Practical Considerations for Probabilistic Backpropagation

Matt Benatan
IBM Research UK
Sci-Tech Daresbury
Warrington, UK.
matthew.benatan@ibm.com

Edward O. Pyzer-Knapp*
IBM Research UK
Sci-Tech Daresbury
Warrington, UK.
epyzerk3@uk.ibm.com

Abstract

Probabilistic Backpropagation (PBP) was developed to address the scalability issue of Bayesian neural networks, and facilitates tractable Bayesian learning of networks with large structures and large amounts of data. PBP exhibits competitive performance when compared to both traditional backpropagation and to other state-of-the-art Bayesian methods. In this paper we provide guidelines for improving training time for PBP through applying early stopping, mini-batching, and lock-free asynchronous updates. We also introduce a method for extending PBP to classification tasks. Performance evaluations carried out on a number of real-world datasets show that significant enhancements to training time can be achieved via the proposed methods, and that PBP can be successfully applied for classification.

1 Introduction

Deep learning has achieved state-of-the-art results for machine learning problems across a range of application contexts [14]. However, the large amounts of data and high complexity of the networks used makes hyper-parameter optimisation difficult. These methods also do not to capture model uncertainty, thus model confidence can't be accounted for. Bayesian neural networks address hyper parameter optimisation by estimating the model parameters through posterior inference, and in doing so, they also account for model uncertainty. while they exhibit clear advantages, many Bayesian neural network methods do not scale well [11][10][9][8].

Probabilistic Backpropagation (PBP) [7] is one of a number of methods that have been proposed for integrating Bayesian methods with neural networks [4] [15] [6]. PBP works by first obtaining the likelihood through propagating weights forward through the network, after which it backpropagates the gradients of the log-marginal likelihood with respect to the parameters of the posterior approximation. This has proven to be an effective approach for Bayesian learning of network parameters, and has yielded competitive results when compared to contemporary methods.

In this paper, we introduce methods for adapting early stopping [13], mini-batching [2], and lock-free asynchronous updating [12] for PBP, before presenting an adaptation of PBP for classification.

2 Contributions

In this section we detail a number of methods to improve PBP through 1) early stopping, 2) mini-batching, 3) lock-free asynchronous updating, and 4) adapting PBP for classification.

One noteable implementation detail is as follows: the initial paper details that γ_i becomes unstable for $\alpha < -30$ [7]. In our work, we additionally found that γ becomes unstable for values of $\alpha > 8$, and

*Corresponding author

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

that the ideal α bounds for γ_i are $-33 < \alpha < 8$. We discovered that using γ_i for these α values, and the approximation of γ_i for α values outside of this range, produce better stability during training.

Generalisation loss (GL) is defined as the ratio between the current validation error, $E_{va}(i)$, and the best validation error achieved so far, $E_{opt}(i)$ [13]. The stopping criterion is defined as the point at which GL exceeds a threshold, α. In this work, we have adapted the generalisation loss to PBP by using the reciprocal of the validation likelihood instead of the validation error, as it is the log-marginal likelihood that is used during backpropagation. We use the likelihood as this ensures positive values, whereas the log-likelihood can take the form of both positive and negative values. We take the reciprocal of the likelihood values as they behave similarly to the error term used in the original form of GL. We therefore define the GL at epoch i as:

$$GL(i) = 100 \cdot \left(\frac{L_{va}(i)^{-1}}{L_{opt}(i)^{-1}} - 1 \right)$$

where L_{va} and L_{opt} are the current and best validation likelihood values at time t, respectively.

Another approach is to use the training progress (P_k) in combination with GL [13], thus incorporating information on training set performance. $P_k(i)$ is obtained by dividing the average objective by the minimum objective w.r.t a sliding window over k epochs. As with GL, we have adapted the original equation for $P_k(i)$ by using the reciprocal of the likelihoods:

$$P_k(i) = 1000 \cdot \left(\frac{\sum_{i'=i-k+1}^{i} L_{tr}(i')^{-1}}{k \cdot min_{i'=i-k+1}^{i} L_{tr}(i')^{-1}} - 1 \right)$$

We then define our stopping criteria, using the quotient of $GL(i)$ and $P_k(i)$:

$$PQ_\alpha = \frac{GL(i)}{P_k(i)} > \alpha.$$

The last early stopping method explored here is patience [2], whereby the validation error is monitored, and training is halted if this fails to improve for some k epochs. In this work, we use validation log-likelihood in place of validation error.

Another method to improve PBP training time is lock-free asynchronous updating [12]. This simply involves running each training step in parallel within each epoch, and updating values in the network asynchronously as and when each step has completed.

Similarly to [5], we present a modification of PBP for classification tasks. In computing the log-marginal likelihood for PBP, a loss, L, is incorporated:

$$-0.5 \cdot \left(\frac{\log(v_f) + L}{v_f} \right)$$

where v_f is the linear activation of the variance weights from the final layer, and L is the squared loss defined as $(y - y')^2$, where y' is the linear activation of the mean weights from the final layer and y is a target value. To adapt PBP for classification, we represent our class targets as one-hot vectors, replace PBP's linear activation on the final layer's mean weights with a softmax activation, and replace the squared loss with a cross-entropy loss [3].

3 Results

3.1 Early Stopping

For the early stopping investigations we have used five datasets from the original paper. For these, a PBP network with one hidden layer of 50 units is used. For the baseline, we set the number of epochs to 40; for each other test, we set the maximum number of epochs to 500. The α values used for the GL and PQ methods were set to 1.0 and 0.01 respectively for all datasets, and k is set to 10 for all early stopping methods. Training is run 20 times for each early stopping method and dataset, randomly reinitialising the train/test split each time.

		Test RMSE				Test LL			
Dataset	N	**Baseline**	**GL**	**PQ**	**Patience**	**Baseline**	**GL**	**PQ**	**Patience**
Boston	506	**3.35**	3.68	3.6	3.49	−2.76	−2.75	−2.73	**-2.71**
Energy	768	2.01	1.56	**1.47**	1.78	−2.25	**-1.80**	−1.83	−2.05
Concrete	1030	5.77	6.07	5.76	**5.74**	−3.19	−3.23	−3.19	**-3.18**
Wine	1599	0.63	**0.62**	0.63	**0.62**	−0.95	−0.95	−0.96	**-0.94**
Yacht	308	0.92	1.37	1.18	**0.75**	−1.56	−1.78	−1.59	**-1.11**

Table 1: Mean results from early stopping tests.

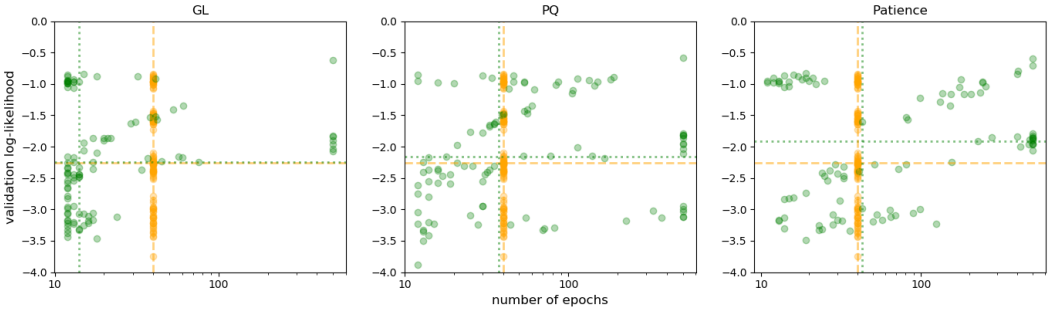

Figure 1: Scatter-plot of early stopping results for all datasets.
Orange: baseline. Green: results from method. Lines denote medians.

As demonstrated in Table 1 and Figure 1, the best results are often achieved when using patience. However, all of the early stopping methods frequently stop training before the number of epochs used in the baseline - with GL typically stopping between 10 and 20 epochs, while suffering only minor RMSE and log-likelihood (LL) performance degradation. PQ and patience often take longer to reach the stopping criterion, but achieve better RMSE and LL results. This indicates that there is a trade-off when selecting early stopping methods: GL can reduce training time and maintain good performance, while the other two methods can improve performance, but may take longer to train.

3.2 Mini-Batching

This set of investigations looks at the effects of mini-batching on six openly-available datasets of varying size [16][1]: Malaria (18,924), kin8nm (8192), Combined Cycle Power Plant (9568), Year Prediction MSD (515,345), Naval Propulsion (11,934), and Protein Structure (45,730). The same network structure as the first investigation was used, apart from the two largest datasets - Protein Structure and Year Prediction MSD - which use one hidden layer of 100 units, as per the original PBP paper [7]. Training was repeated ten times for each mini-batch size and dataset. The mean test-set log-likelihood and training duration were used to construct Figure 2. We normalize for hardware-specific timings by presenting results as a comparison to baseline training times.

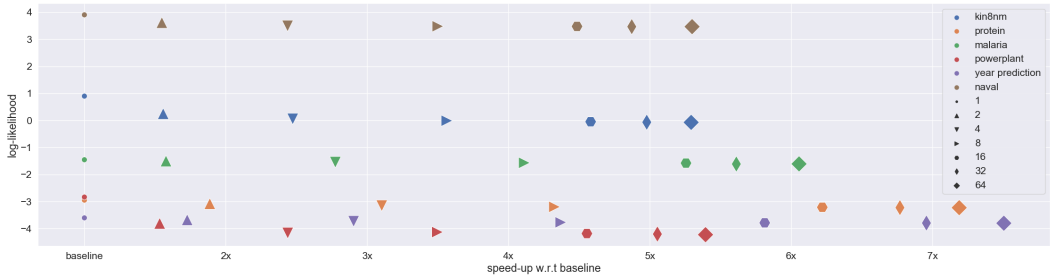

Figure 2: Scatter plot of mini-batching test-set results.

As demonstrated in Figure 2, increasing the batch size greatly improves training time, obtaining a speed-up of >5x on all datasets, with the largest dataset, Year Prediction MSD, obtaining a speed-up of over 7.5x. The improved training time comes at the cost of reduced performance for some

datasets, particularly at mini-batch sizes of 32 and 64, however this reduction in performance may be outweighed by the improved training time depending on the application.

3.3 Combined Methods

We combined the early stopping and mini-batching methods described above with the method for lock-free asynchronous updates described in [12], and applied these to the two largest datasets: Year Prediction MSD and Protein Structure. We used the GL method for early-stopping, and used a mini-batch size of 16, as these parameters provided the greatest speedup with minimal degradation in performance. We compare a baseline with no mini-batching, early stopping or asynchronous updates to 1) mini-batching and early stopping, 2) mini-batching, early stopping and 2 asynchronous workers, and 3) mini-batching, early stopping and 4 asynchronous workers. The following uses the mean results from 5 runs of each configuration.

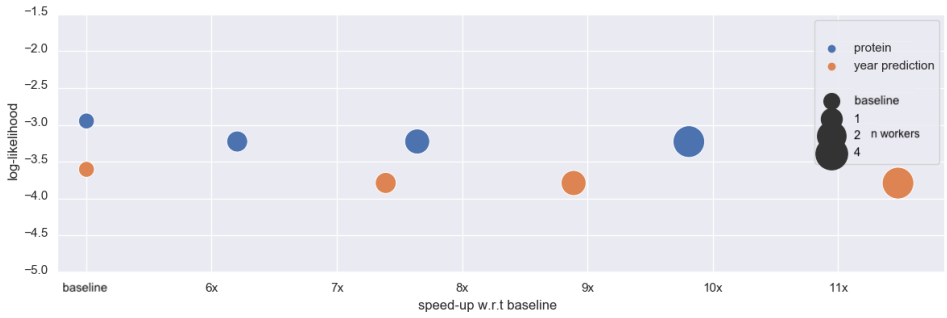

Figure 3: Scatter plot of results from combined methods.

As demonstrated in Figure 3, significant speedup can be attained when combining these three approaches - with a speedup of >11x the baseline with four workers when compared to none of the additional methods.

3.4 Classification

Here, we demonstrate the results of applying PBP classification to a number of classification tasks with varying dimensionality: three binary classification tasks, and one multi-class classification task - all sourced from the UCI ML Repository [1].

				Classification Performance			
Dataset	**Size**	**n classes**	d	**Accuracy**	**Precision**	**Recall**	**F-score**
Digits	1797	10	64	0.921	0.921	0.916	0.915
Breast Cancer	569	2	30	0.977	0.974	0.978	0.975
Tic-Tac-Toe	958	2	9	0.879	0.833	0.91	0.853
Phishing	11055	2	30	0.951	0.95	0.952	0.951

Table 2: PBP classification results.

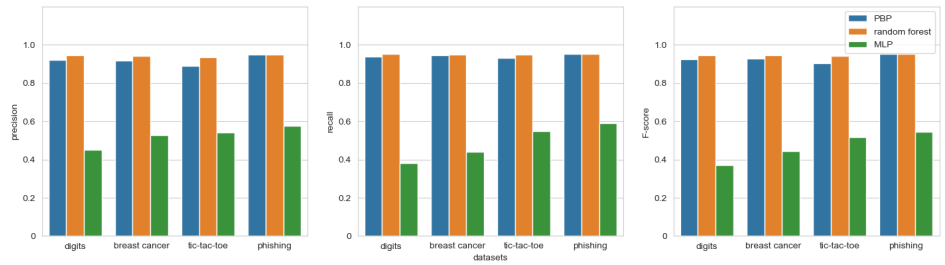

Figure 4: Bar chart comparing PBP performance with neural network (green) and random forest (orange) classifiers.

We compare PBP with two other classifiers. The first is a feedforward neural network (NN) with the same architecture as the PBP model (one hidden layer of 20 neurons) trained using backpropagation with an Adam optimizer. The second is a random forest classifier with 10 trees and a maximum tree-depth of 10. The PBP and NN methods are each trained for 40 epochs for fair comparison. As shown in Table 2 and Figure 4, the classification variant of PBP can be successfully applied to a range of tasks, achieving strong classification results in both binary and multi-class applications. These results significantly outperform the NN, achieving similar performance to the random forest, but with the addition of a high quality model for uncertainty.

4 Conclusion

We have introduced a number of practical considerations for PBP, and have examined how the proposed methods impact PBP's performance, training time, and its application to classification tasks.

The early stopping methods provide a number of ways to improve training - either with respect to training duration, or with respect to performance - while replacing the $epochs$ hyperparameter with a better defined parameter, α, which is more suitable to place a prior over.

We have also demonstrated that mini-batching can be successfully applied to significantly reduce training time for PBP. Additionally, we show that the combination of early stopping, mini-batching, and the lock-free asynchronous update strategy can achieve a significant reduction in training time while maintaining strong performance.

Lastly, we demonstrated that PBP can be successfully adapted for classification tasks using the proposed method, and that this is effective for both binary and multi-class classification problems.

References

[1] UCI Machine Learning Repository.

[2] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533 [cs]*, June 2012. arXiv: 1206.5533.

[3] D. Campbell, R. A. Dunne, and N. A. Campbell. *On The Pairing Of The Softmax Activation And Cross–Entropy Penalty Functions And The Derivation Of The Softmax Activation Function*.

[4] Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *arXiv:1506.02142 [cs, stat]*, June 2015. arXiv: 1506.02142.

[5] S. Ghosh, F. Maria Delle Fave, and J. Yedidia. Assumed Density Filtering Methods for Learning Bayesian Neural Networks. Phoenix, AZ, USA, Feb. 2016.

[6] A. Graves. Practical Variational Inference for Neural Networks. NIPS'11, pages 2348–2356, USA, 2011. Curran Associates Inc.

[7] J. M. Hernández-Lobato and R. P. Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. *arXiv:1502.05336 [stat]*, Feb. 2015. arXiv: 1502.05336.

[8] G. E. Hinton and D. van Camp. Keeping the Neural Networks Simple by Minimizing the Description Length of the Weights. COLT '93, pages 5–13, New York, NY, USA, 1993. ACM.

[9] P. Jylänki, A. Nummenmaa, and A. Vehtari. Expectation Propagation for Neural Networks with Sparsity-promoting Priors. *arXiv:1303.6938 [stat]*, Mar. 2013. arXiv: 1303.6938.

[10] D. J. C. MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, May 1992.

[11] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996.

[12] F. Niu, B. Recht, C. Re, and S. J. Wright. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. *arXiv:1106.5730 [cs, math]*, June 2011. arXiv: 1106.5730.

[13] L. Prechelt. Early Stopping-But When? pages 55–69, London, UK, UK, 1998. Springer-Verlag.

[14] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan. 2015.

[15] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. *arXiv:1502.05700 [stat]*, Feb. 2015. arXiv: 1502.05700.

[16] T. Spangenberg, J. N. Burrows, P. Kowalczyk, S. McDonald, T. N. C. Wells, and P. Willis. The Open Access Malaria Box: A Drug Discovery Catalyst for Neglected Diseases. *PLoS ONE*, 8(6):e62906, June 2013.