Bayesian Recurrent Neural Networks

Meire Fortunato [1] **Charles Blundell** [1] **Oriol Vinyals** [1]

Abstract

In this work we explore a straightforward variational Bayes scheme for Recurrent Neural Networks. Firstly, we show that a simple adaptation of truncated backpropagation through time can yield good quality uncertainty estimates and superior regularisation at only a small extra computational cost during training. Secondly, we demonstrate how a novel kind of posterior approximation yields further improvements to the performance of Bayesian RNNs. We incorporate local gradient information into the approximate posterior to sharpen it around the current batch statistics. This technique is not exclusive to recurrent neural networks and can be applied more widely to train Bayesian neural networks. We also empirically demonstrate how Bayesian RNNs are superior to traditional RNNs on a language modelling benchmark and an image captioning task, as well as showing how each of these methods improve our model over a variety of other schemes for training them. We also introduce a new benchmark for studying uncertainty for language models so future methods can be easily compared.

1. Introduction

Recurrent Neural Networks (RNNs) achieve state-of-the-art performance on a wide range of sequence prediction tasks (Wu et al., 2016; Amodei et al., 2015; Jozefowicz et al., 2016; Zaremba et al., 2014; Lu et al., 2016). 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 (Srivastava et al., 2014) and weight decay as a variational inference scheme (Gal & Ghahramani, 2016), or apply Stochastic Gradient Langevin dynamics (Welling & Teh, 2011, SGLD) to truncated backpropagation in time directly (Gan et al., 2016).

Interestingly, recent work has not explored further directly apply a variational Bayes inference scheme (Beal, 2003) for RNNs as was done in Graves (2011). We derive a straightforward approach based upon Bayes by Backprop (Blundell et al., 2015) 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 (Hinton & Van Camp, 1993) 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.

The contributions of our work are as follows:

- We show how Bayes by Backprop (BBB) can be efficiently applied to RNNs.
- We develop a novel technique which reduces the variance of BBB, and which can be widely adopted in other maximum likelihood frameworks.
- We improve performance on two widely studied

[1]DeepMind. Correspondence to: Meire Fortunato <meirefortunato@google.com>, Charles Blundell <cblundell@google.com>, Oriol Vinyals <vinyals@google.com>.

benchmarks outperforming established regularisation techniques such as dropout by a big margin.

- We introduce a new benchmark for studying uncertainty of language models.

The remainder of the paper is organised as follows. Section 2 and Section 3 review Bayes by Backprop and Backprop through time, respectively. Section 4 derives Bayes by Backprop for RNNs, whilst Section 5 describes posterior sharpening. We briefly review related work in Section 6. Experimental evaluation is in Section 7, and finally we conclude with a discussion in Section 8.

2. Bayes by Backprop

Bayes by Backprop (Graves, 2011; Blundell et al., 2015) is a variational inference (Wainwright et al., 2008) scheme for learning the posterior distribution on the weights of a neural network. The posterior distribution on parameters of the network $\theta \in \mathbb{R}^d$, $q(\theta)$ is typically taken to be a Gaussian with mean parameter $\mu \in \mathbb{R}^d$ and standard deviation parameter $\sigma \in \mathbb{R}^d$, 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:

$$\mathcal{L}(\theta) = \mathbb{E}_{q(\theta)} \left[\log \frac{q(\theta)}{p(y|\theta, x)p(\theta)} \right], \tag{1}$$

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:

$$\mathcal{L}(\theta) = -\mathbb{E}_{q(\theta)} \left[\log p(y|\theta, x) \right] + \mathrm{KL} \left[q(\theta) \,||\, p(\theta) \right]. \tag{2}$$

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.

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 (Blundell et al., 2015; Lipton et al., 2016; Houthooft et al., 2016), but as yet, it has not been applied to recurrent neural networks.

Algorithm 1 Bayes by Backprop

Sample $\epsilon \sim \mathcal{N}(0, I)$, $\epsilon \in \mathbb{R}^d$.
Set network parameters to $\theta = \mu + \sigma\epsilon$.
Do forward propagation and backpropagation as normal.
Let g be the gradient with respect to θ 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 $(g + g_\theta^{KL})\epsilon + g_\sigma^{KL}$.

3. 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 (Hochreiter & Schmidhuber, 1997) 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:

$$\begin{aligned}
s_1 &= f(s_0, x_1), \\
s_2 &= f(s_1, x_2), \\
&\cdots \\
s_T &= f(s_{T-1}, x_T),
\end{aligned} \tag{3}$$

where s_T is the final state of the RNN. We shall refer to an RNN core unrolled for T steps as in (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.

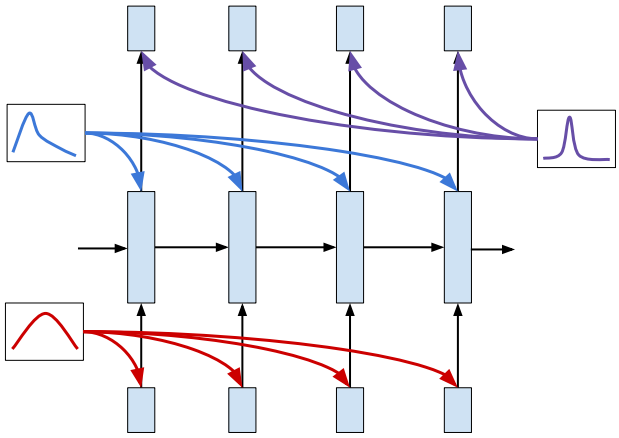


Figure 1. Illustration of BBB applied to an RNN.

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.

4. Truncated Bayes by Backprop Through Time

Applying BBB to RNNs is depicted in Figure 1 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). We shall briefly justify the adaptation of BBB to RNNs, given in Algorithm 2 below.

The variational free energy of (2) for an RNN on a sequence of length T is:

$$\mathcal{L}(\theta) = -\mathbb{E}_{q(\theta)}\left[\log p(y_{1:T}|\theta, x_{1:T})\right] + \mathrm{KL}\left[q(\theta)\,||\,p(\theta)\right], \tag{4}$$

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 (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 (4) as:

$$\mathcal{L}(\theta) = -\mathbb{E}_{q(\theta)}\left[\log \prod_{b=1}^{B}\prod_{c=1}^{C} p(y^{(b,c)}|\theta, x^{(b,c)})\right] + \mathrm{KL}\left[q(\theta)\,||\,p(\theta)\right], \tag{5}$$

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:

$$\mathcal{L}_{(b,c)}(\theta) = -\mathbb{E}_{q(\theta)}\left[\log p(y^{(b,c)}|\theta, x^{(b,c)}, s^{(b,c)}_{\text{prev}})\right] + w^{(b,c)}_{\text{KL}}\mathrm{KL}\left[q(\theta)\,||\,p(\theta)\right], \tag{6}$$

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 (6): for each minibatch, sample a fresh set of parameters.

Algorithm 2 Bayes by Backprop for RNNs

Sample $\epsilon \sim \mathcal{N}(0, I)$, $\epsilon \in \mathbb{R}^d$.
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 $\left(\frac{g+\frac{1}{C}g^{KL}_\theta}{B}\right)\epsilon + \frac{g^{KL}_\sigma}{BC}$.

5. Posterior Sharpening

The choice of variational posterior $q(\theta)$ as described in Section 4 can be enhanced by adding side information that

makes the posterior over the parameters more accurate, thus reducing variance of the learning process. This "sharpened" posterior yields more stable optimisation, a common pitfall of Bayesian approaches to train neural networks. We chose the side information to be the minibatch of data (inputs and targets) (x, y) sampled from the training set. This has the advantage that weights are sampled per minibatch, so that matrix-matrix operations can still be carried. In this section we explain how we construct such $q(\theta|(x, y))$.

A challenging aspect of modelling the variational posterior $q(\theta|(x, y))$ is the large number of dimensions of $\theta \in \mathbb{R}^d$. When the dimensionality is not in the order of millions, a powerful non-linear function (such as a neural network) can be used which transforms observations (x, y) to the parameters of a Gaussian distribution, as proposed in (Kingma & Welling, 2013; Rezende et al., 2014). Unfortunately, this neural network would have far too many parameters, making this approach unfeasible.

Given that the loss $-\log p(y|\theta, x)$ is differentiable with respect to θ, we propose to parameterise q as a linear combination of θ and $g_\theta = -\nabla_\theta \log p(y|\theta, x)$, both d-dimensional vectors. Intuitively, we are applying a correction which follows the gradient of the loss and which should, indeed, be helpful to minimise this loss.

Thus, we can define a hierarchical posterior of the form

$$q(\theta|(x, y)) = \int q(\theta|\varphi, (x, y))q(\varphi)d\varphi, \tag{7}$$

with $\mu, \sigma \in \mathbb{R}^d$, and $q(\varphi) = \mathcal{N}(\varphi|\mu, \sigma)$ – the same as in the standard BBB method. Finally, let $*$ denote element-wise multiplication, we then have

$$q(\theta|\varphi, (x, y)) = \mathcal{N}(\theta|\varphi - \eta * g_\varphi, \sigma_0^2 I), \tag{8}$$

where $\eta \in \mathbb{R}^d$ is a free parameter to be learnt and σ_0 a scalar hyper-parameter of our model. η can be interpreted as a per-parameter learning rate.

During training, we get $\theta \sim q(\theta|(x, y))$ via ancestral sampling to optimise the loss

$$L(\mu, \sigma, \eta) = E_{(x,y)}[E_{q(\varphi)q(\theta|\varphi,(x,y))}[L(x, y, \theta, \varphi|\mu, \sigma, \eta)]], \tag{9}$$

with $L(x, y, \theta, \varphi|\mu, \sigma, \eta)$ given by

$$\begin{aligned} L(x, y, \theta, \varphi|\mu, \sigma, \eta) \quad = \quad & -\log p(y|\theta, x) + \\ & \mathrm{KL}\left[q(\theta|\varphi, (x, y)) \;||\; p(\theta|\varphi)\right] + \\ & \frac{1}{C}\mathrm{KL}\left[q(\varphi) \;||\; p(\varphi)\right], \end{aligned}$$

where μ, σ, η are our model parameters, and p are the priors for the distributions defining q (for exact details of these distributions see Section 7). The bound on the true data likelihood which yields eq. (9) is derived in the supplementary material. Algorithm 3 presents how learning is performed in practice.

Unlike regular BBB where the KL terms can be ignored during inference (see supplementary material), there are two options for doing inference under posterior sharpening. The first involves using $q(\varphi)$ and ignoring any KL terms, similar to regular BBB. The second involves using $q(\theta|(x, y))$ which requires using the term $\mathrm{KL}\left[q(\theta|\varphi, (x, y)) \;||\; p(\theta|\varphi)\right]$ yielding an upper bound on perplexity (lower bound in log probability; see supplementary material for details). A comparison of the two inference methods is provided in the next section.

Algorithm 3 BBB with Posterior Sharpening

Sample a minibatch (x, y) of truncated sequences.
Sample $\varphi \sim q(\varphi) = \mathcal{N}(\varphi|\mu, \sigma)$.
Let $g_\varphi = -\nabla_\varphi \log p(y|\varphi, x)$.
Sample $\theta \sim q(\theta|\varphi, (x, y)) = \mathcal{N}(\theta|\varphi - \eta * g_\varphi, \sigma_0^2 I)$.
Compute the gradients of eq. (9) w.r.t. (μ, σ, η).
Update (μ, σ, η).

6. Related work

We note that the procedure of sharpening the posterior as explained above has similarities with other techniques. Perhaps the most obvious one is line search: indeed, η is a trained parameter that does line search along the gradient direction. Probabilistic interpretations have been given to line search in e.g. Mahsereci & Hennig (2015), but ours is the first that uses a variational posterior with the reparametrization trick/perturbation analysis gradient. Also, the probabilistic treatment to line search can also be interpreted as a trust region method.

Another related technique is dynamic evaluation (Mikolov et al., 2010), which trains an RNN during evaluation of the model with a fixed learning rate. The update applied in this case is cumulative, and only uses previously seen data. Thus, they can take a purely deterministic approach and ignore any KL between a posterior with privileged information and a prior. As we will show in Section 7, performance gains can be significant as the data exhibits many short term correlations.

Lastly, learning to optimise (or learning to learn) (Li & Malik, 2016; Andrychowicz et al., 2016) is related in that a learning rate is learned so that it produces better updates than those provided by e.g. AdaGrad (Duchi et al., 2011) or Adam (Kingma & Ba, 2014). Whilst they train a parametric model, we treat these as free parameters (so that they can adapt more quickly to the non-stationary distribution w.r.t. parameters). Notably, we use gradient information to inform a variational posterior so as to reduce variance

of Bayesian Neural Networks. Thus, although similar in flavour, the underlying motivations are quite different.

Applying Bayesian methods to neural networks has a long history, with most common approximations having been tried. Buntine & Weigend (1991) propose various maximum a posteriori schemes for neural networks, including an approximate posterior centered at the mode. Buntine & Weigend (1991) also suggest using second order derivatives in the prior to encourage smoothness of the resulting network. Hinton & Van Camp (1993) proposed using variational methods for compressing the weights of neural networks as a regulariser. Hochreiter et al. (1995) suggest an MDL loss for single layer networks that penalises non-robust weights by means of an approximate penalty based upon perturbations of the weights on the outputs. MacKay (1995) investigated using the Laplace approximation for capturing the posterior of neural networks. Neal (2012) investigated the use of hybrid Monte Carlo for training neural networks, although it has so far been difficult to apply these to the large sizes of networks considered here.

More recently Graves (2011) derived a variational inference scheme for neural networks and Blundell et al. (2015) extended this with an update for the variance that is unbiased and simpler to compute. Graves (2016) derives a similar algorithm in the case of a mixture posterior. Several authors have claimed that dropout (Srivastava et al., 2014) and Gaussian dropout (Wang & Manning, 2013) can be viewed as approximate variational inference schemes (Gal & Ghahramani, 2015; Kingma et al., 2015, respectively). Gal & Ghahramani (2016) goes a step further and uses Monte Carlo dropout for LSTMs (we compare to this results in our experiments). Variational methods typically underestimate the uncertainty in the posterior (as they are mode seeking, akin to the Laplace approximation), whereas expectation propagation methods are mode averaging and so tend to overestimate uncertainty. Nonetheless, several papers explore applying expectation propagation to neural networks: Soudry et al. (2014) derive a closed form approximate online expectation propagation algorithm, whereas Hernández-Lobato & Adams (2015) proposed using multiple passes of assumed density filtering (in combination with early stopping) attaining good performance on a number of small data sets. Hasenclever et al. (2015) derive a distributed expectation propagation scheme with SGLD (Welling & Teh, 2011) as an inner loop. Others have also considered applying SGLD to neural networks (Li et al., 2015) and Gan et al. (2016) more recently used SGLD for LSTMs (we compare to these results in our experiments).

7. Experiments

We present the results of our method for a language modelling benchmark and an image caption generation task.

7.1. Language Modelling

We evaluated our model on the Penn Treebank (Marcus et al., 1993) benchmark, a task consisting on next word prediction. We used the network architecture from Zaremba et al. (2014), a simple yet strong baseline on this task, and for which there is an open source implementation[1]. The baseline consists of an RNN with LSTM cells and a special regularisation technique, where the dropout operator is only applied to the non-recurrent connections. We keep the network configuration unchanged, but instead of using dropout we apply our Bayes by Backprop formulation. Our goal is to demonstrate the effect of applying BBB to a pre-existing, well studied architecture.

To train our models, we tuned the parameters on the prior distribution, the learning rate and its decay. The weights were initialised randomly and we used gradient descent with gradient clipping for optimisation, closely following Zaremba et al. (2014)'s "medium" recipe.

As in Blundell et al. (2015), the prior of the network weights θ was taken to be a scalar mixture of two Gaussian densities with zero mean and variances σ_1^2 and σ_2^2.

$$p(\theta) = \prod_j \left(\pi \mathcal{N}(\theta_j|0,\sigma_1^2) + (1-\pi)\mathcal{N}(\theta_j|0,\sigma_2^2)\right), \quad (10)$$

where θ_j is the j-th weight of the network. We searched $\pi \in \{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$, $\log \sigma_1 \in \{0, -1, -2\}$ and $\log \sigma_2 \in \{-6, -7, -8\}$.

For speed purposes, during training we used one sample from the posterior for estimating the gradients and computing the (approximate) KL-divergence. For inference, we experimented with either computing the expected loss via Monte Carlo sampling, or using the mean of the posterior distribution as the parameters of the network (MAP estimate). We observed that the results improved as we increased the number of samples but they were not significantly better than taking the mean (as was also reported by (Graves, 2011; Blundell et al., 2015)). For convenience, in Table 1 we report our numbers using the mean of the converged distribution, as there is no computation overhead w.r.t. a standard LSTM model.

In Table 1, we report results for the medium LSTM configuration (2 layers with 650 units each) from Zaremba et al. (2014), which we refer to as "LSTM dropout". We also

[1] https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py

Table 1. Word-level perplexity on the Penn Treebank language modelling task (lower is better).

Model (medium)	Validation	Test
LSTM	120.7	114.5
LSTM dropout	86.2	82.1
SGLD	N/A	109.1
SGLD dropout	N/A	99.6
Variational LSTM (tied weights)	81.8	79.7
Variational LSTM (tied weights, MS)	N/A	79.0
Bayes by Backprop (BBB)	78.8	75.5
BBB with Posterior Sharpening	$\leq$ **77.8**	$\leq$ **74.8**
Model (medium) with Dynamic Evaluation	**Validation**	**Test**
LSTM dropout	79.7	77.1
Bayes by Backprop (BBB)	73.4	70.7
BBB with Posterior Sharpening	$\leq$ **72.6**	$\leq$ **69.8**

included the numbers for two other Bayesian approaches: Variational LSTMs by Gal & Ghahramani (2016) and Stochastic Gradient Langevin dynamics (SGLD) by Gan et al. (2016). Finally, we added dynamic evaluation (Mikolov et al., 2010) results with a learning rate of 0.1, which was found via cross validation.

As with other VAE-related RNNs (Fabius & van Amersfoort, 2014; Bayer & Osendorfer, 2014; Chung et al., 2015) perplexities using posterior sharpening are reported including a KL penalty $\mathrm{KL}\left[q(\theta|\varphi,(x,y))\;||\;p(\theta|\varphi)\right]$ in the log likelihood term (the KL is computed exactly, not sampled). For posterior sharpening we use a hierarchical prior for θ: $p(\theta|\varphi) = \mathcal{N}(\theta|\varphi,\sigma_0^2 I)$ which expresses our belief that a priori, the network parameters θ will be much like the data independent parameters φ with some small Gaussian perturbation. In our experiments we swept over σ_0 on the validation set, and found $\sigma_0 = 0.02$ to perform well, although results were not particularly sensitive to this. Note that with posterior sharpening, the perplexities reported are upper bounds (as the likelihoods are lower bounds).

Lastly, we tested the variance reduction capabilities of posterior sharpening by analysing the perplexity attained by the best models reported in Table 1. Standard BBB yields 258 perplexity after only one epoch, whereas the model with posterior sharpening is better at 227.

Lower perplexities on the Penn Treebank task can be achieved by varying the model architecture, which should be complementary to our work of treating weights as random variables—we are simply interested in assessing the impact of our method on an existing architecture, rather than absolute state-of-the-art. See Kim et al. (2015); Zilly et al. (2016); Merity et al. (2016), for a report on recent advances on this benchmark, where they achieve perplexities of 70.9 on the test set.

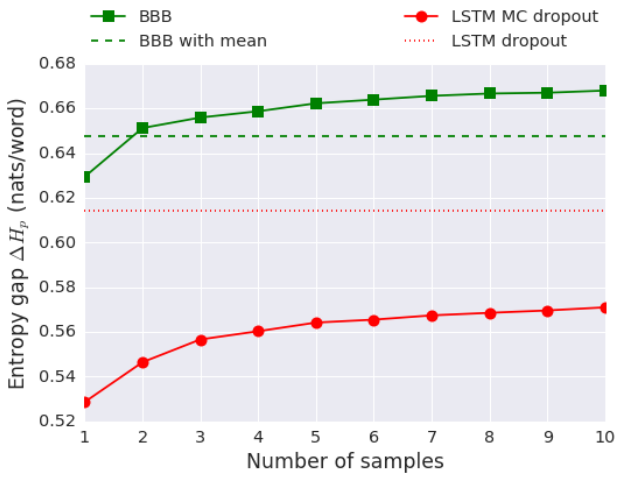


Figure 2. Entropy gap ΔH_p (eq. (12)) between reversed and regular Penn Treebank test sets $\times$ number of samples.

7.1.1. Uncertainty Analysis

We used the Penn Treebank test set, which is a long sequence of $\approx$ 80K words, and reversed it. Thus, the "reversed" test set first few words are: "us with here them see not may we ..." which correspond to the last words of the standard test set: "... we may not see them here with us".

Let V be the vocabulary of a task. For a given input sequence $x = x_{1:T}$ and a probabilistic model p, we define the entropy of x under p, $H_p[x]$, by

$$H_p[x_i] = \sum_{w\in V} p(w|x_{1:i-1}) \log \frac{1}{p(w|x_{1:i-1})}$$
$$H_p[x] = \sum_{i=1}^{T} H_p[x_i]. \tag{11}$$

Let $\frac{1}{T}H_p[X] = \overline{H}_p[X]$, i.e., the per word entropy. Let X be the standard Penn Treebank test set, and X_{rev} the reversed one. For a given probabilistic model p, we define the entropy gap ΔH_p by

$$\Delta H_p = \overline{H}_p[X_{\text{rev}}] - \overline{H}_p[X]. \tag{12}$$

Since X_{rev} clearly does not come from the training data distribution (reversed English does not look like proper English), we expect ΔH_p to be positive and large. Namely, if we take the per word entropy of a model as a proxy for the models' certainty (low entropy means the model is confident about its prediction), then the overall certainty of well calibrated models over X_{rev} should be lower than over X. Thus, $\overline{H}_p[X_{\text{rev}}] > \overline{H}_p[X]$. When comparing two distributions, we expect the better calibrated one to have a larger ΔH_p.

In Figure 2, we plotted ΔH_p for the BBB and the baseline dropout LSTM model. The BBB model has a gap of about

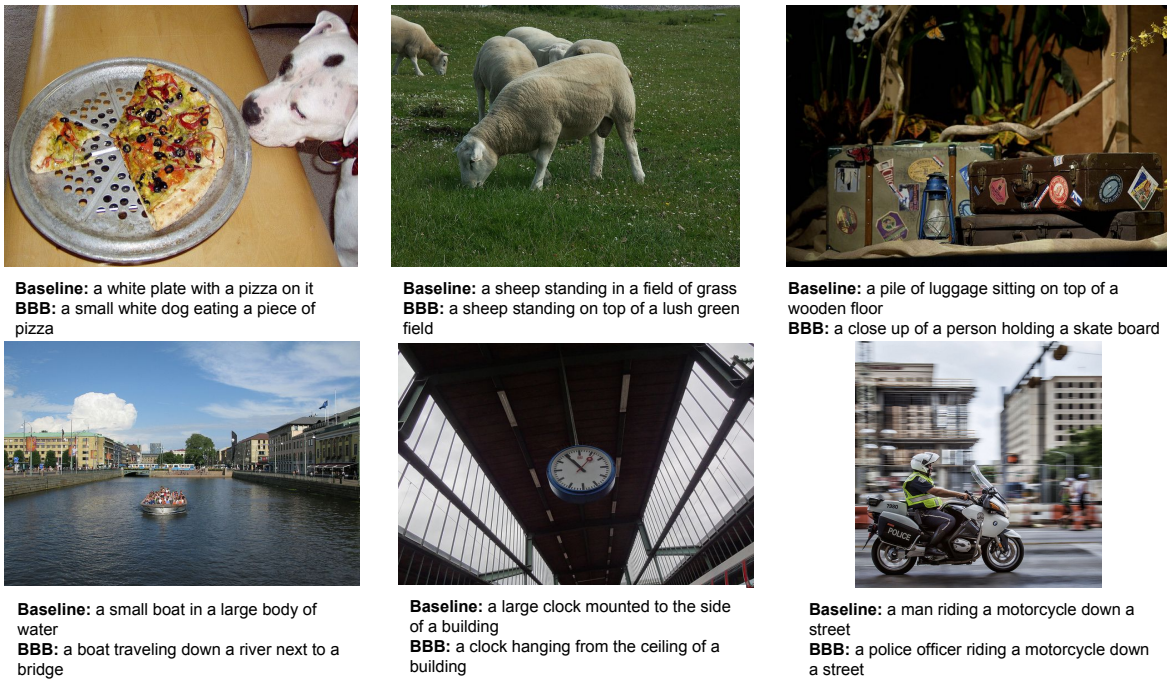


Figure 3. Image captioning results on MSCOCO development set.

0.67 nats/word when taking 10 samples, and slightly below 0.65 when using the posterior mean. In contrast, the model using MC Dropout (Gal & Ghahramani, 2015) is less well calibrated and is below 0.58 nats/word. However, when "turning off" dropout (i.e., using the mean field approximation), ΔH_p improves to below 0.62 nats/word.

We also note that the absolute entropy of both models is nowhere near uniform, so future research should strive to improve further neural networks uncertainty estimates. For instance, the BBB mean has entropy of 4.48 on the reversed set, which is still far below 9.2 nats/word (uniform).

7.2. Image Caption Generation

We also applied Bayes by Backprop for RNNs to image captioning. Our experiments were based upon the model described in Vinyals et al. (2016), where a state-of-the-art pre-trained convolutional neural network (CNN) was used to map an image to a high dimensional space, and this representation was taken to be the initial state of an LSTM. The LSTM model was trained to predict the next word on a sentence conditioned on the image representation and all the previous words in the image caption. We kept the CNN architecture unchanged, and used an LSTM trained using Bayes by Backprop rather than the traditional LSTM with dropout regularisation. As in the case for language modelling, this work conveniently provides an open source implementation[2].

We used the same prior distribution on the weights of the network (10) as we did for the language modelling task, and searched over the same hyper-parameters.

We used the MSCOCO (Lin et al., 2014) data set and report perplexity, BLUE-4, and CIDER scores on Table 2 compared to the Show and Tell model (Vinyals et al., 2016), which was the winning entry of the captioning challenge in 2015[3]. We observe significant improvements in BLUE and CIDER, outperforming the dropout baseline by a large margin. Moreover, a random sample of the captions that were different for both the baseline and BBB is shown in Figure 3. Besides the clear quantitative improvement, it is useful to visualise qualitatively the performance of BBB, which indeed generally outperforms the strong baseline, winning in most cases.

As in the case of Penn Treebank, we chose a performant, open source model. Captioning models that use spatial at-

Table 2. Image captioning results on MSCOCO development set.

Model	Perplexity	BLUE-4	CIDER
Show and Tell	8.3	28.8	89.8
Bayes by Backprop (BBB)	**8.1**	**30.2**	**96.0**

[2] https://github.com/tensorflow/models/tree/master/im2txt

[3] The winning entry was an ensemble of many models, including some with fine tuning w.r.t. the image model. In this paper, though, we report single model performance.

tention, combined with losses that optimise CIDER directly (rather than a surrogate loss as we do) achieve over 100 CIDER points (Lu et al., 2016; Liu et al., 2016).

8. Discussion

We have shown how to adapt Bayes by Backprop to Recurrent Neural Networks and then enhanced it further by introducing the idea of posterior sharpening: a hierarchical posterior on the weights of neural networks that are parameterised by a gradient of the model. Posterior sharpening allows a neural network to adapt locally to batches of data and we have demonstrated that this helps on several challenging domains involving sizeable RNNs: language modelling and image captioning. We show how two open source, widely available models can be improved by adding BBB. We demonstrated that not only do BBB RNNs often have superior performance to their corresponding baseline model, BBB RNNs are also better regularised and have superior uncertainty properties in terms of uncertainty on out-of-distribution data. Furthermore, BBB RNNs through their uncertainty estimates show signs of knowing what they know, and when they do not, a critical property for many real world applications such as self-driving cars, healthcare, game playing, and robotics.

Everything from our work can be applied on top of other enhancements to RNN/LSTM models (and other non-recurrent architectures), and the empirical evidence combined with improvements such as posterior sharpening makes variational Bayes methods look very promising. We are exploring further research directions and wider adoption of the techniques presented in our work.

Acknowledgements

We would like to thank Grzegorz Swirszcz, Daan Wierstra, Vinod Gopal Nair, Koray Kavukcuoglu, Chris Shallue, James Martens, Danilo J. Rezende, James Kirkpatrick, Alex Graves, Jacob Menick, Yori Zwols, Frederick Besse and many others at DeepMind for insightful discussions and feedback on this work.

References

Amodei, Dario, Anubhai, Rishita, Battenberg, Eric, Case, Carl, Casper, Jared, Catanzaro, Bryan, Chen, Jingdong, Chrzanowski, Mike, Coates, Adam, Diamos, Greg, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Andrychowicz, Marcin, Denil, Misha, Gomez, Sergio, Hoffman, Matthew W, Pfau, David, Schaul, Tom, and de Freitas, Nando. Learning to learn by gradient descent by gradient descent. number, pp. 3981–3989, 2016.

Bayer, Justin and Osendorfer, Christian. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Beal, Matthew James. University of London United Kingdom, 2003.

Blundell, Charles, Cornebise, Julien, Kavukcuoglu, Koray, and Wierstra, Daan. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Buntine, Wray L and Weigend, Andreas S. Bayesian backpropagation. *Complex systems*, 5(6):603–643, 1991.

Chung, Junyoung, Kastner, Kyle, Dinh, Laurent, Goel, Kratarth, Courville, Aaron C, and Bengio, Yoshua. A recurrent latent variable model for sequential data. number, pp. 2980–2988, 2015.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Fabius, Otto and van Amersfoort, Joost R. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014.

Gal, Yarin and Ghahramani, Zoubin. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2, 2015.

Gal, Yarin and Ghahramani, Zoubin. A theoretically grounded application of dropout in recurrent neural networks. number, pp. 1019–1027, 2016.

Gan, Zhe, Li, Chunyuan, Chen, Changyou, Pu, Yunchen, Su, Qinliang, and Carin, Lawrence. Scalable bayesian learning of recurrent neural networks for language modeling. *arXiv preprint arXiv:1611.08034*, 2016.

Graves, Alex. Practical variational inference for neural networks. number, pp. 2348–2356, 2011.

Graves, Alex. Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*, 2016.

Hasenclever, Leonard, Lienart, Thibaut, Vollmer, Sebastian, Webb, Stefan, Lakshminarayanan, Balaji, Blundell, Charles, and Whye Teh, Yee. Distributed bayesian learning with stochastic natural-gradient expectation propagation and the posterior server. *arXiv preprint arXiv:1512.09327*, 2015.

Hernández-Lobato, José Miguel and Adams, Ryan. Probabilistic backpropagation for scalable learning of bayesian neural networks. number, pp. 1861–1869, 2015.

Hinton, Geoffrey E and Van Camp, Drew. Keeping the neural networks simple by minimizing the description length of the weights. number, pp. 5–13. ACM, 1993.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hochreiter, Sepp, Schmidhuber, Jürgen, et al. Simplifying neural nets by discovering flat minima. *Advances in Neural Information Processing Systems*, pp. 529–536, 1995.

Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, De Turck, Filip, and Abbeel, Pieter. Curiositydriven exploration in deep reinforcement learning via bayesian neural networks. *arXiv preprint arXiv:1605.09674*, 2016.

Jozefowicz, Rafal, Vinyals, Oriol, Schuster, Mike, Shazeer, Noam, and Wu, Yonghui. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, Diederik P and Welling, Max. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kingma, Diederik P, Salimans, Tim, and Welling, Max. Variational dropout and the local reparameterization trick. number, pp. 2575–2583, 2015.

Li, Chunyuan, Chen, Changyou, Carlson, David, and Carin, Lawrence. Preconditioned stochastic gradient langevin dynamics for deep neural networks. *arXiv preprint arXiv:1512.07666*, 2015.

Li, Ke and Malik, Jitendra. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Lin, Tsung-Yi, Maire, Michael, Belongie, Serge, Hays, James, Perona, Pietro, Ramanan, Deva, Dollár, Piotr, and Zitnick, C Lawrence. Microsoft coco: Common objects in context. number, pp. 740–755. Springer, 2014.

Lipton, Zachary C, Gao, Jianfeng, Li, Lihong, Li, Xiujun, Ahmed, Faisal, and Deng, Li. Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081*, 2016.

Liu, Siqi, Zhu, Zhenhai, Ye, Ning, Guadarrama, Sergio, and Murphy, Kevin. Optimization of image description metrics using policy gradient methods. *arXiv preprint arXiv:1612.00370*, 2016.

Lu, Jiasen, Xiong, Caiming, Parikh, Devi, and Socher, Richard. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. *arXiv preprint arXiv:1612.01887*, 2016.

MacKay, David JC. Probable networks and plausible predictionsa review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.

Mahsereci, Maren and Hennig, Philipp. Probabilistic line searches for stochastic optimization. number, pp. 181–189, 2015.

Marcus, Mitchell P, Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Merity, Stephen, Xiong, Caiming, Bradbury, James, and Socher, Richard. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mikolov, Tomas, Karafiát, Martin, Burget, Lukas, Cernockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *Interspeech*, pp. 3, 2010.

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

Rezende, Danilo Jimenez, Mohamed, Shakir, and Wierstra, Daan. Stochastic backpropagation and approximate inference in deep generative models. number, pp. 1278–1286, 2014.

Soudry, Daniel, Hubara, Itay, and Meir, Ron. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. number, pp. 963–971, 2014.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Vinyals, Oriol, Toshev, Alexander, Bengio, Samy, and Erhan, Dumitru. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE transactions on pattern analysis and machine intelligence*, 2016.

Wainwright, Martin J, Jordan, Michael I, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2): 1–305, 2008.

Wang, Sida I and Manning, Christopher D. Fast dropout training. number, pp. 118–126, 2013.

Welling, Max and Teh, Yee W. Bayesian learning via stochastic gradient langevin dynamics. number, pp. 681–688, 2011.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V, Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

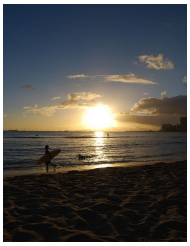
Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Zilly, Julian Georg, Srivastava, Rupesh Kumar, Koutník, Jan, and Schmidhuber, Jürgen. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.

A. Supplementary Material

A.1. Additional Captioning examples

Baseline: a man riding a skateboard up the side of a ramp
BBB: a man flying through the air while riding a skateboard



Baseline: a person walking on the beach with a surfboard
BBB: a person with a surfboard on a beach

Baseline: a red double decker bus parked in a parking lot
BBB: a red double decker bus driving down a street

Baseline: a man is cooking in a large kitchen
BBB: a man is putting something in the oven

Figure 4. Additional Captions from MS COCO validation set.

A.2. Derivation of Free Energy for Posterior Sharpening

Here we turn to deriving the training loss function we use for posterior sharpening. The basic idea is to take a variational approximation to the marginal likelihood $p(x)$ that factorises hierarchically. In particular, we shall assume a hierarchical prior for the parameters such that $p(x) = \int p(x|\theta)p(\theta|\varphi)p(\varphi)d\theta d\varphi$. Then we pick a variational posterior that conditions upon x, and factorises as $q(\theta, \varphi|x) = q(\theta|\varphi, x)q(\varphi)$. The expected lower bound on $p(x)$ is then as follows:

$$\log p(x) \tag{13}$$

$$= \log \left(\int p(x|\theta)p(\theta|\varphi)p(\varphi)d\theta d\varphi \right) \tag{14}$$

$$\geq \mathbb{E}_{q(\varphi,\theta|x)} \left[\log \frac{p(x|\theta)p(\theta|\varphi)p(\varphi)}{q(\varphi,\theta|x)} \right] \tag{15}$$

$$= \mathbb{E}_{q(\theta|\varphi,x)q(\varphi)} \left[\log \frac{p(x|\theta)p(\theta|\varphi)p(\varphi)}{q(\theta|\varphi,x)q(\varphi)} \right] \tag{16}$$

$$= \mathbb{E}_{q(\varphi)} \left[\mathbb{E}_{q(\theta|\varphi,x)} \left[\log p(x|\theta) + \log \frac{p(\theta|\varphi)}{q(\theta|\varphi,x)} \right] + \log \frac{p(\varphi)}{q(\varphi)} \right] \tag{17}$$

$$= \mathbb{E}_{q(\varphi)} \left[\mathbb{E}_{q(\theta|\varphi,x)} \left[\log p(x|\theta) \right] - \mathrm{KL} \left[q(\theta|\varphi,x) \,||\, p(\theta|\varphi) \right] \right] - \mathrm{KL} \left[q(\varphi) \,||\, p(\varphi) \right] \tag{18}$$

A.3. Derivation of Predictions with Posterior Sharpening

Now we consider making predictions. These are done by means of Bayesian model averaging over the approximate posterior. In the case of no posterior sharpening, predictions are made by evaluating: $\mathbb{E}_{q(\theta)} \left[\log p(\hat{x}|\theta) \right]$. For posterior sharpening, we derive a bound on a Bayesian model average over the approximate posterior of φ:

$$\mathbb{E}_{q(\varphi)} \left[\log p(\hat{x}|\varphi) \right]$$

$$= \mathbb{E}_{q(\varphi)} \left[\log \int p(\hat{x}|\theta)p(\theta|\varphi)d\theta \right] \tag{19}$$

$$\geq \mathbb{E}_{q(\varphi)} \left[\mathbb{E}_{q(\theta|\varphi,x)} \left[\log \frac{p(\hat{x}|\theta)p(\theta|\varphi)}{q(\theta|\varphi,x)} \right] \right] \tag{20}$$

$$= \mathbb{E}_{q(\varphi)} \left[\mathbb{E}_{q(\theta|\varphi,x)} \left[\log p(\hat{x}|\theta) \right] - \mathrm{KL} \left[q(\theta|\varphi,x) \,||\, p(\theta|\varphi) \right] \right] \tag{21}$$