Chapter 4

Uncertainty Quality

In this chapter we assess the techniques developed in the previous chapters, concentrating on questions such as what our model uncertainty looks like. We experiment with different model architectures and approximating distributions, and use various regression and classification settings. Assessing the models' confidence quantitatively we can see how much we sacrifice in our attempt at deriving *practical* inference techniques in Bayesian neural networks (NNs).

More specifically, in this chapter we will look at how model structure affects model uncertainty by sampling functions from our prior. This can help in choosing an appropriate model for a given dataset. Given a dataset, we will then assess how different approximating distributions capture noisy data and uncertainty far from the data. We will compare the uncertainty estimates obtained from several stochastic regularisation techniques (SRTs) such as dropout and multiplicative Gaussian noise, as well as more traditional approximating distributions in variational inference (VI) such as a fully factorised Gaussian and a mixture of Gaussians. This is followed by a study of model uncertainty in classification, as well as specialised models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). We defer real-world examples to the next chapter.

4.1 Effects of model structure on uncertainty

We start by looking at how model structure affects uncertainty, i.e. the variation in functions captured by a given model structure. We look at the *a priori* distribution over functions, before performing inference. This distribution weighs some functions as being more likely by giving them a higher prior probability, and weighs other functions as being less likely by giving them a lower prior probability. In practice this results in a higher

penalty for the a priori less likely functions when we optimise our variational objective, trying to find a function to fit our data.

The distribution over functions can be visualised by sampling weight matrices from the prior with different non-linearities, model sizes, and number of hidden layers. We then evaluate the network with the sampled weights over a grid of points (for example $[-2, 2]$) and plot the results. Such a plot depicts a single draw from the prior over functions. Repeating this procedure multiple times we can get an idea of what the prior over functions looks like for a particular prior over weight matrices.

As a starting point, we evaluate networks with four hidden layers (fig. 4.1) and a choice of three non-linearities (ReLU in fig. 4.1g, TanH in fig. 4.1n, or Sigmoid in fig. 4.1u). We use the prior $\mathcal{N}(0, 1/l^2)$ over both the weights and biases, with different length-scale values l. We evaluate three model sizes: networks with 32 units in each hidden layer, 512 units, or 4096 units. We demonstrate the effects of prior choice by varying the prior length-scale. We experiment with length-scales 1 and 10 for the ReLU and TanH non-linearities, and length-scales 0.1 and 1 for the Sigmoid non-linearity. It is clear from the plots that longer length-scales result in smoother functions, and larger models result in more erratic functions[1]. ReLU networks seem to be mostly invariant to prior length-scale, with the bias magnitude changing the spread of the functions. A similar plot for a network with a single hidden layer is given in appendix B (fig. B.1).

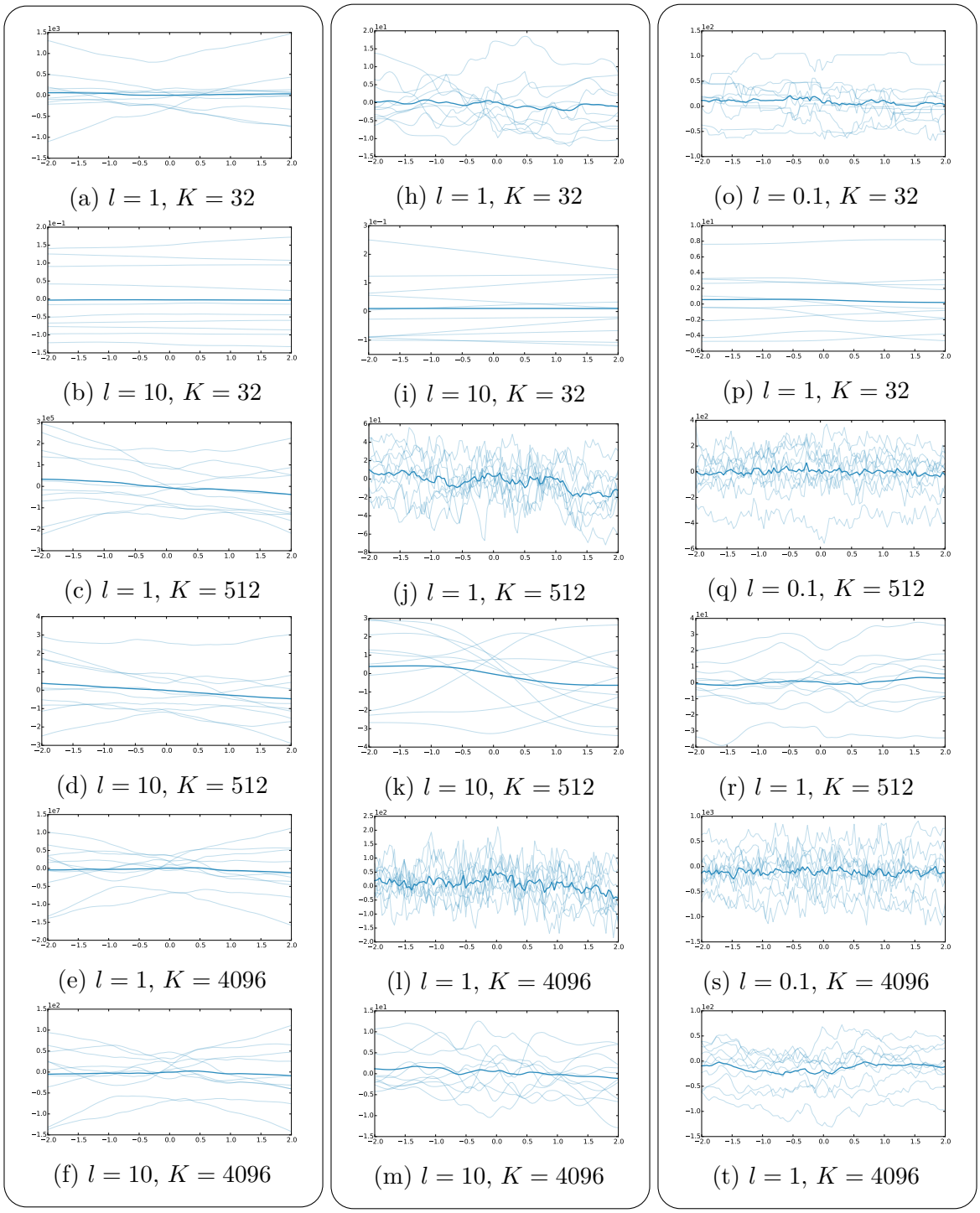
It is interesting to see that short length-scales in the lower layers yield more erratic functions, whereas the length-scale of the output layer ($l_5^{\mathbf{W}}$ for a network with 4 hidden layers) controls the output magnitude (fig. 4.2). The effect of the bias length-scale can be seen in fig. 4.3. Vertical distance between the functions changes with a short bias length-scale, as well as the frequency magnitude for the longer bias length-scale.

4.2 Effects of approximate posterior on uncertainty

We next assess the uncertainty estimates obtained from various approximating distributions on the task of regression. We compare the uncertainty obtained from different model architectures and non-linearities, both on tasks of extrapolation and interpolation. We use three regression datasets and model scalar functions which are easy to visualise. These are tasks one would often come across in real-world data analysis.

We start with a small synthetic dataset based on the one presented in [Snelson and Ghahramani, 2005]. This dataset consists of 5000 data points, with a fairly large amount

[1] This was also observed by [Neal, 1995]. Larger NNs can capture a larger class of functions, thus will have higher uncertainty as well. Note that a function's smoothness depends on its X-axis range.



(a) $l = 1$, $K = 32$ (h) $l = 1$, $K = 32$ (o) $l = 0.1$, $K = 32$

(b) $l = 10$, $K = 32$ (i) $l = 10$, $K = 32$ (p) $l = 1$, $K = 32$

(c) $l = 1$, $K = 512$ (j) $l = 1$, $K = 512$ (q) $l = 0.1$, $K = 512$

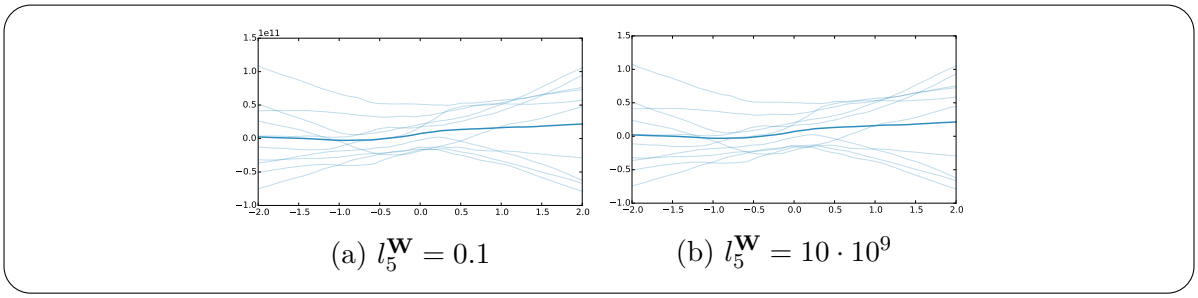
(d) $l = 10$, $K = 512$ (k) $l = 10$, $K = 512$ (r) $l = 1$, $K = 512$

(e) $l = 1$, $K = 4096$ (l) $l = 1$, $K = 4096$ (s) $l = 0.1$, $K = 4096$

(f) $l = 10$, $K = 4096$ (m) $l = 10$, $K = 4096$ (t) $l = 1$, $K = 4096$

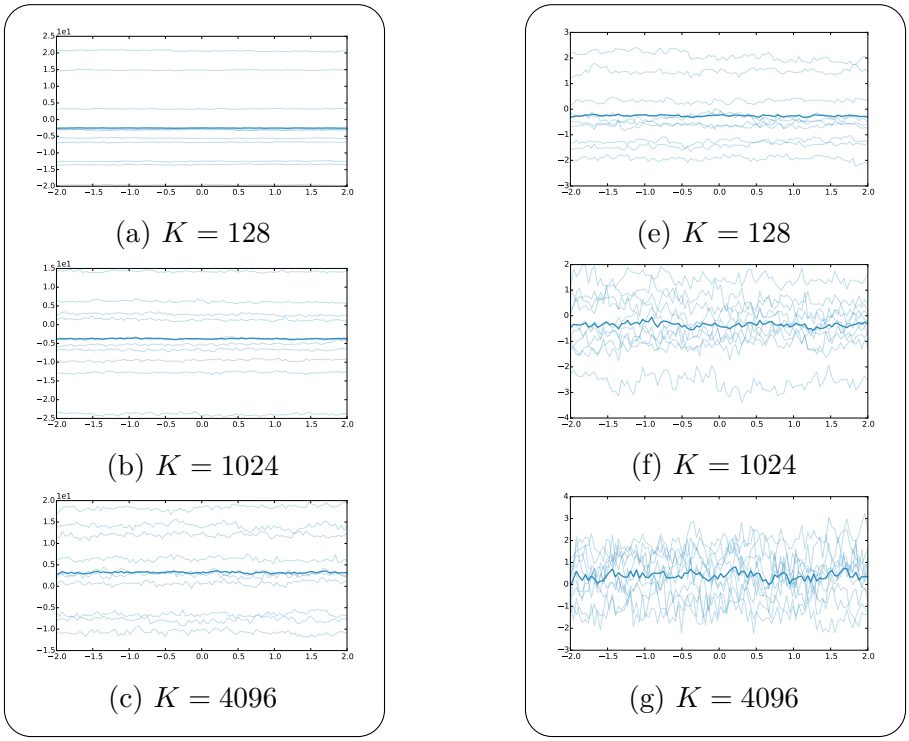
(g) ReLU (n) TanH (u) Sigmoid

Fig. 4.1 Draws from a Bayesian neural network prior ($\mathbf{W} \sim \mathcal{N}(0, 1/l^2)$) with $L = 4$ hidden layers, for various non-linearities, model sizes (K), and length-scales (l). Thick line is the mean of 20 samples. Best viewed on a computer screen.



(a) $l_5^{\mathbf{W}} = 0.1$ (b) $l_5^{\mathbf{W}} = 10 \cdot 10^9$

Fig. 4.2 Draws from a Bayesian neural network prior with $L = 4$ hidden layers, $K = 1024$ units, short length-scale $l = 0.1$, and ReLU non-linearity; Weight length-scale in the last layer $l_5^{\mathbf{W}}$ affects the Y axis magnitude.



(a) $K = 128$ (e) $K = 128$

(b) $K = 1024$ (f) $K = 1024$

(c) $K = 4096$ (g) $K = 4096$

(d) $l_5^{\mathbf{b}} = 0.1$ (h) $l_5^{\mathbf{b}} = 1$

Fig. 4.3 Effect of bias length-scale on draws from a Bayesian neural network prior. All networks use $L = 4$ hidden layers, short length-scale $l = 0.1$, and TanH non-linearity; We use $l_5^{\mathbf{W}} = 100$ to decreasing the output magnitude for drawing purposes. Note the change in vertical distance between the functions with short bias length-scale, and frequency magnitude for the longer bias length-scale.

of noise in the data. We evaluate model extrapolation as well as confidence on the noisy data. We then use a subset of the atmospheric CO_2 concentrations dataset derived from in situ air samples collected at Mauna Loa Observatory, Hawaii [Keeling et al., 2004] (referred to as CO_2) to evaluate model extrapolation on noise*less* data. We give further results on the reconstructed solar irradiance dataset [Lean, 2004] assessing model interpolation. The last two datasets are fairly small, with each dataset consisting of about 200 data points. We centred and normalised both datasets.

In the following experiments we assess VI approximate inference with various approximating distributions:

1. Bernoulli approximating distribution, implemented as **dropout**:

$$\boldsymbol{\omega} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{W}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{W}_2, \mathbf{b}\}$$

   with variational parameters $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}$ and $p(\boldsymbol{\epsilon}_l)$ $(l = 1, 2)$ a product of Bernoulli distributions with probability p_l.

2. Multiplicative Gaussian approximating distribution, implemented as **multiplicative Gaussian noise (MGN)**: $\boldsymbol{\omega} = \{\text{diag}(\boldsymbol{\epsilon}_1)\mathbf{W}_1, \text{diag}(\boldsymbol{\epsilon}_2)\mathbf{W}_2, \mathbf{b}\}$ with variational parameters $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}$ and $p(\boldsymbol{\epsilon}_l)$ (for $l = 1, 2$) a product of draws from $\mathcal{N}(1, p_i/(1 - p_i))$.

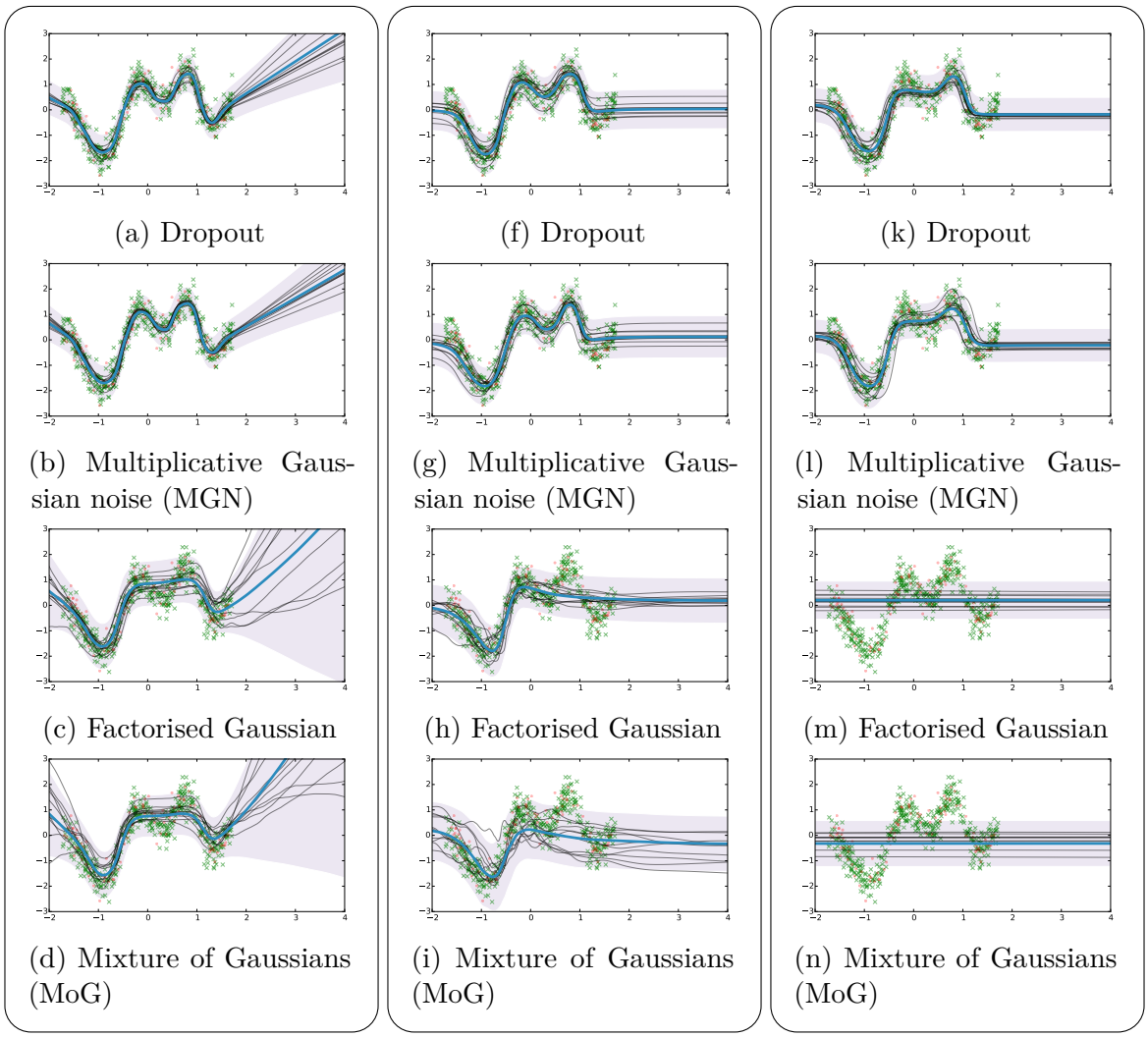
3. A fully **factorised Gaussian** distribution, similar to the one used in [Graves, 2011], but following the pathwise derivative estimator brought above.

4. A **mixture of Gaussians (MoG)** with two mixture components, factorised over the rows of the weight matrices (similar to the factorisation assumptions in the Bernoulli approximating distribution; this is identical to the mixture of Gaussians approximation in appendix A, but with the standard deviations not fixed at small values).

For the first two methods we assumed a delta approximating distribution over the biases, whereas in the last two we placed a factorised Gaussian over the bias. To avoid a big increase in the number of parameters we tied the standard deviations at each layer in these models, and used a single scalar parameter.

4.2.1 Regression

We start by assessing model uncertainty on the small and noisy [Snelson and Ghahramani, 2005] dataset. Figure 4.4 shows the fits of the two SRT approximating distributions



(a) Dropout (f) Dropout (k) Dropout

(b) Multiplicative Gaussian noise (MGN) (g) Multiplicative Gaussian noise (MGN) (l) Multiplicative Gaussian noise (MGN)

(c) Factorised Gaussian (h) Factorised Gaussian (m) Factorised Gaussian

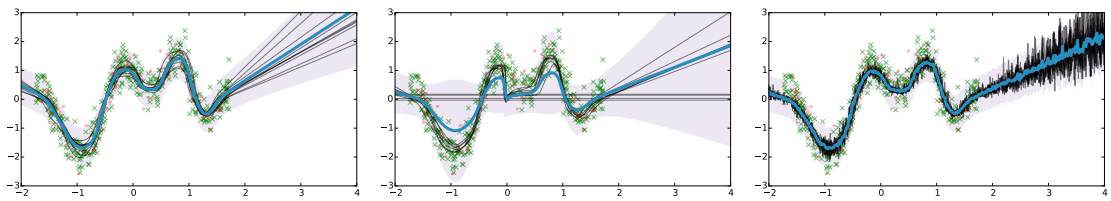
(d) Mixture of Gaussians (MoG) (i) Mixture of Gaussians (MoG) (n) Mixture of Gaussians (MoG)

(e) ReLU (j) TanH (o) Sigmoid

Fig. 4.4 Draws from a Bayesian neural network posterior with various approximating distributions; The networks used have $L = 4$ layers, each with $K = 1024$ units, length-scale $l = 5$, $p = 0.5$ for the mixture models, with different non-linearities. Shown are predictive mean (thick blue line), predictive uncertainty (shaded area, showing 2 standard deviations), and draws from the posterior (thin black lines). Scattered are training points. Best viewed on a computer screen.

(dropout and multiplicative Gaussian noise), as well as the factorised Gaussian and the mixture of Gaussians. This is shown for a network with $L = 4$ hidden layers, $K = 1024$ units in each layer, and various non-linearities. Each SRT model was optimised for 50 epochs using the Adam optimiser [Kingma and Ba, 2014], and the VI models were optimised for 100 epochs. Prior length-scale for all layers was set to $l = 5$, with weight decay set to $4 \cdot 10^{-5}$ (following eq. (3.14), with model precision fixed to its true value at $\tau = 10$). Note how for the ReLU models uncertainty increases far from the data for all approximating distributions, which is not the case with the other non-linearities. Note also how the factorised Gaussian and mixture of Gaussians seem to underfit compared to the SRTs. All models increase their uncertainty to capture the magnitude of the noisy data, with draws from the posteriors depicting the possible functions that can explain the data. Note also how dropout and MGN result in practically identical model uncertainty. Further results are given in appendix B, comparing model fits with different numbers of hidden layers ($L = 1$ and $L = 4$), model sizes ($K = 128$ and $K = 1024$), and non-linearities (ReLU, TanH, and Sigmoid) for dropout (fig. B.2), multiplicative Gaussian noise (fig. B.3), fully factorised Gaussian (fig. B.4), and mixture of Gaussians (fig. B.5). Interestingly, with the smaller models ($K = 128$) MoG and the factorised Gaussian do not underfit.

In the above in each model we set the SRT probability for the first layer to zero: $p_1 = 0$. To see why, observe the difference between fig. 4.5a and fig. 4.5b. Both depict the posterior by sampling a set of weights from the approximate posterior ($\mathbf{W}_i \sim q(\mathbf{W}_i)$)



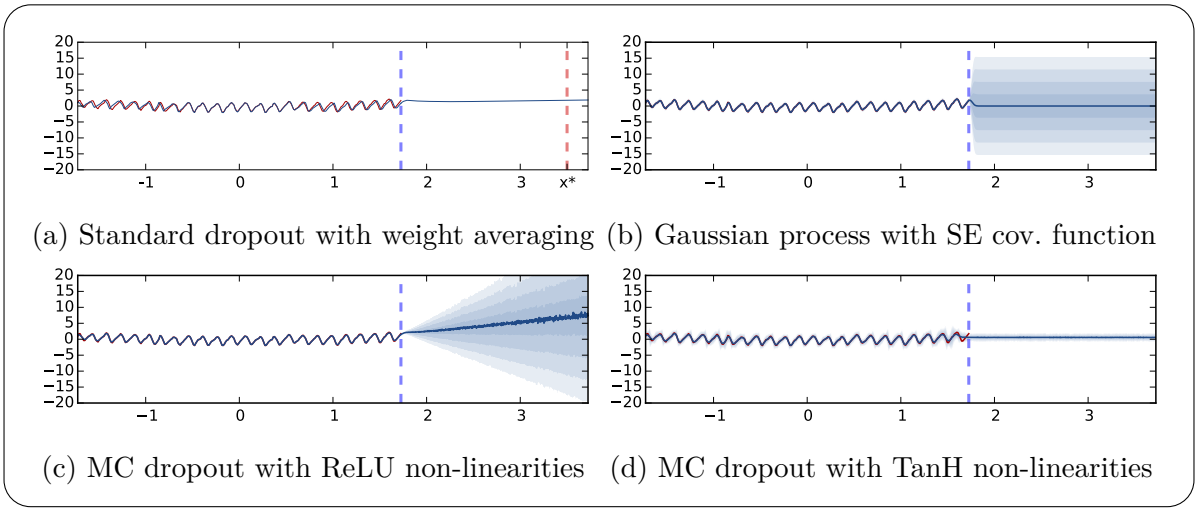
(a) i.i.d. draws per test **set** and $p_1 = 0$ (b) i.i.d. draws per test **set** and $p_1 = 0.5$ (c) i.i.d. draws per test **point** and $p_1 = 0$

Fig. 4.5 Draws from a Bayesian neural network posterior with **dropout** approximating distribution; The networks used have $L = 4$ layers, each with $K = 1024$ units, weight decay $\lambda = 4 \cdot 10^{-5}$ (equivalent to $l = 5$ with $N = 5000$ and $\tau = 10$), and ReLU non-linearities. We have $p = 0.5$ for all layers apart from the first (where we have either $p_1 = 0$ or $p_1 = 0.5$). All networks were trained with dropout masks sampled i.i.d. for each data point, but tested differently. Some were tested by drawing a single mask for the entire test set multiple times (**i.i.d. draws per test set**), whereas others by drawing a different mask for each test point multiple times (**i.i.d. draws per test point**).

and evaluating the network with the sampled weights on the entire space (this is in contrast to sampling a new set of realisations for each test point, as is depicted in figure 4.5c). In fig. 4.5b we set all dropout probabilities to 0.5. As a result, with probability 0.5, the sampled functions from the posterior would be identically zero. This is because a zero draw from the Bernoulli distribution together with a scalar input leads the model to completely drop its input. This is a behaviour we might not believe the posterior should exhibit, and could change this by setting a different probability for the first layer. Setting $p_1 = 0$ for example is identical to placing a delta approximating distribution over the first weight layer. Note that the dropout probability could be optimised instead.

Next, we trained several models on the noiseless CO_2 dataset. We use NNs with either 4 or 5 hidden layers and 1024 hidden units. We use either ReLU non-linearities or TanH non-linearities in each network, and use dropout probabilities of either 0.1 or 0.2. Extrapolation results are shown in figure 4.6. The model is trained on the training data (left of the dashed blue line), and tested on the entire dataset. Fig. 4.6a shows the results for standard dropout (i.e. with weight averaging and without assessing model uncertainty) for the 5 layer ReLU model. Fig. 4.6b shows the results obtained from a Gaussian process with a squared exponential covariance function for comparison. Fig. 4.6c shows the results of the same network as in fig. 4.6a, but with MC dropout used to evaluate the predictive mean and uncertainty for the training and test sets. Lastly, fig. 4.6d shows the same using the TanH network with 5 layers (plotted with 8 times the standard deviation for visualisation purposes). The shades of blue represent model uncertainty: each colour gradient represents half a standard deviation (in total, predictive mean plus/minus 2 standard deviations are shown, representing 95% confidence). Not plotted are the models with 4 layers as these converge to the same results.

Extrapolating the observed data, none of the models can capture the periodicity (although with a suitable covariance function the Gaussian process (GP) will capture it well). The standard dropout NN model (fig. 4.6a) predicts value 0 for point x^* (marked with a dashed red line) with high confidence, even though it is clearly not a sensible prediction. The GP model represents this by increasing its predictive uncertainty—in effect declaring that the predictive value might be 0 but the model is uncertain. This behaviour is captured in MC dropout as well. Even though the models in figures 4.6 have an incorrect predictive mean, the increased standard deviation expresses the models' uncertainty about the point. Note that as before the uncertainty is increasing far from the data for the ReLU model, whereas for the TanH model it stays bounded. For the TanH model we assessed the uncertainty using both dropout probability 0.1 and dropout probability 0.2. Models initialised with dropout probability 0.1 initially exhibit



(a) Standard dropout with weight averaging (b) Gaussian process with SE cov. function

(c) MC dropout with ReLU non-linearities (d) MC dropout with TanH non-linearities

Fig. 4.6 **Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset, for various models.** In red is the observed function (left of the dashed blue line); in blue is the predictive mean plus/minus two standard deviations (8 for fig. 4.6d). Different shades of blue represent half a standard deviation. Marked with a dashed red line is a point far away from the data: standard dropout confidently predicts an unreasonable value for the point; the other models predict unreasonable values as well but with the additional information that the models are uncertain about their predictions.

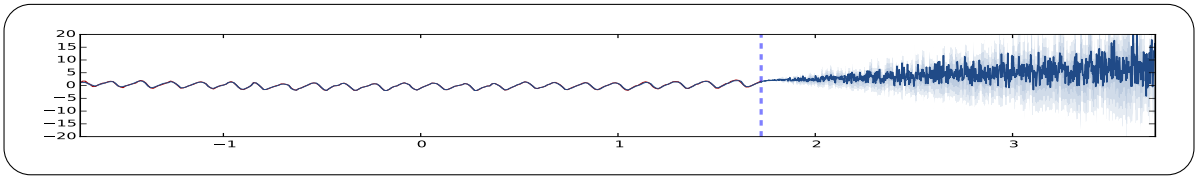


Fig. 4.7 Predictive mean and uncertainties on the Mauna Loa CO_2 concentrations dataset for the MC dropout model with ReLU non-linearities, approximated with 10 samples.

smaller uncertainty than the ones initialised with dropout probability 0.2, but towards the end of the optimisation when the model has converged the uncertainty is almost indistinguishable. It is worth mentioning that we attempted to fit the data with models with a smaller number of layers unsuccessfully.

The number of forward iterations used to estimate the uncertainty (T) was 1000 for drawing purposes. A much smaller numbers can be used to get a reasonable estimation to the predictive mean and uncertainty (see fig. 4.7 for example with $T = 10$).

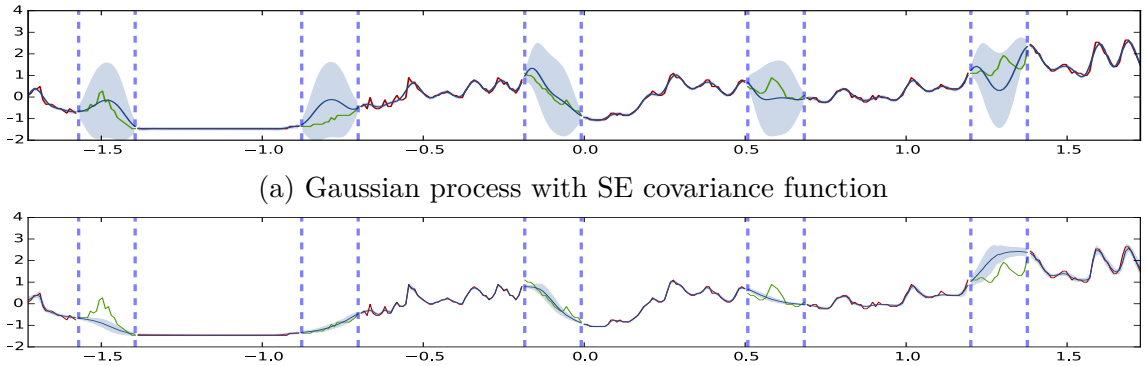
For interpolation we repeat the experiment with ReLU networks with 5 hidden layers and the same setup on a new dataset: solar irradiance. Here we use weight decay of $5 \cdot 10^{-7}$. Interpolation results are shown in fig. 4.8. Fig. 4.8a shows interpolation of missing sections (bounded between pairs of dashed blue lines) for the Gaussian process with squared exponential covariance function, as well the function value on the training set. In red is the observed function, in green are the missing sections, and in blue is the model predictive mean. Fig. 4.8b shows the same for the ReLU dropout model with 5 layers. Both models interpolate the data well, with increased uncertainty over the missing segments. However, VI underestimates model uncertainty considerably here[2]. This experiment demonstrates the dangers with model uncertainty resulting from VI approaches. Note however that all variational inference techniques would under-estimate model uncertainty unless the true posterior is in the class of approximating variational distributions.

4.2.2 Classification

To assess model confidence in classification we test a convolutional neural network trained on the MNIST dataset [LeCun and Cortes, 1998]. We trained the LeNet convolutional neural network model [LeCun et al., 1998] with dropout applied before the last fully connected inner-product layer (the usual way dropout is used in CNNs). We used dropout probability of 0.5. We trained the model for 10^6 iterations. We used Caffe [Jia et al., 2014] reference implementation for this experiment.

We evaluated the trained model on a continuously rotated image of the digit 1 (shown on the X axis of fig. 4.9). We scatter 100 stochastic forward passes of the softmax input (the output from the last fully connected layer, fig. 4.9a), as well as of the softmax output for each of the top classes (fig. 4.9b). The plots show the softmax input value and softmax output value for the 3 digits with the largest values for each corresponding input. When the softmax input for a class is larger than that of all other classes (class 1 for the first

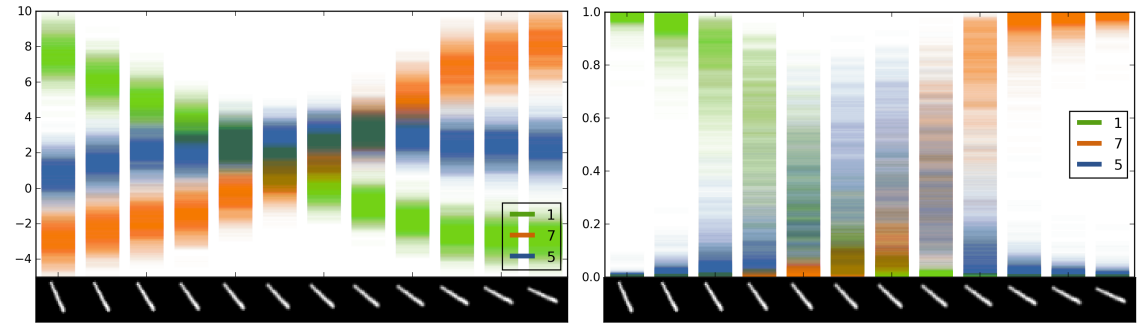
[2]Note the extremely high model precision used: $\tau = \frac{(1-p)l^2}{2N\lambda} = \frac{l^2}{4}10^4$ based on eq. (3.17).



(a) Gaussian process with SE covariance function

(b) MC dropout with ReLU non-linearities

Fig. 4.8 **Predictive mean and uncertainties on the reconstructed solar irradiance dataset with missing segments, for the GP and MC dropout approximation.** In red is the observed function and in green are the missing segments. In blue is the predictive mean plus/minus two standard deviations of the various approximations.



(a) Softmax *input* scatter

(b) Softmax *output* scatter

Fig. 4.9 **A scatter of 100 forward passes of the softmax input and output for dropout LeNet.** On the X axis is a rotated image of the digit 1. The input is classified as digit 5 for images 6-7, even though model uncertainty is extremly large (best viewed in colour).

5 images, class 5 for the next 2 images, and class 7 for the rest in fig 4.9a), the model predicts the corresponding class. For the 12 images, the model predicts classes [1 1 1 1 1 5 5 7 7 7 7 7]. Looking at the softmax input values, if the range of high uncertainty of a class is far from that of other classes' (for example the left most image) then the input is classified with high confidence. On the other hand, if the range of high uncertainty intersects that of other classes (such as in the case of the middle input image), then even though the softmax output can be arbitrarily high (as far as 1 if the mean is far from the means of the other classes), the softmax output uncertainty can be as large as the entire space. This signifies the model's uncertainty in its softmax output value—i.e. in the prediction. In this scenario it would not be reasonable to use argmax to return class 5 for the middle image when its uncertainty is so high. One would expect the model to ask an external annotator for a label for this input.

4.3 Quantitative comparison

We next assess the models' confidence quantitatively to see how much we sacrifice in our attempt at deriving *practical* inference techniques in Bayesian NNs. We compare the Bernoulli approximating distribution (corresponding to dropout) to two existing inference methods: [Graves, 2011] and [Hernandez-Lobato and Adams, 2015]. Quite surprisingly, we show that by using dropout's uncertainty we can obtain a considerable improvement in predictive log-likelihood and root mean square error (RMSE) compared to these techniques.

Predictive log-likelihood captures how well a model fits the data, with larger values indicating better model fit. Uncertainty quality can be determined from this quantity as well. We replicate the experiment set-up in Hernandez-Lobato and Adams [2015] and compare the RMSE and predictive log-likelihood of dropout (referred to as "Dropout" in the experiments) to that of Probabilistic Back-propagation (referred to as "PBP", [Hernandez-Lobato and Adams, 2015]) and to a popular variational inference technique in Bayesian NNs (referred to as "VI", [Graves, 2011]). The aim of this experiment is to compare the uncertainty quality obtained from a *naive* application of dropout in NNs to that of specialised methods developed to capture uncertainty.

Following our Bayesian interpretation of dropout we need to define a prior length-scale, and find an optimal model precision parameter τ which will allow us to evaluate the predictive log-likelihood (eq. (3.18)). Similar to [Hernandez-Lobato and Adams, 2015] we use Bayesian optimisation (BO, [Snoek et al., 2012, 2015]) over validation log-likelihood to find optimal τ, and set the prior length-scale to 10^{-2} for most datasets based on the

range of the data. Note that this is a standard dropout NN, where the prior length-scale l and model precision τ are simply used to define the model's weight decay through eq. (3.17). We used dropout with probabilities 0.05 and 0.005 since the network size is very small (with 50 units following [Hernandez-Lobato and Adams, 2015]) and the datasets are fairly small as well. The BO runs used 40 iterations following the original setup, but after finding the optimal parameter values we used 10x more iterations, as dropout takes longer to converge. Even though the model doesn't converge within 40 iterations, it gives BO a good indication of whether a parameter is good or not. Finally, we used mini-batches of size 32 and the Adam optimiser [Kingma and Ba, 2014]. Further details about the various datasets are given in [Hernandez-Lobato and Adams, 2015].

The results are shown in table 4.1. Dropout significantly outperforms all other models both in terms of RMSE as well as test log-likelihood on all datasets apart from Yacht, for which PBP obtains better RMSE. All experiments were averaged on 20 random splits of the data (apart from Protein for which only 5 splits were used and Year for which one split was used). It is interesting to note that the median for most datasets gives much better performance than the mean. For example, on the Boston Housing dataset dropout achieves median RMSE of 2.68 with an IQR interval of [2.45, 3.35] (compared to mean 2.97) and predictive log-likelihood median of -2.34 with IQR [-2.54, -2.29] (compared to mean -2.46). In the Concrete Strength dataset dropout achieves median RMSE of 5.15 (compared to mean 5.23)[3].

To implement the model we used Keras [Chollet, 2015], an open source deep learning package based on Theano [Bergstra et al., 2010]. In [Hernandez-Lobato and Adams, 2015] BO for VI seems to require a considerable amount of additional time compared to PBP. However our model's running time (including BO) is comparable to PBP's Theano implementation. On the Naval Propulsion dataset for example our model takes 276 seconds on average per split (start-to-finish, divided by the number of splits). Out of that, with the optimal parameters BO found, model training took 95 seconds. This is in comparison to PBP's 220 seconds. For Kin8nm our model requires 188 seconds on average including BO, 65 seconds without, compared to PBP's 156 seconds.

Dropout's RMSE in table 4.1 is given by averaging stochastic forward passes through the network following eq. (3.16) (MC dropout). We observed an improvement using this estimate compared to the standard dropout weight averaging, and also compared to much smaller dropout probabilities (near zero). For the Boston Housing dataset for example, repeating the same experiment with dropout probability 0 results in RMSE of

[3]Full raw results and code are available online at `https://github.com/yaringal/DropoutUncertaintyExps`.

			Avg. Test RMSE and Std. Errors			Avg. Test LL and Std. Errors		
Dataset	N	Q	**VI**	**PBP**	**Dropout**	**VI**	**PBP**	**Dropout**
Boston Housing	506	13	4.32 ±0.29	3.01 ±0.18	**2.97 ±0.19**	-2.90 ±0.07	-2.57 ±0.09	**-2.46 ±0.06**
Concrete Strength	1,030	8	7.19 ±0.12	5.67 ±0.09	**5.23 ±0.12**	-3.39 ±0.02	-3.16 ±0.02	**-3.04 ±0.02**
Energy Efficiency	768	8	2.65 ±0.08	1.80 ±0.05	**1.66 ±0.04**	-2.39 ±0.03	-2.04 ±0.02	**-1.99 ±0.02**
Kin8nm	8,192	8	**0.10 ±0.00**	**0.10 ±0.00**	**0.10 ±0.00**	0.90 ±0.01	0.90 ±0.01	**0.95 ±0.01**
Naval Propulsion	11,934	16	**0.01 ±0.00**	**0.01 ±0.00**	**0.01 ±0.00**	3.73 ±0.12	3.73 ±0.01	**3.80 ±0.01**
Power Plant	9,568	4	4.33 ±0.04	4.12 ±0.03	**4.02 ±0.04**	-2.89 ±0.01	-2.84 ±0.01	**-2.80 ±0.01**
Protein Structure	45,730	9	4.84 ±0.03	4.73 ±0.01	**4.36 ±0.01**	-2.99 ±0.01	-2.97 ±0.00	**-2.89 ±0.00**
Wine Quality Red	1,599	11	0.65 ±0.01	0.64 ±0.01	**0.62 ±0.01**	-0.98 ±0.01	-0.97 ±0.01	**-0.93 ±0.01**
Yacht Hydrodynamics	308	6	6.89 ±0.67	**1.02 ±0.05**	1.11 ±0.09	-3.43 ±0.16	-1.63 ±0.02	**-1.55 ±0.03**
Year Prediction MSD	515,345	90	9.034 ±NA	8.879 ±NA	**8.849 ±NA**	-3.622 ±NA	-3.603 ±NA	**-3.588 ±NA**

Table 4.1 **Average test performance in RMSE and predictive log likelihood** for a popular variational inference method (**VI**, Graves [2011]), Probabilistic back-propagation (**PBP**, Hernandez-Lobato and Adams [2015]), and dropout uncertainty (**Dropout**). Dataset size (N) and input dimensionality (Q) are also given.

	Avg. Test RMSE and Std. Errors			Avg. Test LL and Std. Errors		
Dataset	**Dropout**	**10x Epochs**	**2 Layers**	**Dropout**	**10x Epochs**	**2 Layers**
Boston Housing	2.97 ±0.19	2.80 ±0.19	2.80 ±0.13	-2.46 ±0.06	-2.39 ±0.05	-2.34 ±0.02
Concrete Strength	5.23 ±0.12	4.81 ±0.14	4.50 ±0.18	-3.04 ±0.02	-2.94 ±0.02	-2.82 ±0.02
Energy Efficiency	1.66 ±0.04	1.09 ±0.05	0.47 ±0.01	-1.99 ±0.02	-1.72 ±0.02	-1.48 ±0.00
Kin8nm	0.10 ±0.00	0.09 ±0.00	0.08 ±0.00	0.95 ±0.01	0.97 ±0.01	1.10 ±0.00
Naval Propulsion	0.01 ±0.00	0.00 ±0.00	0.00 ±0.00	3.80 ±0.01	3.92 ±0.01	4.32 ±0.00
Power Plant	4.02 ±0.04	4.00 ±0.04	3.63 ±0.04	-2.80 ±0.01	-2.79 ±0.01	-2.67 ±0.01
Protein Structure	4.36 ±0.01	4.27 ±0.01	3.62 ±0.01	-2.89 ±0.00	-2.87 ±0.00	-2.70 ±0.00
Wine Quality Red	0.62 ±0.01	0.61 ±0.01	0.60 ±0.01	-0.93 ±0.01	-0.92 ±0.01	-0.90 ±0.01
Yacht Hydrodynamics	1.11 ±0.09	0.72 ±0.06	0.66 ±0.06	-1.55 ±0.03	-1.38 ±0.01	-1.37 ±0.02

Table 4.2 **Average test performance in RMSE and predictive log likelihood** for dropout uncertainty as above (**Dropout**), the same model optimised with 10 times the number of epochs and identical model precision (**10x epochs**), and the same model again with 2 layers instead of 1 (**2 Layers**).

3.07 and predictive log-likelihood of -2.59. This demonstrates that dropout significantly affects the predictive log-likelihood and RMSE, even though the dropout probability is fairly small.

Remark. We used dropout following the same way the method would be used in current research—without adapting model structure and without optimising the objective using a derivative w.r.t. the weight decay[a]. We used this standard methodology to demonstrate the results that could be obtained from existing models, with the only change being the MC dropout evaluation. Further, in the results above we attempted to match PBP's run time (hence used only 10x more epochs compared to PBP's 40 epochs). Experimenting with 100x more epochs compared to PBP (10x more epochs compared to the results in table 4.1) gives a considerable improvement both in terms of test RMSE as well as test log-likelihood over the results in table 4.1. We further assessed a model with the same number of units and 2 hidden layers instead of 1. Both experiments are shown in table 4.2. Experimenting with different network architectures we expect the method to give further improved uncertainty estimates.

It is also important to mention the results collected by Bui et al. [2016] extending on the above. Bui et al. [2016] have repeated the experiment setup above, evaluating many more models including the Gaussian process, deep Gaussian processes, and Bayesian neural networks trained with SGLD [Welling and Teh, 2011] as well as HMC [Neal, 1995]. Bui et al. [2016] found[b] SGLD's performance to be similar to dropout in table 4.1, with SGLD outperforming dropout on 5 out of the 10 datasets, and dropout outperforming SGLD on the other 5. HMC seemed to supersede all other techniques on average (although both SGLD and HMC require much longer run-times). I would mention that these experiments were done on single hidden layer NNs with 50 units, and the results might not be the same with larger models.

[a] Note that in the Gaussian processes literature the objective would be optimised w.r.t. the model precision τ, corresponding to our NN's weight decay through eq. (3.17). In the deep learning literature though the weight decay would often be grid-searched over to minimise validation error. We repeated this standard approach here. However to be more efficient we used Bayesian optimisation to determine the weight decay instead of searching over a grid. Alternatively, the objective could be optimised w.r.t. the model precision τ as in the GP case, which we demonstrate in section §4.6.

[b] Note that our reported dropout standard error was erroneously scaled-up by a factor of 4.5 at the time of publication (i.e. for Boston RMSE we reported standard error 0.85 instead of 0.19). The erroneous results are the ones used in [Bui et al., 2016].

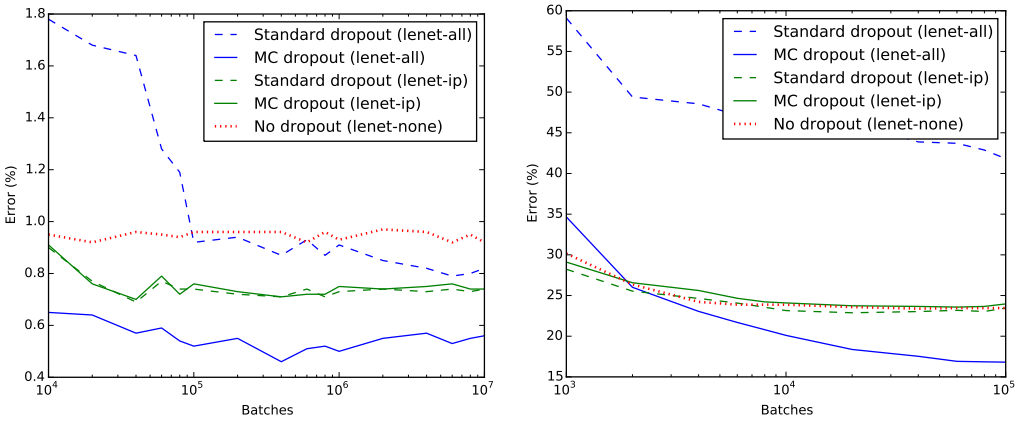
4.4 Bayesian convolutional neural networks

We give further empirical evaluation with specialised model structures—Bayesian convolutional neural networks (CNNs). This extends on the experiments above where dropout was applied only after the inner-product layers of a CNN. In comparison, here we perform dropout after all convolution and weight layers, viewed as approximate inference in a Bayesian CNN following the results of the previous chapter. We assess the LeNet network structure [LeCun et al., 1998] on MNIST [LeCun and Cortes, 1998] and CIFAR-10 [Krizhevsky and Hinton, 2009] with different settings, and show improvement in test accuracy compared to existing techniques in the literature. We evaluate the number of samples needed to obtain an improvement in results using MC dropout (eq. (3.8)), and finish with improved results on CIFAR-10 obtained by an almost trivial change of an existing model. All experiments were done using the Caffe framework [Jia et al., 2014], requiring identical training time to that of standard CNNs[4].

In this section we refer to our Bayesian CNN implementation with dropout used after every parameter layer as "**lenet-all**". We compare this model to a CNN with dropout used after the inner-product layers at the end of the network alone—the traditional use of dropout in the literature. We refer to this model as "**lenet-ip**". Additionally we compare to LeNet as described originally in [LeCun et al., 1998] with no dropout at all, referred to as "**lenet-none**". We evaluate each dropout network structure (lenet-all and lenet-ip) using two testing techniques. The first is using weight averaging, the standard way dropout is used in the literature (referred to as "**Standard dropout**"). We use the Caffe [Jia et al., 2014] reference implementation for this. The second testing technique interleaves Bayesian methodology into deep learning. We average T stochastic forward passes through the model following the Bayesian interpretation of dropout derived in eq. (3.8). This technique is referred to as "**MC dropout**". In this experiment we average $T = 50$ forward passes through the network.

Figure 4.10 shows classification error as a function of batches *on log scale* for all three models (lenet-all, lenet-ip, and lenet-none) with the two different testing techniques (Standard dropout and MC dropout) for MNIST (fig. 4.10a) and CIFAR-10 (fig. 4.10b). It seems that Standard dropout in lenet-ip results in improved results compared to lenet-none, with the results more pronounced on the MNIST dataset than on CIFAR-10. When the Standard dropout testing technique is used with our Bayesian CNN (with dropout applied after every parameter layer—lenet-all) performance suffers. However, by averaging the forward passes of the same network—using the same network weights—the

[4]The configuration files are available online at
https://github.com/yaringal/DropoutUncertaintyCaffeModels.



(a) MNIST (b) CIFAR-10

Fig. 4.10 **Test error for LeNet with dropout applied after every weight layer (lenet-all—our Bayesian CNN implementation, blue), dropout applied after the fully connected layer alone (lenet-ip, green), and without dropout (lenet-none, dotted red line).** Standard dropout is shown with a dashed line, MC dropout is shown with a solid line. Note that although Standard dropout lenet-all performs very badly on both datasets (dashed blue line), when evaluating *the same network* with MC dropout (solid blue line) the model outperforms all others.

performance of lenet-all supersedes that of all other models ("MC dropout lenet-all" in both 4.10a and 4.10b).

Most existing CNN literature uses Standard dropout after the fully-connected layers alone, equivalent to "Standard dropout lenet-ip" in our experiment [Krizhevsky et al., 2012]. Srivastava et al. [2014] claim, based on empirical observations, that Standard dropout gives very close results to MC dropout in standard NN architectures, but the claim was not verified with CNN architectures. Dropout is not used in CNNs after convolution layers in existing literature perhaps because empirical results with Standard dropout suggested deteriorated performance (as can also be seen in our experiments). Standard dropout approximates model output during test time by propagating the mean of each layer to the next. However this can be a crude approximation to the predictive mean of the model, which can otherwise be obtained by Monte Carlo averaging of stochastic forward passes through the model (eq. (3.8)). The empirical results given in Srivastava et al. [2014, section 7.5] suggested that Standard dropout is equivalent to MC dropout, and it seems that most research has followed this approximation. However the results we obtained in our experiments suggest that the Standard dropout approximation can fail in some model architectures.

4.4.1 Model over-fitting

We evaluate the models' tendency to over-fit on training sets decreasing in size. We use the same experiment set-up as above, without changing the dropout ratio for smaller datasets and without increasing model weight decay. We randomly split the MNIST dataset into smaller training sets of sizes 1/4 and 1/32 fractions of the full set. We evaluated the lenet-all model with MC dropout compared to lenet-ip with Standard dropout—the standard approach in the field. We did not compare to lenet-none as it is known to over-fit even on the full MNIST dataset.

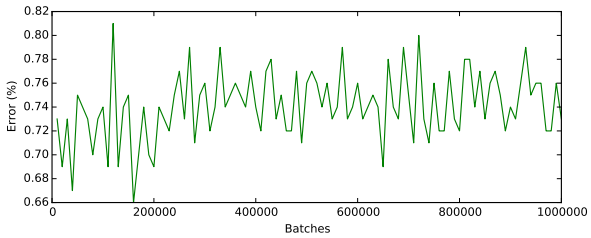
The results are shown in fig. 4.11. For the entire MNIST dataset (figs. 4.11a and 4.11b) none of the models seem to over-fit (with lenet-ip performing worse than lenet-all). It seems that even for a quarter of the MNIST dataset (15,000 data points) the Standard dropout technique starts over-fitting (fig. 4.11c). In comparison lenet-all performs well on this dataset (obtaining better classification accuracy than the best result of Standard dropout on lenet-ip). When using a smaller dataset with 1,875 training examples it seems that both techniques over-fit, and other forms of regularisation are needed.

4.4.2 MC dropout in standard convolutional neural networks

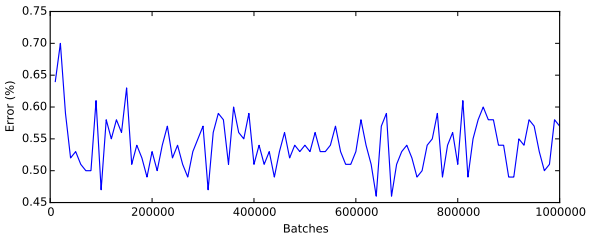
We next evaluate the use of Standard dropout compared to MC dropout on existing well known[5] CNN architectures in the literature. We evaluated two well known models that have achieved state-of-the-art results on CIFAR-10 in the past several years. The first is Network in network (NIN) [Lin et al., 2013]. The model was extended by [Lee et al., 2014] who added multiple loss functions after some of the layers—in effect "encouraging" the bottom layers to explain the data better. The new model was named a Deeply supervised network (DSN). The same idea was used in [Szegedy et al., 2014] to achieve state-of-the-art results on ImageNet.

We assessed these models on the CIFAR-10 dataset, as well as on an augmented version of the dataset for the DSN model [Lee et al., 2014]. We replicated the experiment set-up as it appeared in the original papers, and evaluated the models' test error using Standard dropout as well as using MC dropout, averaging $T = 100$ forward passes. MC dropout testing gives us a noisy estimate, with potentially different test results over different runs. To get faithful results one would need to repeat each experiment several times to get a mean and standard deviation for the test error. We therefore repeated the experiment 5 times and report the average test error. In table 4.3 we report predictive mean, as well as the standard deviation resulting from the multiple repetitions

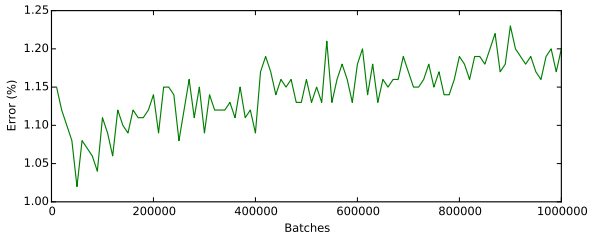
[5] Using http://rodrigob.github.io as a reference.



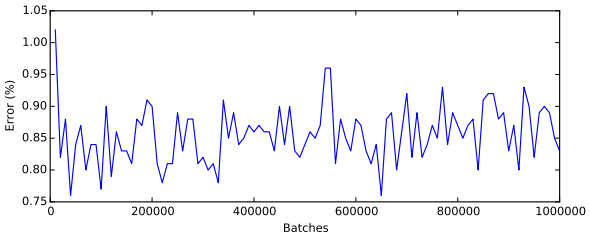
(a) Entire MNIST, Standard dropout+lenet-ip



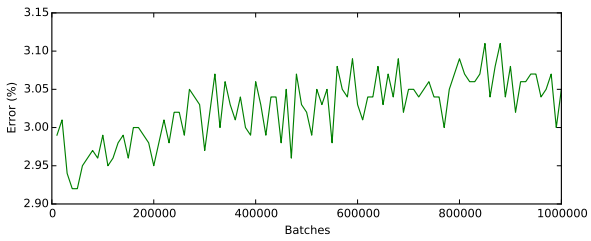
(b) Entire MNIST, MC dropout+lenet-all



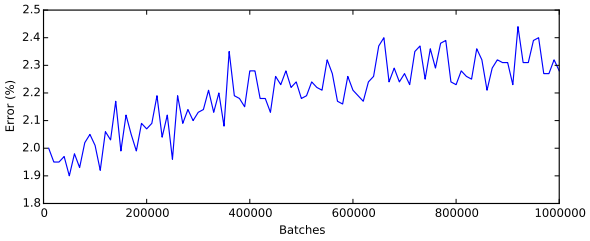
(c) 1/4 of MNIST, Standard dropout+lenet-ip



(d) 1/4 of MNIST, MC dropout+lenet-all



(e) 1/32 of MNIST, Standard dropout+lenet-ip



(f) 1/32 of MNIST, MC dropout+lenet-all

Fig. 4.11 **Test error of LeNet trained on random subsets of MNIST decreasing in size.** To the left in green are networks with dropout applied after the last layer alone (lenet-ip) and evaluated with Standard dropout (the standard approach in the field), to the right in blue are networks with dropout applied after every weight layer (lenet-all) and evaluated with MC dropout—our Bayesian CNN implementation. Lenet-ip starts over-fitting even with a quarter of the dataset (also, note the different Y-axis scales). With a small enough dataset, both models over-fit. MC dropout was used with 10 samples.

	CIFAR Test Error (and Std.)	
Model	**Standard Dropout**	**MC Dropout**
NIN	10.43	$\mathbf{10.27 \pm 0.05}$
DSN	9.37	$\mathbf{9.32 \pm 0.02}$
Augmented-DSN	7.95	$\mathbf{7.71 \pm 0.09}$

Table 4.3 **Test error on CIFAR-10 with the same networks (model structure and weights) evaluated using both Standard dropout as well as MC dropout** ($T = 100$, averaged with 5 repetitions and given with standard deviation). MC dropout achieves consistent improvement in test error compared to Standard dropout.

of the experiment to see if the improvement is statistically significant. It seems that MC dropout results in a statistically significant improvement for all three models (NIN, DSN, and Augmented-DSN), with the largest increase for Augmented-DSN.

Remark. It is interesting to note that we observed no improvement on ImageNet [Deng et al., 2009] using the same models. From initial experimentation, it seems that the large number of classes in ImageNet results in a predictive mean for the true class which is only slightly higher than the other softmax outputs. As a result, the ranges of high uncertainty overlap considerably and MC dropout does not perform well.

4.4.3 MC estimate convergence

Lastly, we assessed the usefulness of the proposed method in practice for applications in which efficiency during test time is important. We give empirical results suggesting that 20 samples are enough to improve performance on some datasets. We evaluated the last model (Augmented-DSN) with MC dropout for $T = 1, ..., 100$ samples. We repeated the experiment 5 times and averaged the results. In fig. 4.12 we see that within 20 samples the error is reduced by more than one standard deviation. Within 100 samples the error converges to 7.71.

This replicates the experiment in [Srivastava et al., 2014, section 7.5], here with the augmented CIFAR-10 dataset and the DSN CNN model. Compared to [Srivastava et al., 2014, section 7.5] we obtained a significant reduction in test error. This might be because CNNs exhibit different characteristics from standard NNs. We speculate that the non-linear pooling layer affects the dropout approximation considerably.

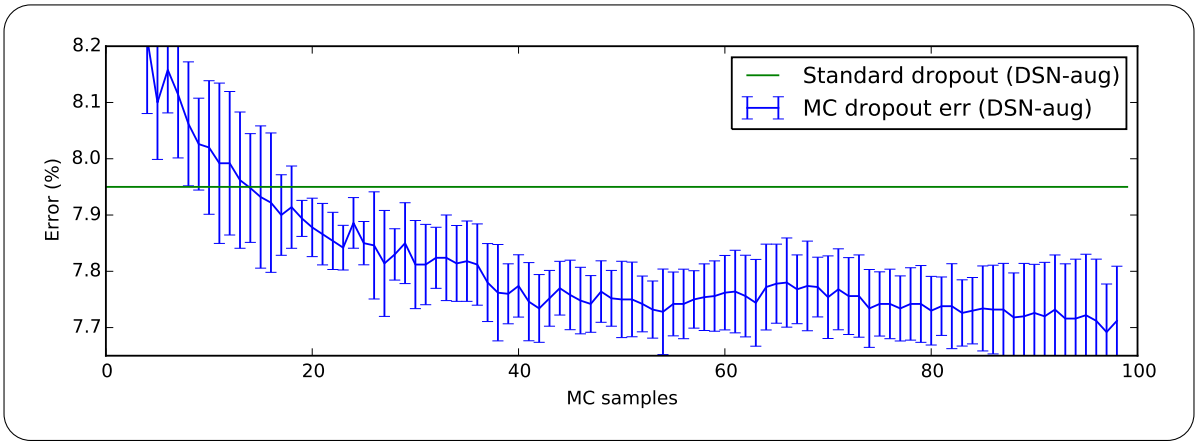


Fig. 4.12 **Augmented-DSN test error for different number of averaged forward passes in MC dropout** (blue) averaged with 5 repetitions, shown with 1 standard deviation. In green is test error with Standard dropout. MC dropout achieves a significant improvement (more than 1 standard deviation) after 20 samples.

4.5 Recurrent neural networks

Extending on the work above with Bayesian CNNs, next we assess a second specialised NN model—the Bayesian recurrent neural network (RNN). Here we take a different approach though. We start with the Bayesian model developed in §3.4.2 and derive a new SRT, or more specifically derive a new dropout variant. This is an interesting use of the ideas above; existing literature in RNNs has established that dropout cannot be applied in RNNs apart from the forward connections [Bayer et al., 2013; Bluche et al., 2015; Pachitariu and Sahani, 2013; Pham et al., 2014; Zaremba et al., 2014]. In these papers different network units are dropped at different time steps in the forward connections, and no dropout is applied to the recurrent connections. However, based on the developments in the previous chapter we propose a new dropout variant that can be successfully applied in RNNs, and assess the model's predictive mean.

We replicate the language modelling experiment of [Zaremba et al., 2014]. The experiment uses the Penn Treebank, a standard benchmark in the natural language processing field. This dataset is considered to be a small one in the community, with $887,521$ tokens (words) in total, making overfitting a considerable concern. Throughout the experiments we refer to LSTMs with the dropout technique proposed following our Bayesian interpretation in §3.4.2 as *Variational LSTMs*, and refer to existing dropout techniques as *naive dropout LSTMs* (different masks at different steps, applied to the input and output of the LSTM alone). We refer to LSTMs with no dropout as *standard LSTMs*.

	Medium LSTM			Large LSTM		
	Validation	Test	WPS	Validation	Test	WPS
Non-regularized (early stopping)	121.1	121.7	5.5K	128.3	127.4	2.5K
Zaremba et al. [2014]	86.2	82.7	5.5K	82.2	78.4	2.5K
Variational (tied weights)	81.8 ± 0.2	79.7 ± 0.1	4.7K	77.3 ± 0.2	75.0 ± 0.1	2.4K
Variational (tied weights, MC)	–	79.0 ± 0.1	–	–	74.1 ± 0.0	–
Variational (untied weights)	81.9 ± 0.2	79.7 ± 0.1	2.7K	77.9 ± 0.3	75.2 ± 0.2	1.6K
Variational (untied weights, MC)	–	$\mathbf{78.6 \pm 0.1}$	–	–	$\mathbf{73.4 \pm 0.0}$	–

Table 4.4 Single model perplexity (on test and validation sets) for the Penn Treebank language modelling task. Two model sizes are compared (a medium and a large LSTM, following [Zaremba et al., 2014]'s setup), with number of processed words per second (WPS) reported. Both dropout approximation and MC dropout are given for the test set with the Variational model. A common approach for regularisation is to reduce model complexity (necessary with the non-regularised LSTM). With the Variational models however, a significant reduction in perplexity is achieved by using larger models.

We implemented a Variational LSTM for both the medium model of [Zaremba et al., 2014] (2 layers with 650 units in each layer) as well as their large model (2 layers with 1500 units in each layer)[6]. The only changes we have made to Zaremba et al. [2014]'s setting are 1) using our proposed dropout variant instead of naive dropout, and 2) tuning weight decay (which was chosen to be zero in [Zaremba et al., 2014]). All other hyper-parameters are kept identical to [Zaremba et al., 2014]: learning rate decay was not tuned for our setting and is used following Zaremba et al. [2014]. Dropout parameters were optimised with grid search (tying the dropout probability over the embeddings together with the one over the recurrent layers, and tying the dropout probability for the inputs and outputs together as well). These are chosen to minimise validation perplexity[7].

Our results are given in table 4.4. For the variational LSTM we give results using both the tied weights model (eq. (3.23), *Variational (tied weights)*), and without weight tying (eq. (3.22), *Variational (untied weights)*). For each model we report performance using both the standard dropout approximation and using MC dropout (obtained by performing dropout at test time 1000 times, denoted *MC*). For each model we report average perplexity and standard deviation (each experiment was repeated 3 times with

[6]Full raw results and code are available online at `https://github.com/yaringal/BayesianRNN`.

[7]Optimal probabilities are 0.3 and 0.5 respectively for the large model, compared [Zaremba et al., 2014]'s 0.6 dropout probability, and 0.2 and 0.35 respectively for the medium model, compared [Zaremba et al., 2014]'s 0.5 dropout probability.

different random seeds and the results were averaged). Model training time is given in *words per second* (WPS).

It is interesting that using the dropout approximation, weight tying results in lower validation error and test error than the untied weights model. But with MC dropout the untied weights model performs much better. Validation perplexity for the large model is improved from [Zaremba et al., 2014]'s 82.2 down to 77.3 (with weight tying), or 77.9 without weight tying. Test perplexity is reduced from 78.4 down to 73.4 (with MC dropout and untied weights).

Comparing our results to the non-regularised LSTM (evaluated with early stopping, giving similar performance as the early stopping experiment in [Zaremba et al., 2014]) we see that for either model size an improvement can be obtained by using our dropout variant. Comparing the medium sized Variational model to the large one we see that a significant reduction in perplexity can be achieved by using a larger model. This cannot be done with the non-regularised LSTM, where a larger model leads to worse results. This shows that reducing the complexity of the model, a possible approach to avoid overfitting, actually leads to a worse fit when using dropout.

We also see that the tied weights model achieves very close performance to that of the untied weights one when using the dropout approximation. Assessing model run time though (on a Titan X GPU), we see that tying the weights results in a more time-efficient implementation. This is because the single matrix product is implemented as a single GPU kernel, instead of the four smaller matrix products used in the untied weights model (where four GPU kernels are called sequentially). Note though that a low level implementation should give similar run times.

4.6 Heteroscedastic uncertainty

We finish this chapter with a discussion of homoscedastic versus heteroscedastic aleatoric uncertainty. Homoscedastic regression assumes identical observation noise for every input point $\mathbf{x}$. Heteroscedastic regression, on the other hand, assumes that observation noise can vary with input $\mathbf{x}$ [Le et al., 2005]. Heteroscedastic models are useful in cases where parts of the observation space might have higher noise levels than others.

Using the developments in the previous chapter we obtained models with homoscedastic aleatoric uncertainty. This can be seen from the model definition in eq. (2.1). The likelihood in our derivations is defined as $\mathbf{y}_i \sim N(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i), \tau^{-1} I)$ with $\mathbf{f}^{\boldsymbol{\omega}}(\cdot)$ the network output, dependent on the weights random variable $\boldsymbol{\omega}$. Here our model precision τ (which

is the same as the inverse observation noise) is a constant, which has to be tuned for the data.

We can easily adapt the model to obtain data-dependent noise. This simply involves making τ into a function of the data, very much like $\mathbf{f}^{\boldsymbol{\omega}}(\cdot)$ is a function of the data. A practical way to obtain this is to tie the two functions together, splitting the top layers of a network between predictive mean $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x})$ and model precision $\mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x})$ (of course we would want to re-parametrise this to make sure $\mathbf{g}^{\boldsymbol{\omega}}(\cdot)$ is positive, and in the multivariate case to make sure the precision matrix $\mathbf{g}^{\boldsymbol{\omega}}(\cdot)$ is positive definite). Thus the new (now heteroscedastic!) model likelihood is given by $\mathbf{y}_i \sim \mathcal{N}(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i), \mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x}_i)^{-1})$. We put a prior over the weights used for the precision as well. The prior length-scale for these weights (equivalently, weight decay) controls the precision smoothness. A long prior length-scale for the precision weights would correspond to a slowly varying precision for example.

We can implement this new model by adapting the loss function of the original model, changing eq. (1.1) to:

$$\begin{aligned} E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x},\mathbf{y}) &:= \frac{1}{2}\Big(\mathbf{y} - \mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\Big)\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\Big(\mathbf{y} - \mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\Big)^T \\ &\quad - \frac{1}{2}\log\det\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x}) + \frac{D}{2}\log 2\pi \\ &= -\log\mathcal{N}(\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i), \mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x}_i)^{-1}) \end{aligned}$$

with D the output dimensionality[8].

We estimate our predictive variance like before by averaging stochastic forward passes through the model, both for $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x})$ and for $\mathbf{g}^{\boldsymbol{\omega}}(\mathbf{x})$. We use the unbiased estimator:

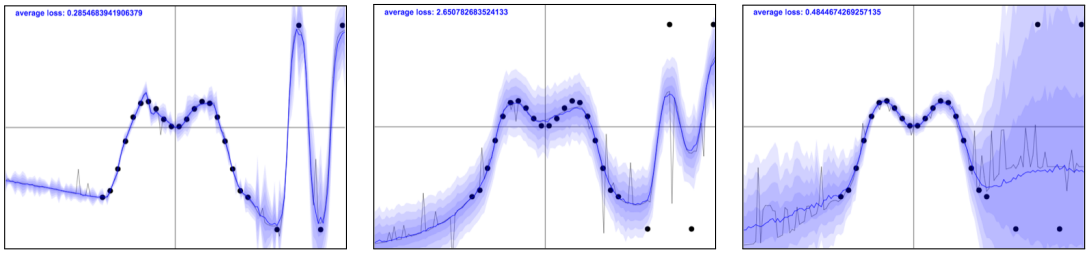
$$\begin{aligned} \widetilde{\mathrm{Var}}[\mathbf{y}^*] &:= \frac{1}{T}\sum_{t=1}^{T}\mathbf{g}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x})\mathbf{I} + \mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*)^T\mathbf{f}^{\widehat{\boldsymbol{\omega}}_t}(\mathbf{x}^*) - \widetilde{\mathbb{E}}[\mathbf{y}^*]^T\widetilde{\mathbb{E}}[\mathbf{y}^*] \\ &\xrightarrow[T\to\infty]{} \mathrm{Var}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] \end{aligned}$$

which equals the sample variance of T stochastic forward passes through the NN plus the averaged inverse model precision.

Predictive uncertainty for the heteroscedastic model compared to the homoscedastic model[9] (both with large fixed observation noise as well as small fixed observation noise) is shown in fig. 4.13.

[8] In the multivariate output case it is convenient to assume $\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})$ to be a diagonal precision matrix, in which case the log determinant would reduce to a sum of the logs over each element of $\mathbf{g}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})$.

[9] Code is available online at
https://github.com/yaringal/HeteroscedasticDropoutUncertainty.



(a) Homoscedastic model with small observation noise.

(b) Homoscedastic model with large observation noise.

(c) Heteroscedastic model with data-dependent observation noise.

Fig. 4.13 Homoscedastic and heteroscedastic uncertainty with Bernoulli approximating distributions (dropout). Here $N = 24$ data points are generated from the scalar function $y = x\sin(x)$ with no observation noise, and 4 points with large noise are added on the right-hand side. The homoscedastic models use $l^2 = 10$ and $p = 0.005$, with $\tau^{-1} = 0$ for sub-figure 4.13a and $\tau = 1$ for sub-figure 4.13b. The heteroscedastic model (sub-figure 4.13c) uses $l^2 = 0.1$ and $p = 0.05$.

This experiment concludes our uncertainty quality assessment. We next examine real-world applications relying on the developments above.