UNCERTAINTY IN RECURRENT NEURAL NETWORKS

ALIREZA SAMAR

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Master of Philosophy

Advanced Informatics School
Universiti Teknologi Malaysia

APRIL 2017

ABSTRACT

Deep learning has outperformed in various fields from computer vision, and language processing to physics, biology, and manufacturing. This means the deep or multi-layer architecture of neural networks are being extensively used in these fields; for instance convolutional neural networks (CNN) as image processing tools, and recurrent neural networks (RNN) as sequence processing model.

However, in traditional sciences fields such as physics and biology, model uncertainty is crucial, especially in time series models where delay cant be tolerated. In this work, I aim to propose a novel theoretical framework and develop tools to measure uncertainty estimates, especially in deep recurrent neural networks.

This work also tackles a widely known difficulty of training recurrent neural networks, vanishing gradient by proposing a novel architecture of RNN that compute weighted average unit on past iteration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	-	Artificial Neural Network
RNN	-	Recurrent Neural Network
BBB	-	Bayes by Backprop
CNN	-	Convolutional Neural Network
KL	-	Kullback-Leibler

LIST OF SYMBOLS

γ	-	Whatever
σ	-	Whatever
ε	-	Whatever

CHAPTER 1

INTRODUCTION

1.1 Introduction: The Importance of Uncertainty

The Bayesian approach to machine learning is based on using probability to represent all forms of uncertainty. There are different models like the Gaussian process to understand possible likely and less likely options to generalize the observed data by defining the probability of distributions over functions. This observation and probabilistic models provides the confidence bounds for understanding data and making the decision based on analysis. For instance, an autonomous vehicle would use the determination from confidence bounds to whether brake or not. The confidence bounds simply means ***how certain the model is about its output?***

Understanding whether the chosen model is the right one or not, or the data has enough signals or not is an active field of research [1] in *Bayesian machine learning*, especially in *deep learning models* where based on predictions result it's difficult to make sure about the certainty level of predictions.

1.2 Introduction: A Fundamental Problem

1.3 Problem Background

Recurrent Neural Networks (RNNs) achieve state-of-the-art performance on a wide range of sequence prediction tasks ([2]; [3]; [4]; [5]; [6]). In this work we shall examine how to add uncertainty and regularisation to RNNs by means of applying Bayesian methods to training. Bayesian methods give RNNs another way to express their uncertainty (via the parameters). At the same time, by using a prior

to integrate out the parameters to average across many models during training, this gives a regularisation effect to the network. Recent approaches either attempt to justify dropout [7] and weight decay as a variational inference scheme [8], or apply Stochastic Gradient Langevin dynamics [9] to truncated backpropagation in time directly [10].

1.4 Problem Statement

Interestingly, recent work has not explored further directly apply a variational Bayes inference scheme for RNNs as was done in practical. We derive a straightforward approach based upon Bayes by Backprop [11] that we show works well on large scale problems. Our approach is a simple alteration to truncated backpropagation through time that results in an estimate of the posterior distribution on the weights of the RNN. Applying Bayesian methods to successful deep learning models affords two advantages: explicit representations of uncertainty and regularisation. Our formulation explicitly leads to a cost function with an information theoretic justification by means of a bits-back argument [12] where a KL divergence acts as a regulariser.

The form of the posterior in variational inference shapes the quality of the uncertainty estimates and hence the overall performance of the model. We shall show how performance of the RNN can be improved by means of adapting ("sharpening") the posterior locally to a batch. This sharpening adapts the variational posterior to a batch of data using gradients based upon the batch. This can be viewed as a hierarchical distribution, where a local batch gradient is used to adapt a global posterior, forming a local approximation for each batch. This gives a more flexible form to the typical assumption of Gaussian posterior when variational inference is applied to neural networks, which reduces variance. This technique can be applied more widely across other variational Bayes models.

1.5 Project Aim

The contributions of this work are as follows:

- To investigate and demonstrate how Bayes by Backprop (BBB) can be efficiently applied to Recurrent Neural Networks (RNNs).

- Develop a novel technique to reduces the variance of Bayes by Backprop (BBB).
- Improve the performance on two widely studied benchmarks with established regularisation technique such as dropout by a big margin.

1.6 Project Questions

1.7 Objective and Scope

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

2.1.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are in forefront of recent development and advances in *deep learning* by making able neural networks to deal with sequences data, which is a major shortcoming in ANN. If the data is based on sequence of events in a video or text, the traditional neural network can't do reasoning for a single event based on its previous one. To tackle this issue RNNs have loops which enables them to persist the information.

As it shown in **Figure 2.1**, a selected neural network, A takes the input x_t and outputs the value of h_t. this might not show how data goes from one step to the next one in a same network until you unroll the loop and see chain architecture of recurrent neural networks that makes them the best choice for sequential data, **Figure 2.2**.

2.1.2 Bayes by Backprop

Bayes by Backprop [11] is a variational inference scheme for learning the posterior distribution on the weights of a neural network. The posterior distribution on parameters of the network $\theta \in R^d$, $q(\theta)$ is typically taken to be a Gaussian with mean parameter $\mu \in R^d$ and standard deviation parameter $\sigma \in R^d$, denoted $\mathcal{N}(\theta|\mu, \sigma)$, noting that we use a diagonal covariance matrix, and where d is the dimensionality of the parameters of the network (typically in the order of millions). Let $\log p(y|\theta, x)$ be the log-likelihood of the neural network, then the network is trained by minimising the

variational free energy:

$$\mathcal{L}(\theta) = E_{q(\theta)}\left[\log \frac{q(\theta)}{p(y|\theta, x)p(\theta)}\right], \tag{2.1}$$

where $p(\theta)$ is a prior on the parameters.

Algorithm 1 shows the Bayes by Backprop Monte Carlo procedure for minimising 1 with respect to the mean and standard deviation parameters of the posterior $q(\theta)$.

Minimising the variational free energy 1 is equivalent to maximising the log-likelihood $\log p(y|\theta, x)$ subject to a KL complexity term on the parameters of the network that acts as a regulariser:

$$\mathcal{L}(\theta) = -E_{q(\theta)}\left[\log p(y|\theta, x)\right] + q(\theta)p(\theta). \tag{2.2}$$

In the Gaussian case with a zero mean prior, the KL term can be seen as a form of weight decay on the mean parameters, where the rate of weight decay is automatically tuned by the standard deviation parameters of the prior and posterior.

Algorithm 1 Bayes by Backprop

Sample $\epsilon \sim \mathcal{N}(0, I)$, $\epsilon \in R^d$.
Set network parameters to $\theta = \mu + \sigma\epsilon$.
Do forward propagation and backpropagation as normal.
Let g be the gradient with respect L(θ) $= E_{q(\theta)}\left[\log \frac{q(\theta)}{p(y|\theta,x)p(\theta)}\right], to\theta$ from backpropagation.
Let $g_\theta^{KL}, g_\mu^{KL}, g_\sigma^{KL}$ be the gradients of $\log \mathcal{N}(\theta|\mu, \sigma) - \log p(\theta)$ with respect to θ, μ and σ respectively.
Update μ according to the gradient $g + g_\theta^{KL} + g_\mu^{KL}$.
Update σ according to the gradient $(g + g_\theta^{KL})\epsilon + g_\sigma^{KL}$. =0

The uncertainty afforded by Bayes by Backprop trained networks has been used successfully for training feedforward models for supervised learning and to aid exploration by reinforcement learning agents [11], [**?**], [**?**], but as yet, it has not been applied to recurrent neural networks.

2.2 Backprop Through Time

The core of an RNN, f, is a neural network that maps the RNN state at step t, s_t and an input observation x_t to a new RNN state s_{t+1}, $f : (s_t, x_t) \mapsto s_{t+1}$.

For concreteness an LSTM core [**?**] has a state $s_t = (c_t, h_t)$ where c is an internal core state and h is the exposed state. Intermediate gates modulate the effect of the inputs on the outputs, namely the input gate i_t, forget gate f_t and output gate o_t. The relationship between the inputs, outputs and internal gates of an LSTM cell (without peephole connections) are as follows:

$$
\begin{aligned}
i_t &= \sigma(W_i[x_t, h_{t-1}]^T + b_i), \\
f_t &= \sigma(W_f[x_t, h_{t-1}]^T + b_f), \\
c_t &= f_t c_{t-1} + i_t \tanh(W_c[x_t, h_{t-1}] + b_c), \\
o_t &= \sigma(W_o[x_t, h_{t-1}]^T + b_o), \\
h_t &= o_t \tanh(c_t),
\end{aligned}
$$

where W_i (b_i), W_f (b_f), W_c (b_c) and W_o (b_o) are the weights (biases) affecting the input gate, forget gate, cell update, and output gate respectively.

An RNN can be trained on a sequence of length T by backpropagation through time where the RNN is unrolled T times into a feedforward network. That is, by forming the feedforward network with inputs $x_1, x_2, \ldots, x_T$ and initial state s_0:

$$
\begin{aligned}
s_1 &= f(s_0, x_1), \\
s_2 &= f(s_1, x_2), \\
&\cdots \\
s_T &= f(s_{T-1}, x_T),
\end{aligned}
\tag{2.3}
$$

where s_T is the final state of the RNN. We shall refer to an RNN core unrolled for T steps as in (2.3) by $s_{1:T} = F_T(x_{1:T}, s_0)$, where $x_{1:T}$ is the sequence of input vectors and $s_{1:T}$ is the sequence of corresponding states. Note that the truncated version of the algorithm can be seen as taking s_0 as the last state of the previous batch, s_T.

RNN parameters are learnt in much the same way as in a feedforward neural network. A loss (typically after further layers) is applied to the states $s_{1:T}$ of the RNN, and then backpropagation is used to update the weights of the network. Crucially, the

weights at each of the unrolled step are shared. Thus each weight of the RNN core receives T gradient contributions when the RNN is unrolled for T steps.

2.3 Truncated Bayes by Backprop Through Time

Applying BBB to RNNs is depicted in Figure 2.3 where the weight matrices of the RNN are drawn from a distribution (learnt by BBB). However, this direct application raises two questions: when to sample the parameters of the RNN, and how to weight the contribution of the KL regulariser of (2.2). We shall briefly justify the adaptation of BBB to RNNs, given in Algorithm 2 below.

The variational free energy of (2.2) for an RNN on a sequence of length T is:

$$\begin{aligned}\mathcal{L}(\theta) = &-E_{q(\theta)}\left[\log p(y_{1:T}|\theta, x_{1:T})\right] \\ &+ q(\theta)p(\theta),\end{aligned} \tag{2.4}$$

where $p(y_{1:T}|\theta, x_{1:T})$ is the likelihood of a sequence produced when the states of an unrolled RNN F_T are fed into an appropriate probability distribution. The parameters of the entire network are θ. Although the RNN is unrolled T times, each weight is penalised just once by the KL term, rather than T times. Also clear from (2.4) is that when a Monte Carlo approximation is taken to the expectation, the parameters θ should be held fixed throughout the entire sequence.

Two complications arise to the above naive derivation in practice: firstly, sequences are often long enough and models sufficiently large, that unrolling the RNN for the whole sequence is prohibitive. Secondly, to reduce variance in the gradients, more than one sequence is trained at a time. Thus the typical regime for training RNNs involves training on mini-batches of truncated sequences.

Let B be the number of mini-batches and C the number of truncated sequences ("cuts"), then we can write (2.4) as:

$$\begin{aligned}\mathcal{L}(\theta) = &-E_{q(\theta)}\left[\log \prod_{b=1}^{B}\prod_{c=1}^{C} p(y^{(b,c)}|\theta, x^{(b,c)})\right] \\ &+ q(\theta)p(\theta),\end{aligned} \tag{2.5}$$

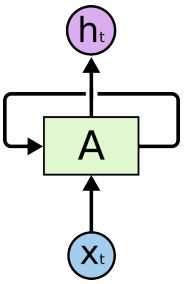

Figure 2.1: Recurrent Neural Networks (RNNs) uses loops.

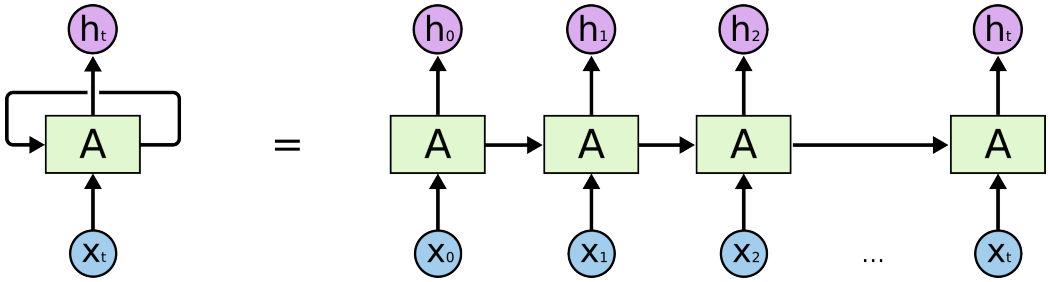

Figure 2.2: An Unrolled Recurrent Neural Networks (RNNs).

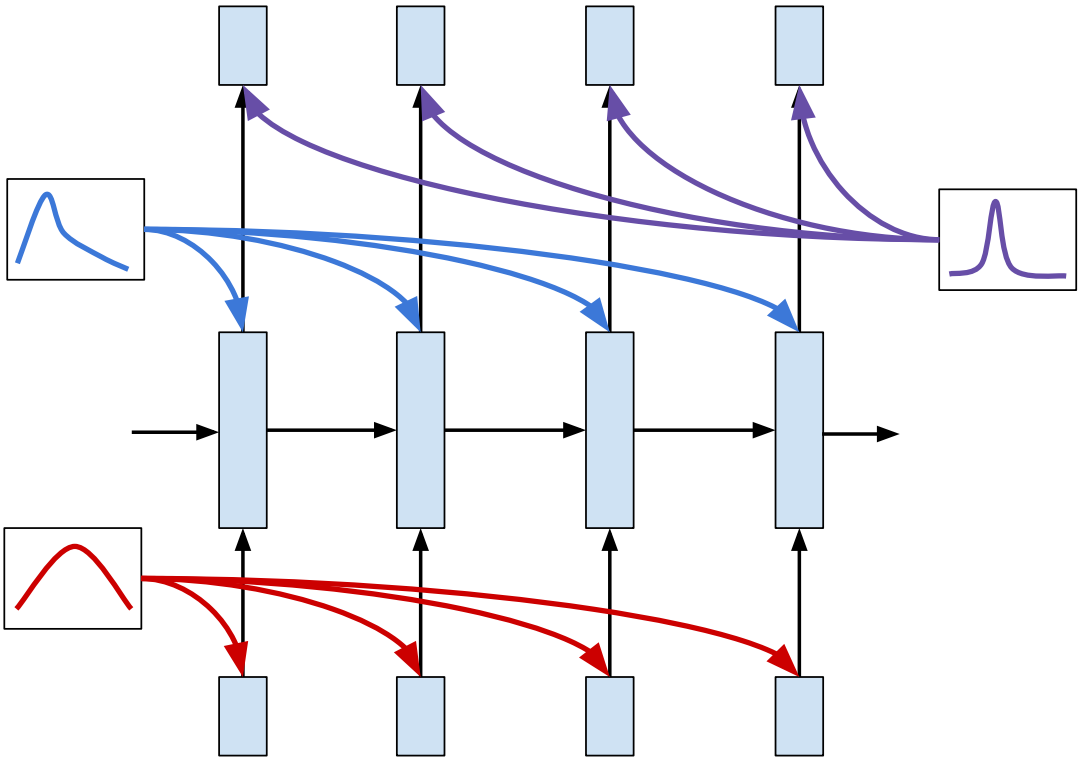

Figure 2.3: Demonstration of Bayes by Backprop (BBB) applied to an RNN.

where the (b, c) superscript denotes elements of cth truncated sequence in the bth minibatch. Thus the free energy of mini-batch b of a truncated sequence c can be written as:

$$\begin{aligned}\mathcal{L}_{(b,c)}(\theta) = &-E_{q(\theta)}\left[\log p(y^{(b,c)}|\theta, x^{(b,c)}, s^{(b,c)}_{\text{prev}})\right] \\ &+ w^{(b,c)}_{\text{KL}} q(\theta)p(\theta),\end{aligned} \tag{2.6}$$

where $w^{(b,c)}_{\text{KL}}$ distributes the responsibility of the KL cost among minibatches and truncated sequences (thus $\sum_{b=1}^{B}\sum_{c=1}^{C} w^{(b,c)}_{\text{KL}} = 1$), and $s^{(b,c)}_{\text{prev}}$ refers to the initial state of the RNN for the minibatch $x^{(b,c)}$. In practice, we pick $w^{(b,c)}_{\text{KL}} = \frac{1}{CB}$ so that the KL penalty is equally distributed among all mini-batches and truncated sequences. The truncated sequences in each subsequent mini-batches are picked in order, and so $s^{(b,c)}_{\text{prev}}$ is set to the last state of the RNN for $x^{(b,c-1)}$.

Finally, the question of when to sample weights follows naturally from taking a Monte Carlo approximations to (2.6): for each minibatch, sample a fresh set of parameters.

Algorithm 2 Bayes by Backprop for RNNs

Sample $\epsilon \sim \mathcal{N}(0, I)$, $\epsilon \in R^d$.
Set network parameters to $\theta = \mu + \sigma\epsilon$.
Sample a minibatch of truncated sequences (x, y).
Do forward propagation and backpropagation as normal on minibatch.
Let g be the gradient with respect to θ from backpropagation.
Let $g^{KL}_\theta, g^{KL}_\mu, g^{KL}_\sigma$ be the gradients of $\log \mathcal{N}(\theta|\mu, \sigma) - \log p(\theta)$ w.r.t. θ, μ and σ respectively.
Update μ according to the gradient $\frac{g+\frac{1}{C}g^{KL}_\theta}{B} + \frac{g^{KL}_\mu}{BC}$.
Update σ according to the gradient $\left(\frac{g+\frac{1}{C}g^{KL}_\theta}{B}\right)\epsilon + \frac{g^{KL}_\sigma}{BC}$. =0

2.3.1 Model Confidence

2.3.2 Model Uncertainty and Safety

2.4 State-of-the-Arts

2.5 Limitations

1. Mentor Graphics 2
   (a) item 3
2. item 4

2.6 Research Gaps

The processing at layer-5

CHAPTER 3

RESEARCH METHODOLOGY

CHAPTER 4

PROPOSED WORK

4.1 The Big Picture

4.2 Analytical Proofs

4.3 Results and Discussion

4.4 Chapter Summary

Table 4.1: Example of a table. This is a long, very long, long long, long caption. You can give a shorter caption for the "list of table" using the square braket symbol.

Temperature	Resonant Frequency	Q factor
13 mK $\pm$ 1 mK	16.93	811
40 mK $\pm$ 1 mK	16.93	817
100 mK $\pm$ 1 mK	16.93	815
300 mK $\pm$ 1 mK	16.93	806
500 mK $\pm$ 1 mK	16.93	811
800 mK $\pm$ 5 mK	16.93	814
1000 mK $\pm$ 5 mK	16.93	806

REFERENCES

1. Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature*, 2015. 521(7553): 452–459. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature14541http://10.0.4.14/nature14541.

2. Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. and Dean, J. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv e-prints*, 2016: 1–23. URL http://arxiv.org/abs/1609.08144.

3. Amodei, D., Anubhai, R., Battenberg, E., Carl, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J. and Zhu, Z. Deep-speech 2: End-to-end speech recognition in English and Mandarin. *Jmlr W&Cp*, 2015. 48: 28. ISSN 10987576. doi:10.1145/1143844.1143891.

4. Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N. and Wu, Y. Exploring the Limits of Language Modeling. *arXiv:1602.02410 [cs]*, 2016. URL http://arxiv.org/abs/1602.02410{%}5Cnhttp://www.arxiv.org/pdf/1602.02410.pdf.

5. Zaremba, W., Sutskever, I. and Vinyals, O. Recurrent Neural Network Regularization. *Iclr*, 2014. (2013): 1–8. ISSN 0157244X. doi:ng. URL http://arxiv.org/abs/1409.2329.

6. Lu, J., Xiong, C., Parikh, D. and Socher, R. Knowing When to Look: Adaptive Attention via A Visual Sentinel for Image Captioning. *1612.01887V1*, 2016. URL http://arxiv.org/abs/1612.01887.

7. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.

Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014. 15: 1929–1958. ISSN 15337928. doi: 10.1214/12-AOS1000.

8. Gal, Y. and Ghahramani, Z. Dropout as a Bayesian Approximation : Representing Model Uncertainty in Deep Learning. *Icml*, 2015. 48: 1–10.

9. Welling, M. and Teh, Y.-W. Bayesian Learning via Stochastic Gradient Langevin Dynamics. *Proceedings of the 28th International Conference on Machine Learning*, 2011: 681–688.

10. Gan, Z., Li, C., Chen, C., Pu, Y., Su, Q. and Carin, L. Scalable Bayesian Learning of Recurrent Neural Networks for Language Modeling. *arXiv preprint*, 2016.

11. Blundell, C., Cornebise, J., Kavukcuoglu, K. and Wierstra, D. Weight Uncertainty in Neural Networks. *Icml*, 2015. 37: 1613–1622. URL `http://arxiv.org/abs/1505.05424{%}5Cnhttp://www.arxiv.org/pdf/1505.05424.pdf`.

12. Hinton, G. E., Hinton, G. E., van Camp, D. and van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. *Proceedings of the sixth annual conference on Computational learning theory - COLT '93*, 1993: 5–13. doi:10.1145/168304.168306. URL `http://portal.acm.org/citation.cfm?doid=168304.168306`.