PonderBayes: Uncertainty-Informed Pondering

Giulio Starace
13010840

Matteo Rosati
13858149

Victor Kyriacou
13438700

Jille van der Togt
12176729

Abstract

By default, neural networks do not adapt their computational power to the complexity of the task at hand. To overcome this Banino et al. [1] proposed PonderNet, a new algorithm that learns to adapt the amount of computation based on the complexity of the problem. However, PonderNet is still limited to pointwise predictions, with no notion of uncertainty which may be valuable for pondering decisions. We explore approaches for a Bayesian treatment of PonderNet to quantify and exploit model uncertainty in pondering decisions. We implement four ensemble-based variants, one of which achieves higher accuracy and faster convergence time on varying difficulty levels of the same task. We also consider a more complete Bayesian treatment of the problem but ultimately run into variance and numerical stability issues. We hope future work can address our shortcomings and improve this area further.

1 Introduction

Machine learning system design almost always involves calibrating the complexity of a model with the complexity of a task. This is typically addressed with hyperparameter depth decisions for linear networks and sequence length in Recurrent Neural Networks (RNNs). This is however sensitive to the biases of human researchers and may lead to sub-optimal configurations. PonderNet [1] attempts to endow neural networks with the ability (referred to as "pondering") to adjust their computational resource use based on the complexity of the task at hand. It improves on the previously introduced concept of Adaptive Computational Time (ACT) [12] by providing a method to backpropagate through all computational steps with unbiased gradients, as well as providing more stability.

While PonderNet provides a promising step in the direction of more general artificial intelligence, its performance in tasks such as identifying conditional dependencies and dealing with uncertainty is still limited in many of the same ways as conventional Deep Learning (DL) models. We hypothesize that uncertainty may provide useful information in the context of pondering. Bayesian Deep Learning (BDL) has recently emerged as a paradigm to address this issue [28, 5, 25, 9] by using Bayes' Theorem [2] to infer a posterior distribution over the model parameters given the observed data. Among others, this allows quantifying a model's uncertainty over its weights and ultimately over its outputs.

In this work, we explore avenues for a Bayesian treatment of PonderNet. We reason that a model that ponders should have some notion of its uncertainty to determine how long it should expect to ponder given its current state. With this premise, we deliver the following contributions:

1. We reproduce the findings of the original PonderNet paper, providing code for the model (not released by the authors) and the rest of our work[1].

[1] https://github.com/thesofakillers/ponder-bayes

Preprint. Under review.

2. We verify PonderNet's robustness by applying it to a new benchmark, MNIST [8].
3. We propose four "ensemble" PonderNet variants for uncertainty quantification and exploitation, described in Section 3.2.2. Our lambdaGT variant outperforms PonderNet.
4. We explore the feasibility of a more complete Bayesian treatment of the problem through Stochastic Variational Inference (SVI) as described in section 3.2.3.

2 Related work

2.1 Dynamic Neural Networks

PonderNet and ACT are part of the broader field of Dynamic Neural Networks (DynNNs). DynNNs distinguish themselves from more conventional static Neural Networks (NNs) in that they can adapt their parameters and/or architecture during inference time based on the incoming inputs. This leads to improvements in computational efficiency, as the networks can dynamically allocate model components such as layers [15], channels [23] or entire subnetworks [31]. DynNNs have also demonstrated improvements in data efficiency in the context of few-shot learning [3, 34]. Adaptive Early Exit Networks [6] modify the forward pass of an existing network. More classical approaches involve Spiking Neural Networks (SNNs) [26, 17] which perform data-dependent inference by propagating pulse signals. For a more comprehensive overview of the field, readers are directed to the excellent survey by Han et al. [13].

2.2 Bayesian Deep Learning

Like DynNNs, the field of BDL has emerged to address the limitations of more conventional NNs. While DL has achieved tremendous advancements in *perception* tasks, Probabilistic Graphical Models (PGM) have instead been excelling in probabilistic or causal *inference* and at dealing with uncertainty. BDL provides a principled probabilistic framework for unifying PGM and DL to reap the benefits from both fields. Examples include Bayesian interpretations of DL techniques such as dropout [10] and ensembling [22]. Initial research mostly focused on addressing the issue of the nontrivial time complexity of BDL methods [27, 29]. More recent advancements such as expectation propagation [14] and the reparametrization trick [5, 20] have addressed scalability issues leading to more widespread practical adoption of BDL methods. For a more complete overview, we direct readers to the surveys by Wang and Yeung [33] and Fortuin [9].

While the notion of "dynamic" Bayesian networks exists in the form of stochastic processes and other works [30], here "dynamic" refers to temporal or sequential. To our knowledge, we are the first to consider a Bayesian treatment of a "dynamic" neural network in terms of computation adaptation.

3 Experimental Setup

To evaluate the capacity to adapt computation to more difficult tasks, Banino et al. [1] distinguish between *interpolation* and *extrapolation* tasks. In the former, the training and evaluation data share the same difficulty. In the latter, the evaluation data is in some way more difficult. We apply this scheme to two datasets, which we describe in Section 3.1, comparing PonderNet to our contributed models, which we describe in Section 3.2.

3.1 Datasets and Evaluation

To test robustness, we apply PonderNet (but none of our variants) to MNIST [8], a dataset that the original authors did not consider. MNIST consists of 28x28 grayscale images of handwritten digits, with 60,000 training and 10,000 testing samples. The task is to classify a given image as one of the digits between 0 and 9. For interpolation, we leave the inputs untouched for both training and testing. For extrapolation, we randomly rotate the testing set images by degrees in the range $[-112, 112]$.

To aid with comparison, we apply all our models to the parity task used by Banino et al. [1], originally introduced by Graves [12]. Here the data consists of P-dimensional vectors of which a random number of elements $Q \geq 1$ is set to 1 or -1, and the remaining elements are set to 0. The task is to output 1 if there is an odd number of ones (including negatives) and 0 if there is an even number. In the

interpolation setting, $Q \in \{1, \ldots, P\}$ for both training and evaluation samples. In the *extrapolation* setting, $Q \in \{1, \ldots, P/2\}$ during training and $Q \in \{P/2+1, \ldots, P\}$ during evaluation. For our work, we set $P = 16$ for interpolation and $P = 24$ for extrapolation[2]. For both interpolation and extrapolation, we generate 128,000 training samples and two pairs of 25,600 samples, one for validation and one for testing, ensuring that the difficulty (number of non-zero elements) is uniformly distributed in our dataset. In both cases, we evaluate the halting step (explained in Section 3.2.1), prediction accuracy, and training time metrics.

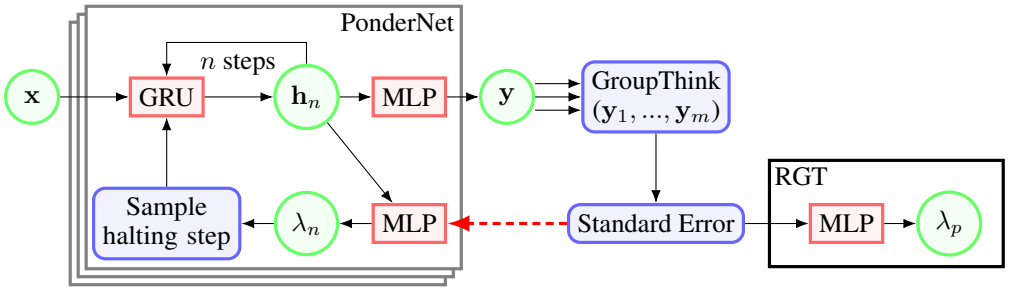

Figure 1: A graphical representation of the implemented models on the parity task modifying the original PonderNet model. The red dashed arrow represents the lambdaGT model, i.e., the standard error from GroupThink being used for the input to the λ_n layer. The additional architecture implemented in RGT and aRGT can be found in the black rectangle.

3.2 Models

The following section describes PonderNet, which our work is based on, and our extensions.

3.2.1 PonderNet

PonderNet [1] considers a supervised setting where the goal is to learn a function $f : x \to y$ from data $(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} = \{x^{(1)}, \ldots, x^{(k)}\}$ and $\mathbf{y} = \{y^{(1)}, \ldots, y^{(k)}\}$. The authors propose a new architecture that adjusts the forward pass and is trained using a custom loss function. The architecture requires a *step function* s of the form $\hat{y}_n, h_{n+1}, \lambda_n = s(x, h_n)$ and an initial state h_0, where $\hat{y}_n$ and λ_n are the network outputs and scalar probability of "halting" at step n.

PonderNet uses λ_n to learn the optimal value for n and uses a Bernoulli random variable Λ_n in order to represent a "continue" ($\Lambda_n = 0$) state and a terminal "halt" ($\Lambda_n = 1$) state. The decision process starts at a "continue" state with the transition probability

$$P(\Lambda_n = 1 | \Lambda_{n-1} = 0) = \lambda_n \quad \forall\, 1 \leq n \leq N. \tag{1}$$

In words, this is the conditional probability of entering a "halt" state at step n given that there has been no previous halting.

The prediction $\hat{y} \sim \hat{Y}$ is sampled with $P(\hat{Y} = \hat{y}_n) = p_n$, where

$$p_n = \lambda_n \prod_{j=1}^{n-1} (1 - \lambda_j). \tag{2}$$

Here p_n is a geometric probability distribution estimating the probability that halting has occurred at steps $0, \ldots, N$, where N is a hyperparameter setting the maximum number of ponder steps available to the model. In Fig. 1, PonderNet is illustrated in the grey box.

To train PonderNet, the sum of a reconstruction loss L_{REC} and regularization loss L_{REG} is optimized:

$$L = \underbrace{\sum_{n=1}^{N} p_n \mathcal{L}(y, \hat{y}_n)}_{L_{\text{REC}}} + \beta \underbrace{\text{KL}(p_n || p_G(\lambda_p))}_{L_{\text{REG}}}, \tag{3}$$

[2]Banino et al. [1] use $P = 64$ for interpolation and $P = 96$ for extrapolation. In our experimentation, we found that using smaller numbers did not make a difference other than reducing training time.

L_{REC} is equivalent to the expected reconstruction loss across the ponder steps, where $\mathcal{L}$ can be any arbitrary loss function. $p_G(\lambda_p)$ is a pre-determined prior geometric distribution parameterized by the hyperparameter λ_p. By using the KL divergence [21] between p_n and p_G, L_{REG} not only biases the network toward the expected number of ponder steps $1/\lambda_p$ but also promotes exploration by incentivizing non-zero probability of halting for each step. β is an additional hyperparameter, scaling the regularization loss contribution.

Training Details For the parity task, we use a gated recurrent unit (GRU) cell [7] with 128 hidden units. We set $\lambda_p = 0.2$ and $\beta = 0.01$ and minimize binary cross-entropy for $\mathcal{L}$ using the Adam optimizer [19] with learning rate 0.0003 and clip gradient norms at 1 for numerical stability. We train for 80,000 steps using a batch size of 128 and use the model with the highest validation halted step accuracy for evaluation. For MNIST, we make use of the same setup but first pass our inputs through a Convolutional Neural Network (CNN)[3] to generate 64-dimensional embeddings as input to our GRU cell. Here we use cross-entropy instead of binary cross-entropy and train for 20,000 steps.

3.2.2 Ensemble Models

Inspired by Lakshminarayanan et al. [22], we consider "ensembles" for uncertainty quantification. An ensemble model employs a group (or "ensemble") of E deterministic NNs and uses some combination of the models' predictions as its output. The different point estimates can be considered different modes of the Bayesian posterior [18]. One can recover from these point estimates the standard deviation σ and hence standard error $\text{SE} = \sigma/\sqrt{E}$, thus quantifying uncertainty. We developed four different architectures in this fashion, which are outlined below. We set $E = 5$ in all our models. We illustrate ensembling in 1 via the layering of the PonderNet boxes.

GroupThink *GroupThink* consists of E PonderNet modules, each trained independently using its own instance of the optimizer and loss mentioned in Section 3.2.1. When receiving an input, GroupThink passes it to each module, generating an output from each of them. The mean prediction is taken, and a standard error can be quantified as described above.

Rational GroupThink (RGT) GroupThink does nothing with the uncertainty it quantifies. The intuition behind RGT is that a more uncertain PonderNet should expect to ponder for longer and vice-versa. RGT attempts to do this by using its uncertainty to update its prior. More specifically, RGT treats the standard error computed at training step t as input data to a separate sigmoid-activated linear layer trained to compute the appropriate λ_p value for the (now dynamic) regularization loss in training step $t + 1$. Because λ_p is now learned and shared across the E models, the computational graph must be retained when training the first $E-1$ models. Apart from eliminating a hyperparameter, we hypothesize that allowing the model to update its priors should lead to increased performance due to the increased flexibility that is afforded.

Annealed RGT (aRGT) RGT shares the same loss configuration as PonderNet and thus only weakly weighs the regularization term ($\beta = 0.01$). This is, however, the only term that contributes to the optimization of λ_p and should be weighted more strongly if a greater exploration of λ_p values is desired. We conjecture that our L_{REG} term should be more strongly weighted at the beginning of training to encourage the model to explore a more extensive range of λ_p but should ultimately return to being dominated by the reconstruction loss. To achieve this, we anneal β with the schedule:

$$A_\beta(e) = \frac{1}{2e + 1 + \gamma} + \gamma \tag{4}$$

Where $A_\beta(e)$ is the annealed value of β at epoch e, an exponentially decreasing value asymptotically approaching the value of γ, which we set to 0.01 to mimic the effect of β in the previous models at later stages of training.

[3] We use two convolutional layers mapping from 10 to 20 channels using a kernel size of 5, with a dropout layer [32] with $p = 0.5$ in between. We pass our outputs to a dense layer that maps to 64 dimensions. We use ReLU as our activation function throughout.

Lambda GroupThink (lambdaGT) LambdaGT attempts to directly incorporate the uncertainty obtained with ensemble methods in the decision of the models to halt or keep pondering. As shown in Figure 1, we modify the MLP that outputs λ_n, the parameter that defines the halting probability distribution, by expanding it to accept as input a concatenated vector of h_n and SE_n. This latter term is the standard error of the ensemble's predicted output at pondering step n for the prior batch. SE_n for the first batch is initialized as the maximum possible standard error.[4] The modules can now utilize the uncertainty information between their predictions to improve their halting decisions. As in PonderNet, we regularise the halting probability p_n to follow a geometric distribution $p_G(\lambda_p)$.

3.2.3 A Fully Bayesian Approach

A full Bayesian treatment of PonderNet would enable more complete characterization of uncertainty in the model's predictions. This could be incorporated to decide how much to ponder, similarly as above. We took steps in the direction of such an implementation by leveraging the Pyro probabilistic programming language [4] to transform the output layer of the model into a Bayesian layer. Our goal was, given the dataset D and an input datapoint x^*, to learn the posterior predictive distribution

$$p(y^*|x^*, D) = \int_W p(y^*|x^*, W)\, p(W|D)\, dW \tag{5}$$

by learning the posterior $p(W|D)$ on the weights W of the layer. We use SVI to estimate this posterior distribution with a multivariate normal $q_\theta(W)$ parametrized by θ. We seek to minimize, therefore, the KL divergence between $p(W|D)$ and $q_\theta(W)$ which is equivalent to maximizing the evidence lower bound (ELBO) [35]. Our L_{REC} term is replaced by the ELBO and the objective function from (3) becomes:

$$L = -\sum_{n=1}^{N} \Big[p_n\, \text{ELBO}(p(W, D), q_\theta(W))\Big] + \beta\, \text{KL}(p_n||p_G(\lambda_p)), \tag{6}$$

where we conserve the weighted average by the halting probability p_n per step n to adequately add up the contributions of each step and the regularization term to encourage exploration.

Ultimately, this approach was unsuccessful due to significant variance and a lack of maturity in the development ecosystem. The Bayesian treatment further increased the numerical instability of PonderNet, with its gradients exhibiting extremely high variance and dependence on prior distributions and hyperparameters. Despite our best efforts searching for adequate hyperparameters, this led to little to no learning or numerical instability in the worst cases. Additionally, we found it challenging to navigate the probabilistic programming ecosystem for implementing BDL solutions at a more advanced and customized level.

4 Results and Analysis

Table 1 presents the test accuracy and average halting step of each model presented in Section 3.2. We successfully reproduce the results shown in the original paper, with a higher accuracy achieved in interpolation (99.2 %) than extrapolation (90.8 %) and a demonstrated adaptation to the more difficult task via the increase in average halting step (10.2 steps vs. 8.5). We verify PonderNet's robustness by observing the same trends on the MNIST dataset. We achieve test accuracies and average halting steps of 99.2 ± 0.002 % and 9.1 ± 0.1 in interpolation and 63 ± 0.004 % and 10.6 ± 0.3 in extrapolation. The much lower extrapolation test accuracy indicates that our image rotation was too challenging compared to the additional non-zero elements in the parity task.

PonderNet matches or outperforms three out of our four ensemble variants. RGT and aRGT suffered in particular to numerical instability, causing several if not all the seeds to halt training prematurely. This can also be noted in Fig. 2. The annealing proposed in aRGT seems to help, but overall this family of variants misses the mark.

[4]Note that using the prior batch's error may lead to issues of significant variance when training. LambdaGT can be adapted to resolve this by using the same batch in a two-step training process: first, freeze the weights that make use of SE_n and backpropagate through the rest of the model, calculate SE_n, and second, freeze all other weights and backpropagate through the weights that use SE_n. We find, however, that the naive implementation trains successfully and requires less compute.

Table 1: Average test set accuracy and halting step of the models on the parity datasets for interpolation and extrapolation. The standard error in the metrics from runs using 5 different seeds is also reported.

	Test Accuracy		Halting Step	
	Interpolation	Extrapolation	Interpolation	Extrapolation
PonderNet	0.992 ± 0.005	0.908 ± 0.076	8.5 ± 1.4	10.2 ± 1.8
GroupThink	0.986 ± 0.009	0.883 ± 0.049	8.9 ± 0.3	9.3 ± 0.3
RGT	0.689 ± 0.084	0.502 ± 0.001	9.1 ± 3.1	2.4 ± 0.2
aRGT	0.745 ± 0.093	0.502 ± 0.001	6.4 ± 1.7	3.8 ± 1.2
lambdaGT	$\mathbf{0.995 \pm 0.0002}$	$\mathbf{0.919 \pm 0.022}$	8.7 ± 0.3	9.9 ± 0.3

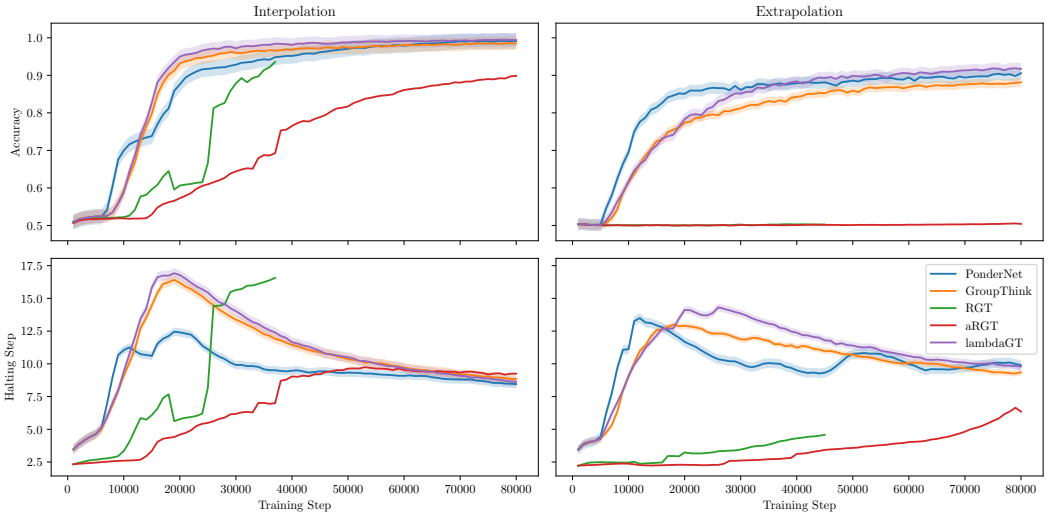

Figure 2: Average validation accuracy and halting step during training for the four models, with standard error. Metrics were computed over 5 random seeds. RGT and aRGT suffer from numerical instability causing premature termination and therefore difficulty in seed aggregation.

Our lambdaGT variant generally outperforms the rest of the model zoo, marginally improving on the performance of PonderNet. We also note from Fig. 2 that lambdaGT converges earlier than PonderNet on interpolation. This supports our hypothesis that additional information from the model's uncertainty aids learning. LambdaGT accuracies also have the smallest SEs, indicating greater stability across runs. Finally, both lambdaGT and GroupThink explore more halting steps than PonderNet during training, suggesting that the ensemble treatment aids exploration and thus more robust halting step convergence.

We hope future work can successfully apply a complete Bayesian treatment to the problem (as discussed in Section 3.2.3) to complete the comparison, perhaps on a new, less saturated task. We also wish to investigate whether lambdaGT outperforms RGT for inherent reasons or whether addressing the numerical instability of the latter could lead to different results. The impact of different initializations of the λ_p term or of additional loss terms and training regimens could be avenues to explore to redeem the RGT family.

5 Conclusion

In this work, we explored approaches for the Bayesian treatment of PonderNet. We replicate the results of the original paper on the parity task and test on MNIST to verify robustness. We then propose, implement and evaluate four ensemble-based methods for uncertainty quantification and exploitation and compare their performance to PonderNet. One of our variants, lambdaGT, outperforms all other models. Future work may leverage a complete Bayesian treatment of PonderNet or investigate more efficient ensembling techniques [11].

References

[1] A. Banino, J. Balaguer, and C. Blundell. PonderNet: Learning to Ponder. *arXiv:2107.05407 [cs]*, Sept. 2021.

[2] T. Bayes and n. Price. LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S. *Philosophical Transactions of the Royal Society of London*, 53:370–418, Jan. 1763. doi: 10.1098/rstl.1763.0053.

[3] L. Bertinetto, J. F. Henriques, J. Valmadre, P. Torr, and A. Vedaldi. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[4] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019. ISSN 1533-7928. URL `http://jmlr.org/papers/v20/18-403.html`.

[5] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Network. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1613–1622. PMLR, June 2015.

[6] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive Neural Networks for Efficient Inference. In *Proceedings of the 34th International Conference on Machine Learning*, pages 527–536. PMLR, July 2017.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Dec. 2014.

[8] L. Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6):141–142, Nov. 2012. ISSN 1558-0792. doi: 10.1109/MSP.2012.2211477.

[9] V. Fortuin. Priors in Bayesian Deep Learning: A Review, Mar. 2022.

[10] Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1050–1059. PMLR, June 2016.

[11] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[12] A. Graves. Adaptive Computation Time for Recurrent Neural Networks. *arXiv:1603.08983 [cs]*, Feb. 2017. URL `http://arxiv.org/abs/1603.08983`. arXiv: 1603.08983.

[13] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang. Dynamic Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. ISSN 1939-3539. doi: 10.1109/TPAMI.2021.3117837.

[14] J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1861–1869, Lille, France, July 2015. JMLR.org.

[15] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger. Multi-Scale Dense Networks for Resource Efficient Image Classification. In *International Conference on Learning Representations*, Feb. 2018.

[16] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456. PMLR, June 2015.

[17] E. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14 (6):1569–1572, Nov. 2003. ISSN 1941-0093. doi: 10.1109/TNN.2003.820440.

[18] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun. Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users. *IEEE Computational Intelligence Magazine*, 17(2):29–48, May 2022. ISSN 1556-6048. doi: 10.1109/MCI.2022.3155327.

[19] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, Jan. 2017.

[20] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014.

[21] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. ISSN 0003-4851.

[22] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[23] J. Lin, Y. Rao, J. Lu, and J. Zhou. Runtime Neural Pruning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[24] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, Sept. 2018.

[25] C. Louizos and M. Welling. Multiplicative Normalizing Flows for Variational Bayesian Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2218–2227. PMLR, July 2017.

[26] W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, Dec. 1997. ISSN 0893-6080. doi: 10.1016/S0893-6080(97)00011-7.

[27] D. J. C. MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, May 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448.

[28] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, USA, Aug. 2012. ISBN 978-0-262-01802-9.

[29] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer Science & Business Media, Dec. 2012. ISBN 978-1-4612-0745-0.

[30] L. Rimella and N. Whiteley. Dynamic Bayesian Neural Networks, June 2020.

[31] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. Nov. 2016.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15 (56):1929–1958, 2014. ISSN 1533-7928.

[33] H. Wang and D.-Y. Yeung. A Survey on Bayesian Deep Learning. *ACM Computing Surveys*, 53 (5):108:1–108:37, Sept. 2020. ISSN 0360-0300. doi: 10.1145/3409383.

[34] X. Wang, F. Yu, R. Wang, T. Darrell, and J. E. Gonzalez. TAFE-Net: Task-Aware Feature Embeddings for Low Shot Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1831–1840, 2019.

[35] D. Wingate and T. Weber. Automated Variational Inference in Probabilistic Programming. Technical Report arXiv:1301.1299, arXiv, Jan. 2013. URL `http://arxiv.org/abs/1301.1299`. arXiv:1301.1299 [cs, stat] type: article.

A Numerical Instability

A number of the models explored in this work suffer from numerical instability. This is where floating-point errors propagate through our algorithms and cause the resulting malformed input to lead to premature termination. We briefly investigated the source of this numerical instability, as it also seemed to affect PonderNet, as mentioned in Banino et al. [1]'s work.

For PonderNet and our ensemble models described in Section 3.2.2, we unsurprisingly found the culprit to be vanishing and exploding gradients. Numerical instability was less prevalent in the fully Bayesian approach of Section 3.2.3, making it more challenging to reproduce the numerical stability issues consistently. We do however suspect that the source of the issue is ultimately the same, namely the vanishing/exploding gradients issue mentioned earlier.

Aside from the already present gradient clipping, we briefly considered a number of techniques for addressing the issue further. We experimented with using other optimization settings, including

- using vanilla Stochastic Gradient Descent (SGD) with a scheduled learning rate.
- using the AdamW optimizer [24].
- Varying the hyperparameters of our Adam-based optimizer, such as the base learning rate or the epsilon term.

We also considered employing Batch Normalization [16] and using ReLU whenever sensible. We found that increasing the epsilon term in Adam optimization from $1\mathrm{e}{-8}$ to $1\mathrm{e}{-6}$ generally helped. However, ultimately none of our hotfixes fully solved our issue, often trading stability for learning capabilities.

Due to a limited budget, we were unable to explore solutions to these issues beyond these hotfixes, but we believe a solution is certainly possible. We hope future work can be dedicated to this direction.