Probabilistic Meta-Representations Of Neural Networks

Theofanis Karaletsos[1], **Peter Dayan**[1,2],
Zoubin Ghahramani[1,3]
[1] Uber AI Labs, San Francisco, CA, USA
[2] University College London, United Kingdom
[3] Department of Engineering, University of Cambridge, United Kingdom
theofanis@uber.com, dayan@gatsby.ucl.ac.uk,
zoubin@eng.cam.ac.uk

Abstract

Existing Bayesian treatments of neural networks are typically characterized by weak prior and approximate posterior distributions according to which all the weights are drawn independently. Here, we consider a richer prior distribution in which units in the network are represented by latent variables, and the weights between units are drawn conditionally on the values of the collection of those variables. This allows rich correlations between related weights, and can be seen as realizing a function prior with a Bayesian complexity regularizer ensuring simple solutions. We illustrate the resulting meta-representations and representations, elucidating the power of this prior.

1 Introduction

We propose a higher-level abstraction of neural network weight representations in which the units of the network are probabilistically embedded into a shared, structured, meta-representational space (generalizing topographic location), with weights and biases being derived conditional on these embeddings. Rich structural patterns can thus be induced into the weight distributions. Our model captures uncertainty on three levels: meta-representational uncertainty, function uncertainty given the embeddings, and observation (or irreducible output) uncertainty. This hierarchical decomposition is flexible, and is broadly applicable to modeling task-appropriate weight priors, weight-correlations, and weight uncertainty. It can also be beneficially used in the context of various modern applications involving out of sample generalization, where the ability to perform structured weight manipulations online is beneficial [1] .

[1] Due to space concerns we provide more background and related work in the Appendix Sec. 8

2 Meta-Representations of Units

We suggest abandoning *direct* characterizations of weights or distributions over weights, in which weights are individually independently tunable. Instead, we couple weights using *meta-representations* (so called, since they determine the parameters of the underlying NN that themselves govern the input-output function represented by the NN). These treat the units as the primary objects of interest and embed them into a shared space, deriving weights as secondary structures.

Consider a code $\mathbf{z}_{l,u} \in \Re^D$ that uniquely describes each unit u (visible or hidden) in layer l in the network. Such codes could for example be one-hot codes or Euclidean embeddings of units in a real space $\Re^K$. A generalization is to use an inferred *latent representation* which embeds units l, u in a D-dimensional vector space. Note that this code encodes the unit itself, not its activation.

Weighs $w_{l,i,j}$ linking two units can then be recast in terms of those units' codes $\mathbf{z}_{l,i}$ and $\mathbf{z}_{l-1,j}$ for instance by concatenation $\mathbf{z}_w(l,i,j) = \left[\mathbf{z}_{l,i}, \mathbf{z}_{l-1,j}\right]$. We call the collection of all such weight codes $\mathcal{Z}_w$ (which can be deterministically derived from the collection of unit codes $\mathbf{Z}$). Biases $\mathbf{b}$ can be constructed similarly, for instance using 0's as the second code; thus we do not distinguish them below. Weight codes then form a conditional prior distribution $P(w_{l,i,j}|\mathbf{z}_w(l,i,j))$, parameterized by a function $g(\mathbf{z}_w(l,i,j),\xi)$ shared across the entire network. Function g, which may itself have parameters ξ, acts as a conditional hyperprior that gives rise to a prior over the weights of the original network:

$$P(\mathcal{W}|\mathcal{Z}_w;\xi) = \prod_{l=1}^{L}\prod_{i=1}^{V_l}\prod_{j=1}^{V_{l-1}} P(w_{l,i,j}|\mathbf{z}_w(l,i,j);\xi).$$

There remain various choices: we commonly use either Gaussian or implicit models for the weight prior and neural networks as hyperpriors (though clustering and Gaussian Processes merit exploration). Further, the weight

code can be augmented with a global state variable $\mathbf{z_s}$ (making $\mathbf{z}_w(l,i,j) = \left[\mathbf{z}_{l,i}, \mathbf{z}_{l-1,j}, \mathbf{z_s}\right]$) which can coordinate all the weights or add conditioning knowledge.

3 MetaPrior: A Generative Meta-Model of Neural Network Weights

The meta-representation can be used as a prior for Bayesian NNs. Consider a case with latent codes for units being sampled from $P(\mathbf{Z}) = \prod_{l,i} P(\mathbf{z}_{l,i})$, and the weights of the underlying NN being sampled according to the conditional distribution for the weights $P(\mathcal{W}|\mathcal{Z}_w)$. Put together this yields the model:

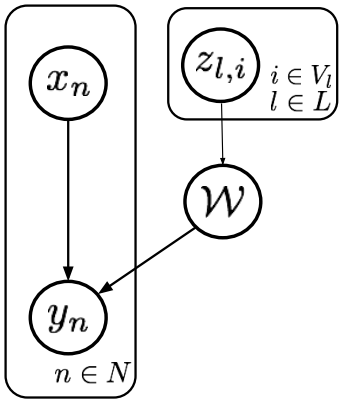

Figure 1: A graphical model showing the MetaPrior. Note the additional plate over the meta-variables $\mathbf{Z}$ indicating the distribution over meta-representations.

$$P(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{Z}} P(\mathbf{Z}) \int_{\mathcal{W}} P(\mathbf{y}|\mathbf{x},\mathcal{W}) P(\mathcal{W}|\mathbf{Z}) d\mathcal{W} d\mathbf{Z}$$
$$P(\mathbf{Z}) = \prod\nolimits_{l,i} P(\mathbf{z}_{l,i}) = \prod_{l,i} \mathcal{N}(\mathbf{0},\mathbf{1}).$$

Crucially, the conditional distribution over weights depends on more than one unit representation. This can be seen as a structural form of weight-sharing or a function prior and is made more explicit using the plate notation in Fig 1. Conditioned on a set of sampled variables $\mathbf{Z}$, our model defines a particular space of functions $f_{\mathbf{Z}} : \mathbf{X} \to \mathbf{Y}$. We can recast the predictive distribution given training data $\mathcal{D}^*$ as:

$$P(\mathbf{y}|\mathbf{x},\mathcal{D}^*) = \int_{\mathbf{Z}} Q(\mathbf{Z}|\mathcal{D}^*) \int_{f_{\mathbf{Z}}} P(\mathbf{y}|\mathbf{x},f_{\mathbf{Z}}) P(f_{\mathbf{Z}}|\mathbf{Z}) df_{\mathbf{Z}} d\mathbf{Z}.$$

Uncertainty about the unit embeddings affords the model the flexibility to represent diverse functions by coupling weight distributions with meta-variables.

The learning task for training MetaPriors for a particular dataset $\mathcal{D}$ consists of inferring the posterior distribution of the latent variables $P(\mathbf{Z}|\mathcal{D})$. Compared with the typical training loop in Bayesian Neural Networks, which involves learning a posterior distribution over weights, the posterior distribution that needs to be inferred for MetaPriors is the approximate distribution $Q(\mathbf{Z};\Phi)$ over the collection of unit variables as the model builds meta-models of neural networks. We train by maximizing the evidence lower bound (**ELBO**):

$$\log P(\mathcal{D};\xi) \geq \mathbb{E}_{Q(\mathbf{Z})}\Big[\log \frac{P(\mathcal{D}|\mathbf{Z};\xi)P(\mathbf{Z})}{Q(\mathbf{Z};\Phi)}\Big] \quad (1)$$

with $\log P(\mathcal{D}|\mathbf{Z}) \approx \frac{1}{S}\sum_{s=1}^{S} P(\mathbf{y}|\mathbf{x},\mathcal{W}^{s,m})$ and $\mathcal{W}^{s,m} \sim P(\mathcal{W}|\mathbf{Z}^m)$. In practice, we apply the reparametrization trick (15; 33; 39) and its variants (14) and subsample $\mathbf{Z}^m \sim Q(\mathbf{Z};\Phi)$ to maximize the objective:

$$\begin{aligned}\textbf{ELBO}(\Phi,\xi) =& \frac{1}{M}\sum_{m=1}^{M}\Big[\frac{1}{S}\sum_{s=1}^{S}\log P(\mathbf{y}|\mathbf{x},\mathcal{W}^{s,m})\Big] \\ &- \mathrm{KL}\big(Q(\mathbf{Z};\Phi)||P(\mathbf{Z})\big).\end{aligned} \quad (2)$$

However, pure posterior inference over Φ without gradient steps to update ξ results in learning meta-representations which best explain the data given the current hypernetwork with parameters ξ. We call the process of inferring representations $P(\mathbf{Z}|\mathcal{D})$ *illation*. Intuitively, inferring the meta-representations for a dataset induces a functional alignment to its input-output pairs and thus reduces variance in the marginal representations for a particular collection of data points.

We can also recast this as a two-stage learning procedure, similar to Expectation Maximization (EM), if we want to update the function parameters ξ and the meta-variables independently. First, we approximate $P(\mathbf{Z}|\mathcal{D};\xi)$ by $Q(\mathbf{Z};\phi)$ by maximizing $\textbf{ELBO}(\phi;\xi)$. Then, we can maximize of $\textbf{ELBO}(\xi;\phi)$ to update the hyperprior function. In practice, we find that Eq. 2 performs well with a small amount of samples for learning, but using EM can help reduce gradient variance in the small-data setting.

4 Experiments

We illustrate the properties of MetaPriors with a series of tasks of increasing complexity, starting with simple regression and classification, and then graduating to few-shot learning.

4.1 Toy Example: Regression

Here, we consider the toy regression task popularized in (10) ($y = x^3 + \epsilon$ with $\epsilon \sim \mathcal{N}(0,3)$). We use neural networks with a fixed observation noise given by the model, and seek to learn suitably uncertain functions. For all networks, we use 100 hidden units, and 2-dimensional latent codes and 32 hidden units in the hyper-prior network f.

In Fig. 2 we show two function fits to this example: our model and a mean field network. We observe that both

models increase uncertainty away from the data, as would be expected. We also illustrate function draws which show how each model uses its respective weight representation differently to encode functions and uncertainty. We sample functions in both cases by clamping latent variables to their posterior mean and allowing a single latent variable to be sampled from its prior. Our model, the MetaPrior-based Bayesian NN, learns a global form of function uncertainty and has global diversity over sampled functions for sampling just a single latent variable. It uses the composition of these functions through the meta-representational uncertainty to model the function space. This can be attributed to the strong complexity control enacted by the pull of the posterior fitting mechanism on the meta-variables. Maximum a posteriori fits of this model yielded just the mean line directly fitting the data. The mean field example shows dramatically less diversity in function samples, we were forced to sample a large amount of weights to elicit the diversity we got as single weight samples only induced small local changes. This suggests that the MetaPrior may be capturing interesting properties of the weights beyond what the mean field approximation does, as single variable perturbations have much more impact on the function space.

4.2 Toy Example: Classification

We illustrate the model's function fit to the half-moon two class classification task in Fig. 3, also visualizing the learned weight correlations by sampling from the representation. The model reaches 95.5% accuracy, on par with a mean field BNN and an MLP. Interestingly, meta-representations induce intra- and inter-layer correlations of weights, amounting to a form of soft weight-sharing with long-range correlations. This visualizes the mechanisms by which complex function draws as observed in Sec. 4.1 are feasible with only a single variable changing. The model captures structured weight correlations which enable global weight changes subject to a low-dimensional parametrization. This is a conceptual difference to networks with individually tunable weights.

4.3 MNIST50k-Classification

We use NNs with one hidden layer and 100 hidden units to test the simplest model possible for MNIST-classification. We train and compare the deterministic one-hot embeddings (Gaussian-OH), with the latent variable embeddings (Gaussian-LV) used elsewhere (with 10 latent dimensions); along with mean field NNs with a unit Gaussian prior (Gaussian-ML). We visualize the learned $\mathbf{Z}$s (Fig. 4) by producing a T-SNE embedding of their means which reveal relationships among the units in the network. The figure shows that the model infers semantic structure in the input units, as it compresses boundary units to a similar value. This is representationally efficient as no capacity is wasted on modeling empty units repeatedly.

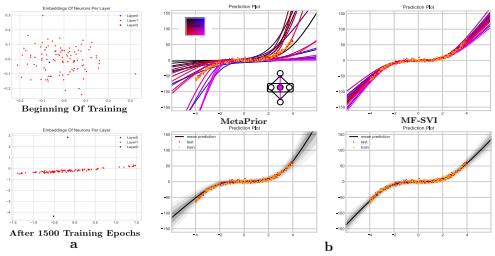

Figure 2: **MetaPrior as a Function Prior**: **(a)** Means of the meta-variables $\mathbf{Z}$ of MetaPrior model embedded in shared 2d space at the onset of training and at the end show how the variables perform a structured topographic mapping over the embedding space. **(b) Left**: MetaPrior fit and function draws $f_{\mathbf{Z}}$ are visualized. Functions are drawn by keeping all meta-variables clamped to the mean except a random one among the hidden layer units which is then perturbed according to the offset indicated by the color-legend. Changes in a single meta-variable induce global changes across the entire function. The function space itself is interesting as the model appears to have generalized to a variety of smoothly varying functions as we change the sampling offset, the mean of which is the cubic function. This is a natural task for our model, as all units of the hidden layer are embedded in the same space and sampling means exploring that space and all the functions this induces. **Right**: A **MF-VI** BNN with fixed observation noise function fit to the toy example and function draws are shown. The function draws are performed by picking 40 weights at random and sampling from their priors, while keeping the rest clamped to their posterior means. Single weight changes induce barely perceptible function fluctuations and in order to explore the representational ability of the model we sample multiple weights at once.

4.4 Few-Shot and Multi-Task Learning

The model allows task-specific latent variables $\mathbf{Z}_t$ to be used to learn representations for separate, but related, tasks, effectively *aligning* the model to the current input data. In particular, the hierarchical structure of the model facilitates few-shot learning on a new task t by inferring $P(\mathbf{Z}_t|\mathcal{D}_t)$, starting from the $\mathbf{Z}$ produced by the original training, and keeping fixed the mapping $P(\mathcal{W}|\mathbf{Z}_t;\xi)$. We tested this ability using the MNIST network. After learning the previous model for clean MNIST, we evaluate the model's performance on the MNIST test data in both a clean and a permuted version. We sample random permutations π for input pixels and classes and apply them to the test data, visualized in Fig. 5. This yields three datasets, with permutations of input, output or both. We

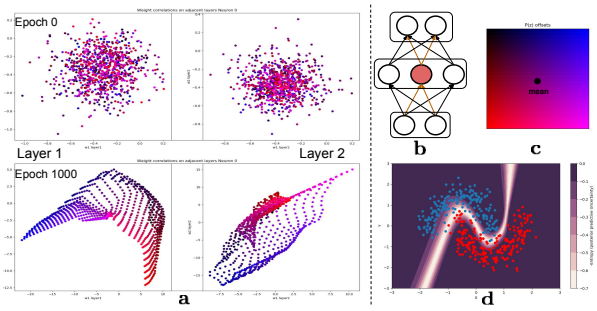

Figure 3: **Classification and Weight Correlation Example**: **(a)** We illustrate correlation structures in the marginal weight distributions for the Half Moon classification example. All units $\mathbf{Z}$ are clamped to their mean. We sample a single unit $\mathbf{z}_0$ repeatedly from its 2-dimensional distribution and sample $P(\mathcal{W}|\mathbf{Z})$. We track how particular ancestral samples generate weights by color-coding the resulting sampled weights by the offset of the ancestral sample from its mean. Similar colors indicate similar samples. Horizontal axes denote value for weight 1, vertical weight for 2, so that each point describes a state of a weight column in a layer. **Left**: We show the sampled weights from layer 1 of the neural network connected to unit 0. **Right**: The sampled weights from the output layer 2 connected to unit 0. **Top**: At the onset of training the effect of varying the unit meta-variable $\mathbf{z}_u$ is not observable. **Bottom**: After learning the function shown in SubFig. **d** we can see that the effect of varying the unit meta-variable induces correlations within each layer and also across both layers. **(b)** Sketch of the NN used with highlighted unit whose meta-variable we perturb and the affected weights. **(c)** Legend encoding shifts from the mean in the meta-variable to colors. **(d)** Predictive uncertainty of the classifier and the predicted labels of the datapoints, demonstrating that the model learns more than a point estimate for the class boundary.

then proceed to illate $P(\mathbf{Z}_t|\mathcal{D}_t)$ on progressively growing numbers of scrambled observations (shots) $\mathcal{D}_t$. For each such attempt, we reset the model's representations to the original ones from clean training. As a baseline, we train a mean field neural network. We also keep track of performance on the clean dataset as reported in Fig. 5. We examine a related setting of generalization in the Appendix Sec. 6.

5 Discussion

We proposed a meta-representation of neural networks. This is based on the idea of characterizing neurons in terms of pre-determined or learned latent variables, and using a shared hypernetwork to generate weights and biases from conditional distributions based on those variables. We used this meta-representation as a function prior, and showed its advantageous properties as a learned, adaptive, weight regularizer that can perform complexity control in function space. We also showed the complex correlation structures in the input and output weights of hidden units that arise from this meta-representation, and demonstrated how the combination of hypernetwork and network can adapt to out-of-task generalization settings and distribution shift by re-aligning the networks to the new data. Our model handles a variety of tasks without requiring task-dependent manually-imposed structure, as it benefits from the *blessing of abstraction* (9) which arises when rich structured representations emerge from hierarchical modeling.

We believe this type of modeling jointly capturing representational uncertainty, function uncertainty and observation uncertainty can be beneficially applied to many different neural network architectures and generalized further with more interestingly structured meta-representations.

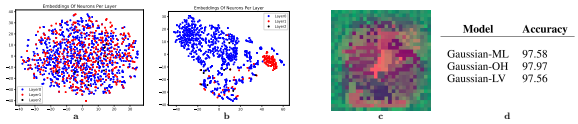

Figure 4: **MNIST Study**: **(a)** T-SNE visualization of the units in an MNIST network before training reveals random structure of the units of various layers. **(b)** Training induces structured embeddings of input, hidden and class units. **(c)** Input units color coded by coordinates of corresponding unit embeddings are visualized. Interestingly, many units in the input layer appear to cluster according to their marginal occupancy of digits. This is an expected property for the boundary pixels in MNIST which are always empty. The model represents those input pixels with similar latent units Z and effectively compresses the weight space. **(d)** Performance table of MNIST models.

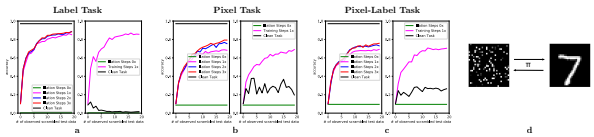

Figure 5: **Few-shot Disentangling Task (a)-(c) Left**: Performance of MetaPrior when inferring $Q(\mathbf{Z}|\mathcal{D})$ as a function of inference work (color coding) and revealed test data (shots divided by 20) for adaptation to a new task. **Right**: Performance of the baseline MF-BNN model decays on the clean task. **(d)** Illustration of the pixel permutation we stress the model with.

References

D. F. Andrews and C. L. Mallows. Scale mixtures of normal distributions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 99–102, 1974.

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

J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, 15(3):346–367, 2011.

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

P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

J. R. Finkel and C. D. Manning. Hierarchical bayesian domain adaptation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 602–610. Association for Computational Linguistics, 2009.

J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation*, 22(7):1860–1898, 2010.

N. D. Goodman, T. D. Ullman, and J. B. Tenenbaum. Learning a theory of causality. *Psychological review*, 118(1):110, 2011.

J. M. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

G. E. Hinton and D. Van Camp. 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. ACM, 1993.

A. Joshi, S. Ghosh, M. Betke, and H. Pfister. Hierarchical bayesian neural networks for personalized classification. In *Neural Information Processing Systems Workshop on Bayesian Deep Learning*, 2016.

D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

D. P. Kingma and M. Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, 2014.

D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

B. M. Lake, N. D. Lawrence, and J. B. Tenenbaum. The emergence of organizing structure in conceptual representation. *Cognitive science*, 2018.

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

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6405–6416, 2017.

N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems*, pages 329–336, 2004.

P. Liang, M. I. Jordan, and D. Klein. Learning programs: A hierarchical bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 639–646, 2010.

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

D. J. MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

D. J. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 354(1):73–80, 1995.

R. M. Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

G. Papamakarios, I. Murray, and T. Pavlakou. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2335–2344, 2017.

N. Pawlowski, M. Rajchl, and B. Glocker. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

Y. N. Perov and F. D. Wood. Learning probabilistic programs. *arXiv preprint arXiv:1407.2646*, 2014.

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

D. J. Rezende and S. Mohamed. Variational in-

ference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and variational inference in deep latent gaussian models. In *International Conference on Machine Learning*, volume 2, 2014.

S. Risi and K. O. Stanley. An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons. *Artificial life*, 18(4):331–363, 2012.

E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

M. Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

M. Titsias and N. D. Lawrence. Bayesian gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851, 2010.

M. K. Titsias and M. Lázaro-Gredilla. Local expectation gradients for black box variational inference. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 2638–2646. MIT Press, 2015.

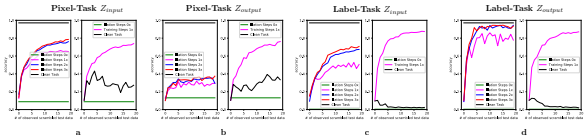

Figure 6: **Structured Surgery Task (a)-(d) Left**: Performance of MetaPrior when inferring $Q(\mathbf{Z}_\cdot|\mathcal{D})$ as a function of inference work (color coding) and revealed test data (shots divided by 20) for adaptation to a new task. **Right**: Performance of the baseline MF-BNN model decays on the clean task.

6 Appendix: Structured Surgery

In the multi-task experiments in Sec. 4.4, we studied the model's ability to update representations holistically and generalize to unseen variations in the training data. Here, we manipulate the meta-variables in a structured and targeted way (see Fig. 6). Since $\mathbf{Z} = \{\mathbf{Z}_{input}, \mathbf{Z}_{hidden}, \mathbf{Z}_{output}\}$, we can elect to perform illation only on a subset of variables. Instead of learning a new set of task-dependent $\mathbf{Z}_t$ variables, we only update input or output variables per task to demonstrate how the model can disentangle the representations it models and can generalize in highly structured fashion. When updating only the input variables, the model reasons about pixel transformations, as it can move the representation of each input pixel around its latent feature space. The model appears to solve for an input permutation by searching in representation space for a program approximating $\tilde{\mathbf{Z}}_{input} = \pi(\mathbf{Z}_{input})$. This search is ambiguous, given little data and the sparsity underlying MNIST. This process demonstrates the alignment the model automatically performs only of its inputs in order to change the weight distribution it applies to datasets, while keeping the rest of the features intact. Similarly, we observe that updating only the class-label units leads to rapid and effective generalization for class shifts or, in this case, class permutation, since only 10 variables need to be updated. The model could also easily generalize to new subclasses smoothly existing between current classes. These demonstrate the ability of the model to react in differentiated ways to shifts, either by adapting to changes in input-geometry, target semantics or actual features in a targeted way while keeping the rest of the representations constant.

7 Appendix: Probabilistic Neural Networks

Let $\mathcal{D}$ be a dataset of n tuples $\{(x_1, y_1), ..., (x_n, y_n)\}$ where x are inputs and y are targets for supervised learning. Take a neural network (NN) with L layers, V_l units in layer l (we drop l where this is clear), an overall collection of weights and biases $\mathcal{W} = \{\mathbf{W}_l\}_{1:L}$, and fixed nonlinear activation functions. In Bayesian terms, the NN realizes the likelihood $p(y|x, \mathcal{W})$; together with a prior $P(\mathcal{W})$ over $\mathcal{W}$, this generates the conditional distribution (also known as the marginal likelihood) $P(\mathbf{y}|\mathbf{x}) = \int P(\mathbf{y}|\mathbf{x}, \mathcal{W})P(\mathcal{W})d\mathcal{W}$, where $\mathbf{x}$ denotes $(x_1, \ldots, x_n)$ and $\mathbf{y}$ denotes $(y_1, \ldots, y_n)$. One common assumption for the prior is that the weights are drawn iid from a zero-mean common-variance normal distribution, leading to a prior which factorizes across layers and units:

$$P(\mathcal{W}) = \prod_{l=1}^{L}\prod_{i=1}^{V_l}\prod_{j=1}^{V_{l-1}} P(w_{l,i,j}) = \prod_{l=1}^{L}\prod_{i=1}^{V_l}\prod_{j=1}^{V_{l-1}} \mathcal{N}(w_{l,i,j}|0, \lambda),$$

where i and j index units in adjacent layers of the network, and λ is the prior weight variance. Maximum-a-posteriori inference with this prior famously results in an objective identical to L_2 weight regularization with regularization constant $1/\lambda$. Other suggestions for priors have been made, such as (26)'s proposal that the precision of the prior should be scaled according to the number of hidden units in a layer, yielding a contribution to the prior of $\mathcal{N}(0, \frac{1}{V_l})$.

Bayesian learning requires us to infer the posterior distribution of the weights given the data $\mathcal{D}$: $P(\mathcal{W}|\mathcal{D}) = P(\mathcal{W})\prod_{i=1}^{n} P(y_i|x_i, \mathcal{W})/P(\mathbf{y}|\mathbf{x})$. Unfortunately, the marginal likelihood and posterior are intractable as they involve integrating over a high-dimensional space defined by the weight priors. A common step is therefore to perform approximate inference, for instance by varying the parameters Φ of an approximating distribution $Q(\mathcal{W}; \Phi)$ to make it close to the true posterior. For instance, in Mean Field Variational Inference (**MF-VI**), we consider the factorized posterior:

$$Q(\mathcal{W}; \Phi) = \prod_{l=1}^{L}\prod_{i=1}^{V_l}\prod_{j=1}^{V_{l-1}} Q(w_{l,i,j}; \boldsymbol{\phi}_{l,i,j}).$$

Commonly, Q is Gaussian for each weight, $Q(w_{l,i,j}; \boldsymbol{\phi}_{l,i,j}) = \mathcal{N}(\mathbf{w}_{l,i,j}|\mu_{l,i,j}, \sigma^2_{l,i,j})$, with variational parameters $\boldsymbol{\phi}_{l,i,j} = \{\mu_{l,i,j}, \sigma^2_{l,i,j}\}$. The parameters Φ are adjusted to maximize a lower bound $\mathcal{L}(\Phi)$ to the marginal likelihood given by the Evidence Lower Bound (**ELBO**) (a relative of the free energy):

$$\mathcal{L}(\Phi) = \mathbb{E}_{Q(\mathcal{W})}\big[\log P(\mathbf{y}|\mathbf{x}, \mathcal{W}) + \log P(\mathcal{W}) - \log Q(\mathcal{W})\big]. \quad (3)$$

The mean field factorization assumption renders maximizing the **ELBO** tractable for many models. The

predictive distribution of a Bayesian Neural Network can be approximated utilizing a mixture distribution $P(y|x, \mathcal{D}) \approx \frac{1}{S} \sum_{s=1}^{S} P(y|x, \mathcal{W}^s)$ over S sampled instances of weights $\mathcal{W}^s \sim Q(\mathcal{W})$.

8 Appendix: Related Work

Our work brings together two themes in the literature. One is the probabilistic interpretation of weights and activations of neural networks, which has been a common approach to regularization and complexity control (2; 5; 10; 11; 12; 23; 24; 25; 27). The second theme is to consider the structure and weights of neural networks as arising from embeddings in other spaces. This idea has been explored in evolutionary computation (3; 8; 34; 36) and beyond, and applied to recurrent and convolutional NNs and more. Our learned hierarchical probabilistic representation of units, which we call a meta-representation because of the way it generates the weights, is inspired by this work. It can thus also be considered as a richly structured hierarchical Bayesian Neural network (7; 13). In important recent work training ensembles of neural networks (19) was proposed. This captures uncertainty well; but ensembles are a departure from a single, self-contained model.

Our work is most closely related to two sets of recent studies. One considers reweighting activation patterns to improve posterior inference (16; 22; 29). The use of parametric weights and normalizing flows (6; 28; 32) to model scalar changes to those weights offers a probabilistic patina around forms of batch normalization. However, our work is not aimed at capturing posterior uncertainty for given weight priors, but rather as a novel weight prior in its own right. Our method is also representationally more flexible, as it provides embeddings for the weights as a whole.

Equally, our meta-representations of units is reminiscent of the inducing points that are used to simplify Gaussian Process (GP) inference (31; 35; 37), and that are key components in GP latent variable models (20; 38). Rather like inducing points, our units control the modeled function and regularize its complexity. However, unlike inducing points, the latent variables we use do not even occupy the same space as the input data, and so offer the blessing of extra abstraction. The meta-representational aspects of the model can be related to Deep GPs, as proposed by (4).

Finally, as is clearest in the permuted MNIST example, the hypernetwork can be cast as an interpreter, turning one representation of a program (the unit embeddings) into a realized method for mapping inputs to outputs (the NN). Thus, our method can be seen in terms of program induction, a field of recent interest in various fields, including concept learning (17; 18; 21; 30).