-shot 5-way	5-shot 5-way	1-shot 20-way	5-shot 20-way
Learnet	1-shot	*98.8*	99.6	*95.8*	98.5
	5-shot	98.3	*99.6*	94.3	*98.4*
Bayes-Learnet	1-shot	*98.7*	99.6	*95.8*	98.3
	5-shot	98.3	*99.6*	94.2	*98.2*

Table 4.8: Accuracy (%) for all the tasks, for a model trained for 1-shot or 5-shot learning. Numbers in italic correspond to the standard training and testing settings.

To show this, we train a Learnet and Bayes-Learnet for both $K = 1$ and $K = 5$, and test it on the 4 standard settings: 1-shot 5-way, 5-shot 5-way, 1-shot 20-way and 5-shot 20-way. The results are shown in Table 4.8. We see that a Learnet trained for 1-shot learning, performs better or equal to the 5-shot trained model, for all the test settings, including those with $K = 5$. The same behaviour is observed for Bayes-Learnet. These results show that one can train a single 1-shot Learnet model, and it will generalise correctly to any K and N at test time. However, since Learnet and Bayes-Learnet perform equally, we conclude the ability to generalise across ways and shots does not come from the Bayesian formulation, but is rather a characteristic of the base Learnet model itself. Li et al. [37] run analogous experiments but only for the sinusoidal data case.

Chapter 5

Conclusions

In this work we presented Bayes-Learnet, a variational Bayesian, K-shot N-way, extension of the Learnet model [4]. We showed that our model is able to correctly estimate uncertainty in toy problems like sinusoidal regression, active learning and task outlier detection on the Omniglot dataset. We also proposed new data aggregation methods, like the normalized product-of-experts.

Though the normalized product-of-experts aggregation worked better than mean aggregation for the sinusoidal dataset, in Omniglot problems we found them to perform equally well. One reason for this might be that when data and task embeddings are high-dimensional, the extra modelling that comes from the nPoE has little influence, since the mean aggregation has enough expressive power, and the nPoE can learn to default to mean aggregation by setting the covariance of all the experts to the same default value. However, the lack of advantage might also be related to the fact that we use only a diagonal covariance on the experts, which significantly reduces the ability of each expert to model covariances in its output. As we show breifly in the Appendix, the use a relation network to improve data aggregation shows promising results, and we believe that the problem of accurate and principled aggregation is worth investigating further.

We showed the advantage of Bayes-Learnet against Learnet for 1-shot trained models in a a 100-way classification setting. However, for query outlier detection those results were not verified, which begs the question of what is the fundamental difference between the two settings. We believe that the lack of advantage of Bayes-Learnet over Learnet in this task comes from the fact that there is not enough ambiguity in a 5-way task to make the uncertainty estimation of Bayes-Learnet shine. While the support sets a meta-test time contained any combination of characters from the meta-test set, a more ambiguous setting could be constructed by providing support sets containing only characters from the same alphabet.

Both outlier detection and active learning rely on the underlying ability of the model to reason about uncertainty in an area of input space where datapoints have not been provided, and we cannot decisively conclude that Bayes-Learnet has better uncertainty estimates for these

two tasks than Learnet. Investigating the reason for this, or how to overcome it, is a possible research direction.

One important point that is lacking from the experiments section is running the experiments done on the Omniglot dataset on the large-scale MiniImageNet dataset. This would provide a better means of comparison with competing models mostly in terms of K-shot N-way accuracy, which would indicate whether our model scales to more complex visual domains like natural images. We chose not to work with the MiniImageNet dataset because the goal of this work was not so much to obtain state-of-the-art classification performance against competing models, but rather to thoroughly explore the benefits and shortcomings of a Bayesian version of Learnet versus the deterministic model. Additionally, instead of testing our models on regression tasks that require uncertainty estimation like Bayesian optimisation or contextual bandits, we focused on classifications tasks, for which a strong baseline using the deterministic Learnet model is available, and which had been underexplored in previous work.

Another interesting avenue of future work is applying Bayes-Learnet for more interesting problems such as few-shot imitation learning, active discovery, or adapting the model to do few-shot reinforcement learning.

Bibliography

[1] Andrychowicz, M., Denil, M., Gómez Colmenarejo, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N., and Deepmind, G. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*.

[2] Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. (1992). On the Optimization of a Synaptic Learning Rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8.

[3] Bertinetto, L., Henriques, J. F., Torr, P. H. S., and Vedaldi, A. (2018). Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*.

[4] Bertinetto, L., Henriques, J. F., Valmadre, J., Torr, P. H. S., and Vedaldi, A. (2016). Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*.

[5] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning*.

[6] Boney, R. and Ilin, A. (2017). Semi-Supervised and Active Few-Shot Learning with Prototypical Networks. *arXiv preprint arXiv:1711.10856*.

[7] Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2015). Generating Sentences from a Continuous Space. *arXiv preprint arXiv:1511.06349*, pages 10–21.

[8] Daumé III, H. (2009). Bayesian Multitask Learning with Latent Hierarchies. In *Conference on Uncertainty in Artificial Intelligence*.

[9] Duan, Y., Andrychowicz, M., Stadie, B., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-Shot Imitation Learning. In *Advances in Neural Information Processing Systems*.

[10] Edwards, H. and Storkey, A. (2017). Towards a Neural Statistician. In *International Conference on Learning Representations*.

[11] Fei-Fei, L., Fergus, R., and Perona, P. (2003). A Bayesian Approach to Unsupervised One-Shot Learning of Object Categories. In *IEEE International Conference on Computer Vision*.

[12] Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*.

[13] Finn, C. and Levine, S. (2018). Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm. In *International Conference on Learning Representations*.

[14] Finn, C., Xu, K., and Levine, S. (2018). Probabilistic Model-Agnostic Meta-Learning. *arXiv preprint arXiv:1806.02817*.

[15] Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

[16] Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep Bayesian Active Learning with Image Data. In *International Conference on Machine Learning*.

[17] Garcia, V. and Bruna, J. (2018). Few-Shot Learning with Graph Neural Networks. In *International Conference on Learning Representations*.

[18] Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Whye Teh, Y., Rezende, D. J., and Eslami, S. M. A. (2018a). Conditional Neural Processes. In *International Conference on Machine Learning*.

[19] Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. (2018b). Neural Processes. *arXiv preprint arXiv:1807.01622*.

[20] Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *International Conference on Learning Representations*.

[21] Graves, A. (2011). Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems*.

[22] Ha, D., Dai, A., and Le, Q. V. (2017). HyperNetworks. In *International Conference on Learning Representations*.

[23] Hinton, G. E. and van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13, Santa Cruz, California, USA. ACM.

[24] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

[25] Hochreiter, S., Younger, S. A., and Conwell, P. R. (2001). Learning to Learn Using Gradient Descent. In *International Conference on Artificial Neural Networks*. Springer.

[26] Jordan, M. I., Jaakkola, T. S., and Saul, L. K. (1999). An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37:183–233.

[27] Kendall, A. and Gal, Y. (2017). What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems*.

[28] Kim, T., Yoon, J., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian Model-Agnostic Meta-Learning. *arXiv preprint arXiv:1806.03836*.

[29] Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[30] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

[31] Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese Neural Networks for One-shot Image Recognition. In *ICML Deep Learning Workshop*.

[32] Lake, B. M., Salakhutdinov, R., and Tnenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

[33] Lakshminarayanan, B., Pritzel, A., and Charles Blundell (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*.

[34] Lawrence, N. D. and Platt, J. C. (2004). Learning to Learn with the Informative Vector Machine. In *International Conference on Machine Learning*.

[35] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[36] Lee, Y. and Choi, S. (2018). Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace. In *International Conference on Machine Learning*.

[37] Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-SGD: Learning to Learn Quickly for Few-Shot Learning. *arXiv preprint arXiv:1707.09835*.

[38] Liu, Q. and Wang, D. (2016). Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In *Advances in Neural Information Processing Systems*.

[39] Louizos, C. and Welling, M. (2017). Multiplicative Normalizing Flows for Variational Bayesian Neural Networks. *arXiv preprint arXiv:1703.01961*.

[40] Martens, J. and Grosse, R. (2015). Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *International Conference on Machine Learning*.

[41] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations*.

[42] Munkhdalai, T. and Yu, H. (2017). Meta Networks. In *International Conference on Machine Learning*.

[43] Munkhdalai, T., Yuan, X., Mehri, S., and Trischler, A. (2018). Rapid Adaptation with Conditionally Shifted Neurons. In *International Conference on Machine Learning*.

[44] Naik, D. and Mammone, R. (1992). Meta-neural networks that learn by learning. In *International Joint Conference on Neural Networks*, volume 1, pages 437–442. IEEE.

[45] Nichol, A., Achiam, J., and Schulman, J. (2018). On First-Order Meta-Learning Algorithms. *arXiv preprint arXiv:1803.02999*.

[46] Pawlowski, N., Brock, A., Lee, M. C. H., Rajchl, M., and Glocker, B. (2017). Implicit Weight Uncertainty in Neural Networks. In *NIPS Workshop on Bayesian Deep Learning*.

[47] Ravi, S. and Larochelle, H. (2017). Optimization as a Model for Few-Shot Learning. In *International Conference on Learning Representations*.

[48] Reed, S., Chen, Y., Paine, T., Oord, A. v. d., Eslami, S. M. A., Rezende, D., Vinyals, O., and de Freitas, N. (2018). Few-shot Autoregressive Density Estimation: Towards Learning to Learn Distributions. In *International Conference on Learning Representations*.

[49] Rezende, D. J., Mohamed, S., Danihelka, I., Gregor, K., and Wierstra, D. (2016). One-Shot Generalization in Deep Generative Models. In *International Conference on Machine Learning*.

[50] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *International Conference on Machine Learning*.

[51] Ritter, H., Botev, A., and Barber, D. (2018). A Scalable Laplace Approximation for Neural Networks. In *International Conference on Learning Representations*.

[52] Robert, C. P. and Casella, G. (2004). *Monte Carlo Statistical Methods*. Springer New York.

[53] Salehinejad, H., Barfett, J., Valaee, S., and Dowdell, T. (2017). Training Neural Networks with Very Little Data - A Draft. Technical report.

[54] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-Learning with Memory-Augmented Neural Networks. *Journal of Machine Learning Research*.

[55] Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A Simple Neural Network Module for Relational Reasoning. In *Advances in Neural Information Processing Systems*.

[56] Schmidhuber, J. (1987). *Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook*. PhD thesis, Technische Universitat Munchen.

[57] Smith, L. B. and Slone, L. K. (2017). A Developmental Approach to Machine Learning? *Front. Psychol*, (8:2124).

[58] Snell, J., Twitter, K. S., and Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems*.

[59] Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H. S., and Hospedales, T. M. (2018). Learning to Compare: Relation Network for Few-Shot Learning. In *IEEE Conference on Computer Vision and Pattern Recognition*.

[60] Sung, F., Zhang, L., Xiang, T., Hospedales, T., and Yang, Y. (2017). Learning to Learn: Meta-Critic Networks for Sample Efficient Learning. *arXiv preprint arXiv:1706.09529*.

[61] Tenenbaum, J. B. (1999). *A Bayesian Framework for Concept Learning*. PhD thesis, Massachusetts Institute of Technology.

[62] Thrun, S. and Pratt, L. (1998). *Learning to learn*. Kluwer Academic Publishers.

[63] Vedantam, R., Fischer, I., Huang, J., and Murphy, K. (2018). Generative Models of Visually Grounded Imagination. In *International Conference on Learning Representations*.

[64] Vinyals, O., Deepmind, G., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching Networks for One Shot Learning. *arXiv preprint arXiv:1606.04080*.

[65] Wang, Y.-X. and Hebert, M. (2016). Learning to Learn: Model Regression Networks for Easy Small Sample Learning. In *European Conference on Computer Vision*.

[66] Welling, M. and Teh, Y. W. (2011). Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *International Conference on Machine Learning*.

[67] Woodward, M. and Finn, C. (2017). Active One-shot Learning. *arXiv preprint arXiv:1702.06559*.

Appendix

6.1 Proof of hierarchical ELBO

We now derive the evidence lower bound in (3.11). Starting with the KL-divergence between the approximate posterior and the true posterior, $\mathrm{KL}(q(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D})\|p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D}))$, we can expand it and use Bayes' rule, $p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z})/p(\mathcal{D})$ to obtain:

$$
\begin{aligned}
&\mathrm{KL}(q(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D})\|p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D})) \\
&= \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D})} \int q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D}) \log \left[\frac{q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})q(\boldsymbol{z}|\mathcal{D})}{p(\boldsymbol{w}, \boldsymbol{z}|\mathcal{D})} \right] d\boldsymbol{w} \\
&= \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D})} \int q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D}) \log \left[\frac{q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})q(\boldsymbol{z}|\mathcal{D})p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{z})p(\boldsymbol{z})} \right] d\boldsymbol{w} \\
&= \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D})} \left[\int q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D}) \log \frac{q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})}{p(\boldsymbol{w}|\boldsymbol{z})} - q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D}) \log p(\mathcal{D}|\boldsymbol{w}) d\boldsymbol{w} \right] + \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D})} \left[\log \frac{q(\boldsymbol{z}|\mathcal{D})}{p(\boldsymbol{z})} \right] + \log p(\mathcal{D}) \\
&= \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D})} \left[\mathrm{KL}(q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})\|p(\boldsymbol{w}|\boldsymbol{z})) - \mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})} \log p(\mathcal{D}|\boldsymbol{w}) d\boldsymbol{w} \right] + \mathrm{KL}(q(\boldsymbol{z}|\mathcal{D})\|p(\boldsymbol{z})) + \log p(\mathcal{D})
\end{aligned}
$$

Noting that the KL-divergence is a positive quantity, we can rewrite the last equation as:

$$
\log p(\mathcal{D}) \geq \mathbb{E}_{q(\boldsymbol{z}|\mathcal{D})} \left[\mathbb{E}_{q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})} \log p(\mathcal{D}|\boldsymbol{w}) - \mathrm{KL}(q(\boldsymbol{w}|\boldsymbol{z}, \mathcal{D})\|p(\boldsymbol{w}|\boldsymbol{z})) \right] - \mathrm{KL}(q(\boldsymbol{z}|\mathcal{D})\|p(\boldsymbol{z}))
$$

where the right-hand side corresponds to the lower bound on the evidence $\log p(\mathcal{D})$. ■

6.2 Relation network to improve data aggregation

Though the aggregation methods described in Section 3.4 provide good performance in the Omniglot tasks, the fact that on the sinusoidal problem a task network could not regress the support set to the ground-truth amplitude and phase with zero error was unsatisfying. One possible limiting factor is the fact that the use of a shared network that takes a single point $[\boldsymbol{x}, \boldsymbol{y}]$ as input might not allow the model to take the correlations of points into account. To solve this, we use a relation module [55] to produce all the pair of points from the support set. Each of these pairs $[\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{x}_j, \boldsymbol{y}_j]$, $i \neq j$, is then passed to a network (expert), where all these networks share weights. We run the same experiment as in Table 4.1, and the results are

shown in Table 6.1. For both mean aggregation and product-of-experts the loss has decreased significantly, which shows that capturing correlations between points is beneficial.

	Training loss	Test loss
Mean aggregation	0.018 ± 0.001	0.021 ± 0.002
Product-of-experts	0.0084 ± 0.0015	0.0064 ± 0.0006

Table 6.1: MSE of mean aggregation vs product-of-experts for sinusoidal parameter regression, using all datapoint pair combinations.

Though using datapoint pairs increases the ability of the model to use data correlations, this comes at the price of scalability since the number of pairs scales quadratically with the number of datapoints. These experiments were ran only briefly so we did not have the time to experiment with the method on the Omniglot dataset.