# Active Learning with Bayesian Neural Networks

**Amit Gamane** A0119264E
**Ian Ngiaw** A0121437N
**Martynas Grigonis** A0174684R
**Wu Wenqi** A0124278A
**Xu Bili** A0124368A

"True wisdom is knowing what you don't know."

Confucius

## Introduction

Supervised learning methods such as deep learning require large amounts of data during the training phase to provide robust prediction accuracy. Additionally, labelling of such large datasets is often a costly process. In this project, we explore using active learning to select the next batch of data to be labelled and trained on. We show that by carefully selecting what data to train on, our model is just as performant as ordinary neural networks models, while being trained on a significantly smaller set of data.

Active learning is an iterative process of choosing the most informative (high entropic) samples from a population of unlabelled data. This is equivalent to selecting samples which the model is most uncertain about predicting, by observing the model's prediction distribution over the unlabelled data. Training over actively selected data samples allows the model to learn faster with less training data.

Ordinary neural networks are unable to represent uncertainty in their predictions. While it is intuitive to think of the softmax outputs as a probability distribution, this is insufficient in giving us the model uncertainty (Gal, 2016). We have to turn to Bayesian neural network models to get an accurate probabilistic interpretation of our predictions.

Besides being able to actively select optimal data samples to train on, there are many other advantages to having a Bayesian model. While ordinary neural networks are prone to overfitting and tend to make overly confident decisions about the correct class, this does not seem to happen to Bayesian models. Knowing the model's certainty (or uncertainty) about its decision allows the practitioner to decide whether to go ahead with the model's decision, rather than having blind faith. The model uncertainty serves as a deterrent to what could otherwise be a detrimental decision, and can be very useful in real world applications, such as autonomous piloting.

To assess the viability of various active learning methods for Bayesian neural networks, we trained a Bayesian dropout convolutional neural network (CNN) using a reduced-size MNIST dataset that was actively selected, and benchmarked its performance against an ordinary CNN.

The remaining of this report is organized as follows: First, we will give a detailed introduction to Bayesian neural networks, elaborating on our model implementations and highlighting various active learning methods which we tried. We then proceed to discuss our findings from our experiments. Finally, we go beyond active learning and explore possible augmentations to our Bayesian model that could possibly improve its performance.

## Bayesian Neural Networks

A Bayesian neural network is one which has a prior distribution $p(w)$ defined over its weights. By conditioning on the training data $\{X, Y\}$, where $X$ and $Y$ are the inputs and observed outputs respectively, the neural network is able to update its prior to get the posterior $p(w|X, Y)$. We can then perform a prediction, $y^*$, for an unobserved input $x^*$ via the model. This is given by:

$$p(y^*|x^*, X, Y) = \int p(y^*|x^*, w)p(w|X, Y)dw \quad (1)$$

This is similar to Gaussian Processes, where a prediction is done based on the posterior distribution. In fact, a neural network containing a single hidden layer comprising of infinite hidden units with independent zero-mean Gaussian priors over the weights converges to a Gaussian Process (Neal, 1995, Chapter 2). Particularly, given a limit of infinite width, the Central Limit Theorem will imply that the function computed in the final hidden layer in the neural network is equivalent to a function that is drawn from a GP. (Neal, 1995; Lee et. al, 2017).

When it comes to a finite number of weights, we can still model uncertainity by placing a prior distribution over the weights (MacKay, 1992; Neal, 1995). There is a correspondence between the addition of more hidden units and a GP modelling infinite basis functions. Thus, we may intuitively perceive a hidden layer with finite dimensions to approximate a Gaussian Process (Neal, 1995). In Equation 1, $p(w|X, Y)$ is computationally intractable due to the size of the training dataset.

Instead, we can use variational Bayesian methods to approximate the posterior to a simpler parameterized distribution $q_\theta(w)$ which is easier to compute.

$$q_\theta(y^*|x^*) = \int p(y^*|x^*, w)q_\theta(w)dw \quad (2)$$

To determine the suitability of our approximation for $p(w|X,Y)$ with $q_\theta(w)$, we can calculate the Kullback-Leibler (KL) divergence between the two densities.

$$\begin{aligned} & KL(q_\theta(w)||p(w|X,Y)) \\ &= \int q_\theta(w) \log \frac{q_\theta(w)}{p(w|X,Y)} dw \\ &= \int q_\theta(w)(\log q_\theta(w) - \log p(Y|X,w) - \log p(w) \\ &+ \log p(X))dw \\ &= -\mathbb{E}_{q_\theta(w)}[\log p(Y|X,w)] + KL(q_\theta(w)||p(w)) \\ &+ \log p(X) \end{aligned} \quad (3)$$

Since the last term $\log p(X)$ is constant under $q_\theta(w)$, we can remove it.

The resulting objective function $L_{VI}$, commonly refered to as either variational free energy or evidence lower bound (ELBO), is given by the negation of the KL divergence. Hence, **minimizing** the KL divergence is equivalent to **maximizing** the ELBO.

$$L_{VI} = \mathbb{E}_{q_\theta(w)}[\log p(Y|X,w)] - KL(q_\theta(w)||p(w)) \quad (4)$$

In order to maximize the objective function $L_{VI}$, we maximize the model's log-likelihood over the training data, which is expressed by the first term in Equation 4 as $\mathbb{E}_{q_\theta(w)}[\log p(Y|X,w)]$, and minimize the KL divergence between the variational distribution and the prior, which is expressed by the second term in Equation 4 as $KL(q_\theta(w)||p(w))$. However, as both terms are intractable to compute, Monte Carlo approximations are used in their place (Blundell, 2015; Gal, 2016).

During each forward pass, the model generates a single realization of every parameter (weights and biases) from the variational distribution, and uses them to compute the loss. The model runs several forward passes to calculate a series of losses and averages them. During the backward pass, the averaged loss is propagated to the parameters, which will then run their local gradient updates.

## Bayesian Dropout Model

Deep neural networks with large number of parameters have been prone to overfitting. Dropout is a technique used to address this problem by randomly dropping units (along with their connections) from the neural network during training. This prevents units from co-adapting too much (Srivastava, 2014).

Dropout has been proposed as an approximation to Bayesian neural networks (Gal, 2016; Li, 2017), where the log-likelihood over the training data (first term in Equation 4) is approximated using Monte Carlo integration and the KL divergence between the variational distribution and the prior (second term in Equation 4) is approximated to be the $L_2$ regularization of the weights and biases of the network. Typically in a normal neural network, dropout is applied only during training. On the other hand, in a Bayesian dropout model, dropout is used during inference as a component of the model's variational distribution.

$$q_\theta(w_i) = p\mathcal{N}(m_i, s^2 I_k) + (1-p)\mathcal{N}(0, s^2 I_k) \quad (5)$$

In the Bayesian dropout model proposed by Gal (2016), the following variation distribution to approximate the posterior. (5) Here, $p$ is the dropout keep probability. The subscript $i$ for $w$ and $m$ is used to denote the $i$th row of the matrix. $m_i$ is the value of the weights in the $i$th row before dropout is applied. Applying dropout to $m_i$ gives us $w_i$. From Equation 5, we see that $w_i$ has an expected value of $p \odot m_i$, where $\odot$ is the Hadamard product.

There are several ways to consider this variational distribution. The most straightforward way is to view it as a mixture of two Gaussians that forms a multimodal distribution (Gal, 2016). Gal set both $p$ and $s$ to be small. He suggested that $s$ take the value of machine epsilon, which is an upper bound for relative error due to rounding in floating point arithmetic in computers. This means that even if the variances are non-zero, they are too small to be considered or be expressed precisely by the computer, and most samples will have their values as mean of either Gaussians. As such, we can also consider the variational distribution as a Bernoulli distribution that returns $m_i$ with probability $p$ and 0 with probability $1-p$, not unlike an ordinary dropout model. One more perspective is to view the variational distribution as a Bernoulli sum of delta functions.

One thing we found novel in Gal's approach is that he fixed the variance of the variational distribution, and only allowed its mean to change. Though this is more tractable computationally, we felt that such restrictions may worsen the variational approximation. For example, fixing the variances to be small can cause the model to underestimate uncertainty for predictions. This is observed in Gal's Mauna Loa experiment, where the uncertainty of his Bayesian dropout model is much smaller that that of a GP model. This caused the true function to lie outside the uncertainty region (95% confidence interval) most of the times, which was not the case for the GP model. We used Gal's Bayesian dropout CNN for the active learning experiment.

## Active Learning

Given a Bayesian neural network that provides us with information about the model's uncertainty, we can attempt to try and reduce the amount of data required to train a reliable neural network model. This acheived by actively sampling more data points from our unobserved data and is known as active learning. We considered the following active learning criteria for selecting the samples to train our neural network. Here, $p_i$ is a shorthand for $P(y^* = i|x^*, w, X, Y)$, that is the probability of $y^*$ being in class $i$, given the posterior and the unobserved input $x^*$.

### Maximum entropy

$$\mathbb{H}[f_d] = -\sum_{i=1}^{C} p_i \log p_i \tag{6}$$

Entropy of a single unobserved data sample $d$ is the expected negative log density (Equation 6 above). $C$ denotes number of classes and $p_i$ is the probability of $d$ being in class $i$. We want to select a constant-sized sample $D$ from the population of available unobserved locations such that $\sum_{d \in D} \mathbb{H}[d]$ is maximized. This is equivalent to choosing examples with highest uncertainty and minimising the posterior entropy or uncertainty left in the remaining unobserved locations.

### Maximum mean variance

$$\sigma^2 = \frac{1}{C} \sum_{c} \left( \mathbb{E}_{q(w)}[p_c^2] - \mathbb{E}_{q(w)}[p_c]^2 \right) \tag{7}$$

In the mean variance of a sample (Equation 7 above), $p_c$ is the probability of a sample being classified as class $c$. $\sigma^2$ is found by averaging sum of variances over all $c$ classes that the sample can take. This is similar to maximing mean standard deviation (Kampffmeyer et al., 2016; Kendall et al, 2015).

### Maximum variation ratio

$$\text{var\_ratio} = 1 - \max_{c} p_c \tag{8}$$

The variation ratio (Freeman, 1965) of a sample can found by the above equation. Where $c$ denotes the class $c$ and $p_c$ is the probability of the sample being in class $c$. Similar to maximum entropy, this acquisition function measures the lack of confidence.

### Random acquisition

We define a uniform distribution over population of unobserved locations and randomly select a sample from it. This method does not utilize any heuristics and serves as a control for other active learning methods.

## Active Learning Experiment

CNNs have been successfully applied to analyzing visual contents and other spatially distributed data. Its defining feature is convolution operation on the receptive field which leads to some layers being non-fully connected. Generally, a simple case of conventional CNN architecture could be described as a series of convolutional blocks each containing convolution, element-wise nonlinearity and pooling layers followed by a classification part consisting of flattening and one or more fully connected layers with softmax activation function.

In our active learning experiment, we compare the performance of a Bayesian dropout CNN conditioned on a reduced-size actively selected MNIST dataset using various active learning methods. The Bayesian dropout CNN we used is identical to Gal's, having a dropout layer with dropout probability $0.5$ right before the final fully-connected layer (Gal, 2016). The rest of the model is similar to LeNet's architecture (see Figure 1).

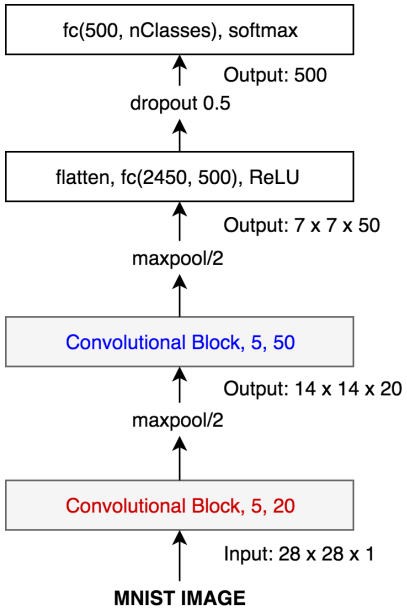



Figure 1: Gal's (2016) Bayesian dropout CNN

MNIST is a subset of 60,000 training and 10,000 test examples of NIST. Data has been centered and normalized in size and comes in as greyscale 28x28 images as seen in 2.

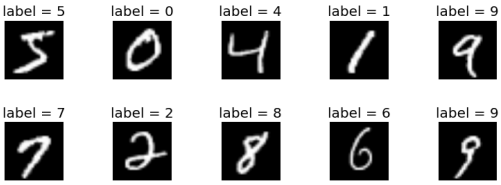



Figure 2: Sample from MNIST dataset

## Results and Evaluation

We first explore the empirical results produced from performing active learning on the Bayesian dropout CNN as proposed in (Gal, 2016). The dataset used for these experiements is the MNIST dataset as described earlier. The neural network is first trained with only 50 rows of input data. With the uncertainty values provided by our Bayesian neural network, we are able to make use of several active learning criteria. We then subsequently select 100 more data points from the unobserved data by calculating these criteria. We perform 20 iterations of selecting new data points. Thus the total data that we used to train our neural network

is merely 2,050. We then plot the test accuracies (see Figure 3) after each iteration of active learning.

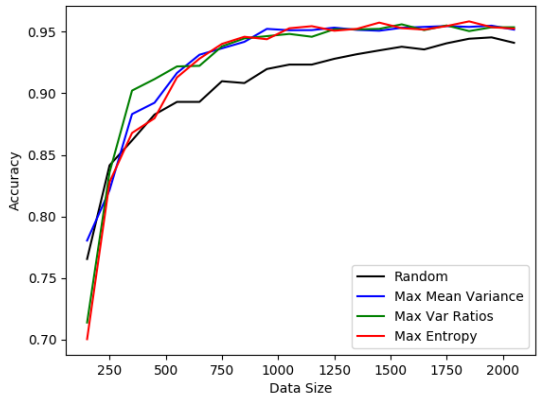



Figure 3: Accuracy scores of models with trained with different data sizes

We found that after 10 iterations (results in Table 1), the models trained with non-random acquisition functions are able to perform as well as neural networks that trained with the full 60,000 rows of data.

| Criterion | Accuracy |
|---|---|
| Maximum Entropy | 95.28% |
| Maximum Mean Variance | 95.12% |
| Maximum Variation Ratio | 94.82% |
| Random Sampling (Control) | 90.83% |

Table 1: Active Learning Results (1,050 total data points)

We can observe that the performance of the different acquisition functions are very similar. At this point, we suspect that the cause this similarity was due to the fact that the MNIST dataset is trivial, i.e. if we are given a large enough dataset, the performance will not vary as much. However it can be observed that all our models trained with active learning performs better than our control, which is random sampling.

To push our active learning to the limits, we switch to sampling 50 data point at a time instead. We then run the active learning iteration for 10 iterations and compare to see how each approach performs with only 550 data points, nearly half of what we had before. The results of our experiments are shown in Table 2 below.

| Criterion | Accuracy |
|---|---|
| Maximum Entropy | 90.48% |
| Maximum Mean Variance | 91.67% |
| Maximum Variation Ratio | 92.57% |
| Random Sampling (Control) | 83.84% |

Table 2: Active Learning Results (550 total data points)

We also tried to push the limits even further by training the neural network with only an initial amount of 10 rows of data. However, with a datasize this small, it is not possible to obtain a stable neural network. Therefore, 50 is a comfortable enough size for the neural network to give meaningful predictive mean and variance values for active learning to take place.

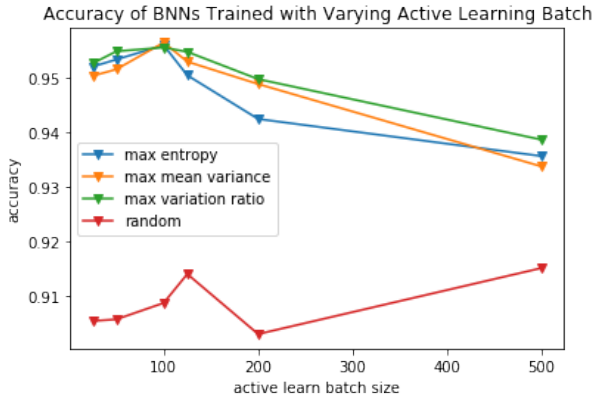



Figure 4: Accuracy scores of models trained with active learning batch sizes of 25, 50, 100, 150, 200 and 500

From the above experiment we found out that the 550 data points are too few to make an accurate comparison between the different active learning criterions.

Another interesting thing to notice is how taking small batches of best samples to add to the training data performs similarly, or even better than taking individual best samples, while being more computationally efficient. We call this **batch active learning**. We can tune the size of each active learning batch to experiment with the efficiency of our Bayesian neural network. Selecting too many samples at a time introduces independence between the different rows of data, and hence affects the criterion value of the next iteration. Therefore, we ran experiments on our Bayesian neural network for sample sizes of 25, 50, 100, 150, 200 and 500. The results can be seen in the graph plotted in Figure 4.

This is analagous to how humans learn. In school when preparing for finals, we do past year exams to identify our weakest topic. We then do drill questions for this topic. However, we should not do too many drill questions (taking a large batch) since there are deminishing returns (once we have grasped the topic, our time can be better spent on other topics). Nor should we do a single drill question (taking single best sample) and go back to doing past year exams, since taking the past year exam itself is time-consuming and we have to finish studying before finals.

## Conclusion

In conclusion, active learning allows Bayesian neural networks to learn faster with less training data. This has merit, given how neural networks tend to be data hungry, and while raw data is abundant, getting them preprocessed and labeled

properly can be both tedious and expensive. We propose doing batch active learning, since it is more computational efficient while being just as effective as individual sampling, as shown in our experiments.

We found out that defining priors over Bayesian neural networks is not trivial, and there are many factors to consider. A prior which works for one architecture (Gaussian on feedforward) may not work for another (Gaussian on CNN). We want to make sense of why this is so, but most of the time we could not find a rigorous enough explanation.

Finally, we want to evaluate on the practicality of Bayesian neural networks. Bayesian neural networks combine the strengths of both conventional neural networks and Bayesian approaches such as GPs, such as being fast during inference and being able the model uncertainty. However, they also have several notable disadvantages. They are still slower than ordinary neural networks, since they require sampling, which can be computationally expensive. Additionally, how well they model the uncertainty of their predictions depends a lot on the variational inference used. In Yarin's Mauna Loa experiment, his model constantly underestimated the uncertainty of its predictions while a GP did not.

The priors we define over neural networks are supposed to embody our initial beliefs about the problem, however they are anything but that. Despite this, certain priors, such as Gaussian, have been shown to produce at least reasonable results (Neal, 1995). Bayesian neural networks can be practical if we can define a prior which is both computable and effective, for example a Bernoulli prior converges to the posterior much better than Gaussian priors for a CNN. However, finding such priors can be extremely challenging and empirical.

The source code for our experiments can be found at `https://github.com/wenqiwooo/Active-Learning-on-Bayesian-Neural-Networks`.

The Appendix section contains several other experiments conducted by us.

## Appendix: Other Experiments

Besides the the active learning experiment, we conducted several other experiments on Bayesian neural networks and thought it would be interesting to share our results and evaluations here.

### Gaussian Prior on CNN

Instead of applying dropout during both training and test times and having a multimodal prior (Gal, 2016), our initial attempt at a Bayesian CNN was to place a Gaussian prior over each parameter. In our implementation, we store the means and variances of our weights and biases as matrices. Similar to the Bayesian dropout CNN (Gal, 2016), we used variational inference to approximate the posterior by minimizing the KL divergence between our variational distribution and the posterior. Unlike Gal's dropout CNN (which samples only the final layer's parameters), we have to sample every parameter when making inference, and this was rather computationally expensive. We also had an ordinary CNN with similar architecture to be used as benchmark. Unfortunately, our Bayesian model's performance was not ideal, with a test accuracy of 11.1% (hardly better than random guessing) after 10000 iterations while our benchmark model was already getting 64.3% accuracy after 100 iterations. We use iterations here to denote each time the model performs a minibatch gradient descent to optimize its parameters.

Gaussian priors over parameters seem most popular with feedforward networks and recurrent neural networks (RNNs). So far, we have yet to find any working Gaussian CNN models.

### Gaussian Prior on Feedforward Network

We reduced the complexity of our Bayesian model to a simple feedfoward network with no hidden layers and softmax output. All weights and biases priors were defined as standard Gaussian distributions. Surprisingly, this simple model performed much better than our initial Bayesian CNN attempt and achieved a test accuracy of 90.0% after 10000 iterations, which was comparable to the performance of a non-Bayesian model with similar architecture. However, we are more interested in networks with one or more hidden layers.

| Architecture | Parameters | Accuracy |
|---|---|---|
| 784x10 | 7840 | 0.90 |
| 784x64, 64x10 | 50816 | 0.81 |
| 784x128, 128x10 | 101632 | 0.85 |
| 784x392, 392x10 | 311248 | 0.89 |
| 784x64, 64x32, 32x10 | 52544 | 0.11 |
| 784x392, 392x64, 64x10 | 333056 | 0.11 |

Table 3: MNIST results for feedforward NN with standard Gaussian as prior. All models were run for 10000 iterations.

For networks with a single hidden layer, we observe notable improvements in performance in models with higher dimensional hidden layers (see Table 3). Perhaps more complex models perform better for Bayesian deep learning, but we will need more empirical results to validate this.

For networks with two hidden layers, we found deciding on priors to be extremely challenging and we're unable to come up with priors which allowed the network to achieve better than random performance. Poorly selected priors are also likely to be what caused our initial Bayesian CNN to fail at prediction.

We hypothesized that not all of our priors should be standard Gaussians, since the variance can get very large towards the lower layers with the multiplication of random parameter variables. True enough, it has been proposed that the hidden $\rightarrow$ output parameters should be scaled by a constant $|N|^{-\frac{1}{2}}$,

where $|N|$ denotes the number of hidden units, when defining priors for networks with multiple hidden layers (Neal, 1995). However, performance did not get any better when we implemented this on our model.

## Beta-Bernoulli Prior on Bayesian Dropout

Gal's Bayesian dropout CNN defines the prior as a mixture of two Gaussians, or equivalently as a Bernoulli sum of delta functions, since the variances of the Gaussians are scaled down to machine epsilon.

We felt that the novelty in Yarin's Bayesian dropout (in the variational distribution, only the mean changes and the mean is fixed) is also an restriction. What if the practitioner does not know what fixed value to set the dropout hyperparameter to? And what if the data does not agree with such a value? Can we place a prior over the dropout hyperparameter instead, and let the training data decide on a good value?

We augmented Yarin's Bayesian dropout CNN to have a Beta prior with $\alpha = 20$ and $\beta = 20$ on its dropout hyperparameter (its most probable value is $0.5$ but it may take other values as well). We will refer to this as the Beta dropout CNN. After training on the full-sized MNIST dataset, the model achieved an accuracy on $98.0\%$.

We also accessed performance of our Beta dropout CNN using the CIFAR-10 dataset and it managed an accuracy of $63.5\%$ after 200 epochs. We were more curious in whether the posterior of the dropout hyperparamter deviated from its prior after training. From Figure 5, we observe a slight rightward shift of the distribution curve. This may suggest that a larger dropout hyperparameter may fit the data better.

Initially, our model was unable to attain better than random performance. After much trial and error, we finally got our model to converge by reducing the learning rate and doubling the hidden layer size from 500 to 1000. Coincidentally, we observed performance improvement in our single hidden layer Bayesian feedforward network as its hidden layer size increases (see Table 3). It is also interesting how Bayesian neural networks with larger hidden layers approximates a GP better. Perhaps when one's Bayesian neural network is not converging, he/she should try increasing the size of its hidden layers.

## Multiple Beta-Bernoulli Priors on Bayesian Dropout

In ordinary CNNs, it is not uncommon to perform a dropout operation after each convolutional block. We augmented our Beta dropout CNN by inserting additional dropout layers and placing a Beta prior over each of them (see Figure 6).

From the results in Table 4, we observe that the Beta priors can have a huge impact on the model's performance. The model was unable to converge if most of the dropout hyperparameters sampled were too low. Additionally, to get the model to converge, we had to scale down the learn rate by a factor of 10 compared to our previous model.

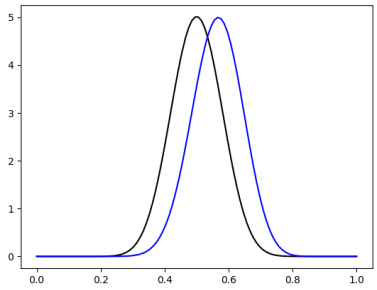



Figure 5: Prior (black) and posterior (blue) of distributions of dropout hyperparameter in the Beta dropout CNN model after 200 epochs of training.

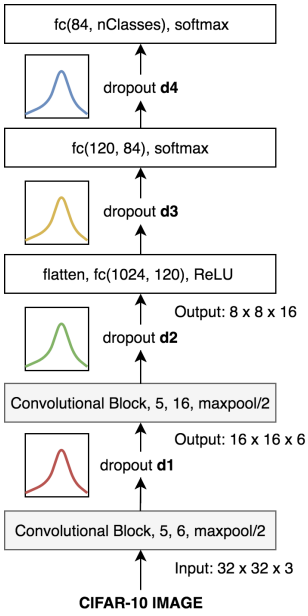
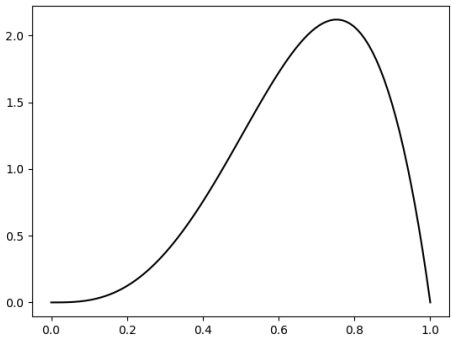



Figure 6: Architecuture of a Bayesian neural network with Beta distributions on the dropout layers.

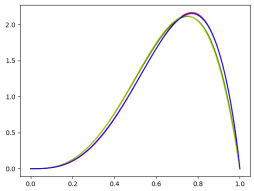
We hypothesize that for a Bayesian neural network to converge within a reasonable time, the prior over the network should not be too "fluctuating". For a more fluctuating prior, we have to reduce the learning rate, however, the model is more likely to end up in an local optimum in this case and may never converge.

When we deliberately have our priors produce higher dropout probabilities with higher probability when sampled (with $\alpha = 4$ and $\beta = 2$), the model achieves an accuracy of $65\%$ after 200 epochs, which is comparable to the performance of the Beta dropout CNN with a single dropout layer.

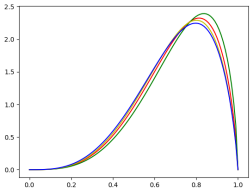
The plots for the posteriors of the model's dropout hyperparameters are shown in Figure 7. It is interesting to notice the posteriors gradually shift rightwards, which is similar to our results shown in Figure 5.



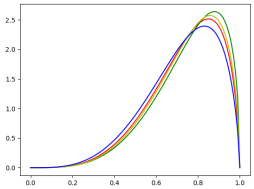
(a) Prior



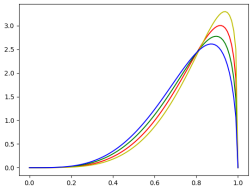
(b) After 10 epochs



(c) After 20 epochs



(d) After 30 epochs



(e) After 40 epochs

Figure 7: Prior and posterior of dropout hyperparameters $d_1$ (red), $d_2$ (green), $d_3$ (yellow), $d_4$ (blue) over 40 epochs. The same prior is defined over all dropout parameters with $\alpha = 4$ and $\beta = 2$.

| $\alpha$ | $\beta$ | Accuracy |
|---|---|---|
| 4 | 2 | 0.53 |
| 20 | 20 | 0.10 |
| 2 | 4 | 0.10 |

Table 4: CIFAR-10 results for Bayesian dropout CNN with Beta priors for dropout hyperparameters. All models were run for 40 epochs.

### Active Learning with Higher Dropout

When we increased value of the dropout (keep) probability in Yarin's model to 0.99, it achieved 97.1% accuracy with 50 initial and 1000 additional data points, which were actively selected using max entropy criterion. This is notably better compared than our results in Table 1.

Interestingly, when we set dropout (keep) probability to 1 (nothing was dropped at all), the model's performance faltered and active learning did not seem to perform any better than random sampling.

## References


Ba, L.J. & Frey B. (2013). Adaptive dropout for training deep neural networks.

Blei, D.M. & Kucukelbir, A. & McAuliffe J.D. (2017) Variational Inference: A Review for Statisticians.

Blundell, C. & Cornebise, J. & Kavukcuoglu, K. & Wierstra, D. (2015, May). Weight Uncertainty in Neural Networks. International Conference on Machine Learning. arXiv:1505.05424.

Duvenaud, D., Rippel, O., Adams, R., & Ghahramani, Z. (2014, April). Avoiding pathologies in very deep networks. In Artificial Intelligence and Statistics (pp. 202-210).

Fortunato, M. & Blundell, C. & Vinyals, O. (2017). Bayesian Recurrent Neural Networks.

Freeman, Linton G. Elementary applied statistics, 1965.

Gal, Y. & Ghahramani, Z. (2016, June). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. International Conference on Machine Learning (pp. 1050-1059).

Gal, Y. (2015, July). What My Deep Model Doesn't Know... `http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html`

Graves, A (2011). Practical Variational Inference for Neural Networks.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., & Sohl-Dickstein, J. (2017). Deep Neural Networks as Gaussian Processes. arXiv preprint arXiv:1711.00165.

Li, Y. & Gal, Y. (2017). Dropout Inference in Bayesian Neural Networks with Alpha-divergences.

MacKay, D. (2003) Information Theory, Inference and Learning Algorithms.

Neal, Radford M. (1995). Bayesian learning for neural networks.

Shewry, M.C. & Wynn, H.P. (1987). Maximum entropy sampling, Journal of Applied Statistics 46, 165  170.

Srivastava, N. & Hinton, G. & Krizhevsky, A. & Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 15. 1929-1958.

Tran, D. & Brevdo, E. & Hoffman, M.D. & Murphy, K. & Saurous, R.A. & Blei, D.M. (2017). Deep Probabilistic Programming.

Williams, C. K. (1997). Computing with infinite networks. In Advances in neural information processing systems (pp. 295-301).