Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks

José Miguel Hernández-Lobato

joint work with Ryan P. Adams

Workshop on Gaussian Process Approximations

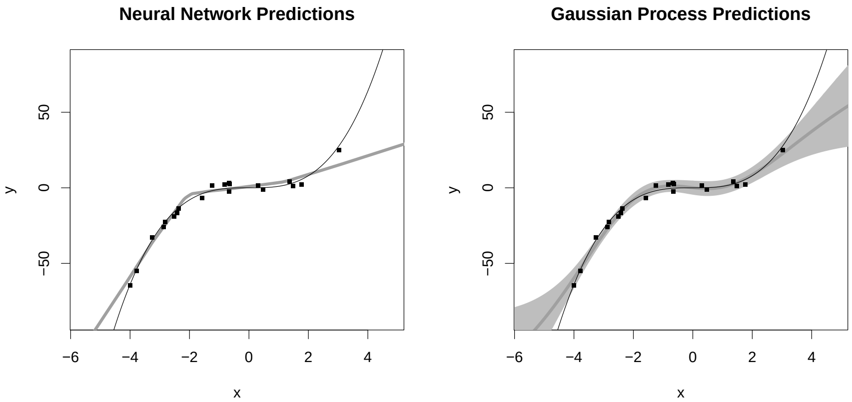
May 21, 2015

VE RI TAS

HARVARD
INTELLIGENT
PROBABILISTIC
SYSTEMS

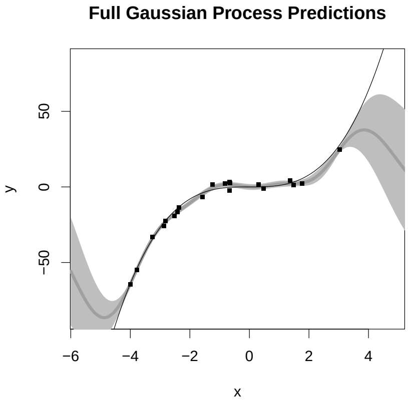
Infinitely-big Bayesian Neural Networks

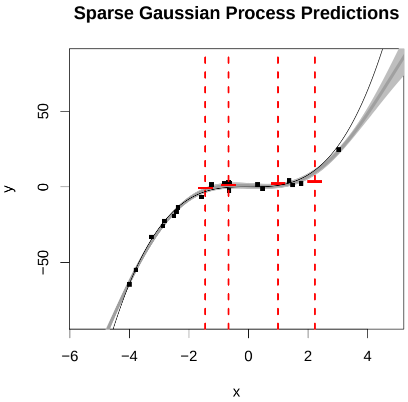
- Neal [1996] showed that a **neural network** (NN) converges to a **Gaussian Process** (GP) as the number of hidden units increases.
- GPs allow for **exact** Bayesian inference. Learning infinitely-big Bayesian networks is then **easier** and more **robust to overfitting**.



Sparse Gaussian Processes

- The price paid is in **scalability**. From $\mathcal{O}(n)$ we go to $\mathcal{O}(n^3)$. The **Non-parametric** approach is infeasible with massive data!
- Solution: transform the full GP back into a **sparse GPs** using m inducing points. From $\mathcal{O}(n^3)$ we go back to $\mathcal{O}(n)$.





Sparse Gaussian Processes as Parametric Methods

FITC approximation: the most widely used method for sparse GPs.

The evaluations $\mathbf{f}$ of the function are **conditionally independent** given the value $\mathbf{u}$ of the function at the m inducing points:

$$p(\mathbf{f}|\mathbf{u}) \approx \tilde{p}(\mathbf{f}|\mathbf{u}) = \prod_{i=1}^{n} \mathcal{N}(f_i|\mathbf{K}_{f_i,\mathbf{u}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{u}, \mathbf{k}_{f_i f_i} - \mathbf{K}_{f_i\mathbf{u}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{K}_{\mathbf{u}f_i})\,.$$

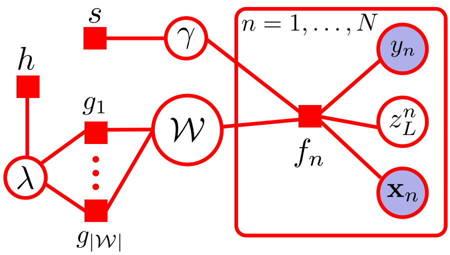
The values $\mathbf{u}$ at the inducing points are the **parameters** of the sparse GP.

A Gaussian approximation $q(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{m}, \mathbf{V})$ can then be adjusted to the posterior on $\mathbf{u}$ using scalable **stochastic** and **distributed** **VI** or **EP** (Hensman et al. [2013, 2014], Hernández-Lobato et al. [2015]).

Is the cycle *parametric* $\rightarrow$ *non-parametric* $\rightarrow$ *parametric* worth it?
Perhaps, because scalable inference in NNs is very hard, or perhaps not...

Probabilistic Multilayer Neural Networks

- L layers with $\mathcal{W} = \{\mathbf{W}_l\}_{l=1}^L$ as weight matrices and outputs $\{\mathbf{z}_l\}_{l=0}^L$.
- The l-th layer input is $\mathbf{a}_l = \mathbf{W}_l \mathbf{z}_{l-1} / \sqrt{\dim(\mathbf{z}_{l-1})}$.
- ReLUs activation functions for the hidden layers: $a(x) = \max(x, 0)$.
- **The likelihood**: $p(\mathbf{y}|\mathcal{W}, \mathbf{X}, \gamma) = \prod_{n=1}^N \mathcal{N}(y_n | z_L(\mathbf{x}_n|\mathcal{W}), \gamma^{-1}) \equiv f_n$.
- **The priors**: $p(\mathcal{W}|\lambda) = \prod_{l=1}^L \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l}|0, \lambda^{-1}) \equiv g_k$,
  $p(\lambda) = \text{Gamma}(\lambda|\alpha_0^\lambda, \beta_0^\lambda) \equiv h$, $p(\gamma) = \text{Gamma}(\gamma|\alpha_0^\gamma, \beta_0^\gamma) \equiv s$.



The **posterior approximation** is

$$q(\mathcal{W}, \gamma, \lambda) = \left[\prod_{l=1}^L \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{ij,l}|m_{ij,l}, v_{ij,l})\right] \text{Gamma}(\gamma|\alpha^\gamma, \beta^\gamma) \text{Gamma}(\lambda|\alpha^\lambda, \beta^\lambda) .$$

Probabilistic Backpropagation (PBP)

PBP (Hernández-Lobato and Adams [2015]) is based on the **assumed density filtering** (ADF) algorithm (Opper and Winther [1998]).

After seeing the n-th data point, our beliefs about w are updated as

$$p(w) = Z^{-1}\mathcal{N}(y_n | z_L(\mathbf{x}_n | w), \gamma^{-1})\,\mathcal{N}(w | m, v)\,,$$

where Z is the normalization constant.

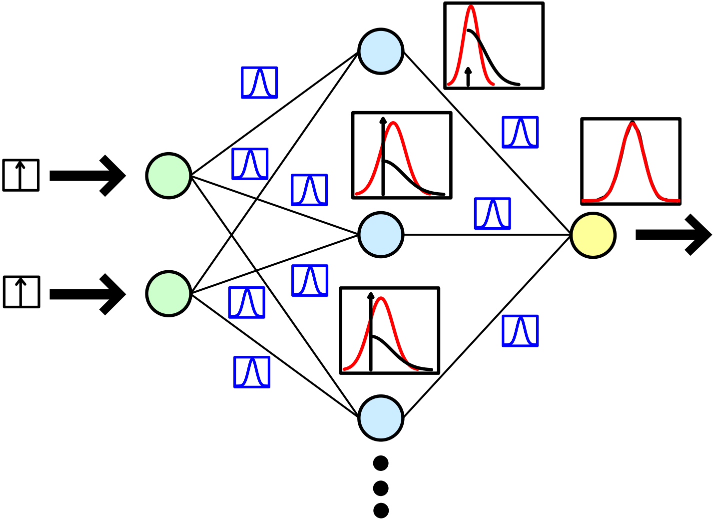
The parameters of the new Gaussian beliefs $q^{\text{new}}(w) = \mathcal{N}(w | m^{\text{new}}, v^{\text{new}})$ that minimize the the KL divergence between $p(w)$ and $q^{\text{new}}(w)$ are then

$$\boxed{\begin{aligned} m^{\text{new}} &= m + v\frac{\partial \log Z}{\partial m}\,, \\ v^{\text{new}} &= v - v^2\left[\left(\frac{\partial \log Z}{\partial m}\right)^2 - 2\frac{\partial \log Z}{\partial v}\right]\,. \end{aligned}}$$

We need a way to approximate Z and then obtain its gradients!

Forward Pass

Propagate distributions through the network and approximate them with **Gaussians** by moment matching.



Backward Pass and Implementation Details

Once we compute $\log Z$, we obtain its gradients by **backpropagation**.

Like in classic backpropagation, we obtain a recursion in terms of **deltas**:

$$\delta_j^m = \frac{\partial \log Z}{\partial m_j^a} = \sum_{k \in O(j)} \left\{ \delta_k^m \frac{\partial m_k^a}{\partial m_j^a} + \delta_k^v \frac{\partial v_k^a}{\partial m_j^a} \right\} ,$$

$$\delta_j^v = \frac{\partial \log Z}{\partial v_j^a} = \sum_{k \in O(j)} \left\{ \delta_k^m \frac{\partial m_k^a}{\partial v_j^a} + \delta_k^v \frac{\partial v_k^a}{\partial v_j^a} \right\} .$$

Can be automatically implemented with **Theano** or **autograd**.

Implementation details:

- Approximation of the Student's t likelihood with a Gaussian.
- We do several passes over the data with ADF.
- The approximate factors for the prior are updated using EP.
- Posterior approximation q initialized with random mean value.

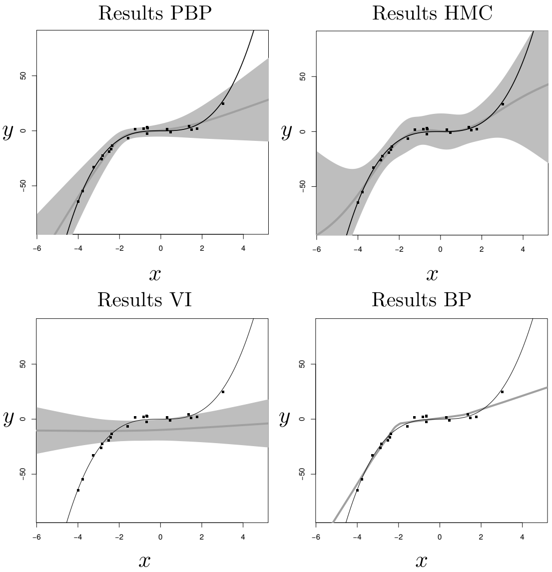
Results on Toy Dataset

40 training epochs.

100 hidden units.

VI uses two stochastic approximations to the lower bound (Graves [2011]).

BP and VI tuned with Bayesian optimization (`www.whetlab.com`).



Exhaustive Evaluation on 10 Datasets

Table : Characteristics of the analyzed data sets.

Dataset	N	d
Boston Housing	506	13
Concrete Compression Strength	1030	8
Energy Efficiency	768	8
Kin8nm	8192	8
Naval Propulsion	11,934	16
Combined Cycle Power Plant	9568	4
Protein Structure	45,730	9
Wine Quality Red	1599	11
Yacht Hydrodynamics	308	6
Year Prediction MSD	515,345	90

Always 50 hidden units except in Year and Protein where we use 100.

Average Test RMSE

Table : Average test RMSE and standard errors.

Dataset	**VI**	**BP**	**PBP**
Boston	4.320±0.2914	3.228±0.1951	**3.010±0.1850**
Concrete	7.128±0.1230	5.977±0.2207	**5.552±0.1022**
Energy	2.646±0.0813	**1.185±0.1242**	1.729±0.0464
Kin8nm	0.099±0.0009	**0.091±0.0015**	0.096±0.0008
Naval	0.005±0.0005	**0.001±0.0001**	0.006±0.0000
Power Plant	4.327±0.0352	4.182±0.0402	**4.116±0.0332**
Protein	4.842±0.0305	**4.539±0.0288**	4.731±0.0129
Wine	0.646±0.0081	0.645±0.0098	**0.635±0.0078**
Yacht	6.887±0.6749	1.182±0.1645	**0.922±0.0514**
Year	9.034±NA	8.932±NA	**8.881± NA**

Average Training Time in Seconds

PBP does not need to optimize hyper-parameters and is run **only once**.

Table : Average running time in seconds.

Problem	VI	BP	PBP
Boston	1035	677	**13**
Concrete	1085	758	**24**
Energy	2011	675	**19**
Kin8nm	5604	2001	**156**
Naval	8373	2351	**220**
Power Plant	2955	2114	**178**
Protein	7691	4831	**485**
Wine	1195	917	**50**
Yacht	954	626	**12**
Year	142,077	65,131	**6119**

These results are for the **Theano** implementation of PBP. **C code** for PBP based on open-blas is about 5 times faster.

Comparison with Sparse GPs

VI implementation described by Hensman et al. [2014].
Same number m of pseudo-inputs as hidden units in neural networks.
Stochastic optimization with ADADELTA and **minibatch size m**.

Table : Average Test Log-likelihood.

Dataset	**SGP**	**PBP**
Boston	-2.614±0.074	**-2.577±0.095**
Concrete	-3.417±0.031	**-3.144±0.022**
Energy	**-1.612±0.022**	-1.998±0.020
Kin8nm	0.872±0.008	**0.919±0.008**
Naval	**4.320±0.039**	3.728±0.007
Power Plant	-2.997±0.016	**-2.835±0.008**
Protein	-3.046±0.006	**-2.973±0.003**
Wine	-1.071±0.023	**-0.969±0.014**
Yacht	-2.507±0.062	**-1.465±0.021**
Year	-3.793±NA	**-3.603± NA**

These results are a collaboration with Daniel Hernández-Lobato.

Results with Deep Neural Networks

Performance of networks with up to 4 hidden layers.

Same number of units in each hidden layer as before.

Table : Average Test RMSE.

Dataset	BP_1	BP_2	BP_3	BP_4	PBP_1	PBP_2	PBP_3	PBP_4
Boston	3.23±0.195	3.18±0.237	3.02±0.185	2.87±0.157	3.01±0.180	**2.80±0.159**	2.94±0.165	3.09±0.152
Concrete	5.98±0.221	5.40±0.127	5.57±0.127	5.53±0.139	5.67±0.093	**5.24±0.116**	5.73±0.108	5.96±0.160
Energy	1.18±0.124	0.68±0.037	**0.63±0.028**	0.67±0.032	1.80±0.048	0.90±0.048	1.24±0.059	1.18±0.055
Kin8nm	0.09±0.002	0.07±0.001	0.07±0.001	0.07±0.001	0.10±0.001	**0.07±0.000**	0.07±0.001	0.07±0.001
Naval	0.00±0.000	**0.00±0.000**	0.00±0.000	0.00±0.000	0.01±0.000	0.00±0.000	0.01±0.001	0.00±0.001
Plant	4.18±0.040	4.22±0.074	4.11±0.038	4.18±0.059	4.12±0.035	**4.03±0.035**	4.06±0.038	4.08±0.037
Protein	4.54±0.023	4.18±0.027	4.02±0.026	**3.95±0.016**	4.69±0.009	4.24±0.014	4.10±0.023	3.98±0.032
Wine	0.65±0.010	0.65±0.011	0.65±0.010	0.65±0.016	**0.63±0.008**	0.64±0.008	0.64±0.009	0.64±0.008
Yacht	1.18±0.164	1.54±0.192	1.11±0.086	1.27±0.129	1.01±0.054	**0.85±0.049**	0.89±0.099	1.71±0.229
Year	8.93±NA	8.98±NA	8.93±NA	9.04±NA	**8.87± NA**	8.92±NA	8.87±NA	8.93±NA

Summary and Future Work

Summary:

- PBP is a state-of-the-art method for scalable inference in NNs.
- PBP is very similar to traditional backpropagation.
- PBP often outperforms backpropagation at a lower cost.
- PBP seems to outperform sparse GPs.

Very fast C code available at https://github.com/HIPS

Future Work:

- Extension to multi-class classification problems.
- Can PBP be efficiently implemented using minibatches?
- ADF seems to be better than EP.
  Can PBP benefit from minimizing an α divergence?
- Can we avoid posterior variance shrinkage?
  Collaboration with Rich Turner and Yingzhen Li.
- Can deep GPs benefit from similar inference techniques?

Thanks!

Thank you for your attention!

References I

A. Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems 24*, pages 2348–2356. Curran Associates, Inc., 2011.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, page 282, 2013.

J. Hensman, A. Matthews, and Z. Ghahramani. Scalable variational gaussian process classification. In *International Conference on Artificial Intelligence and Statistics*, pages 351–360, 2014.

J. M. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *ICML*. 2015.

D. Hernández-Lobato et al. Scalable gaussian process classification via expectation propagation. In *GP Approximations Workshop, Copenhagen*, 2015.

References II

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

M. Opper and O. Winther. A Bayesian approach to on-line learning. *On-line Learning in Neural Networks, ed. D. Saad*, pages 363–378, 1998.