MLSALT4 Paper Replication Exercise: Concrete Dropout

Team 5

June 7, 2018

Abstract

Concrete Dropout is a new dropout variant which learns the dropout probability as part of the training objective, instead of requiring hyper-parameter grid search over the dropout probability as in traditional dropout technique. In this writeup, we first review background and theory governing uncertainty estimation, the variational interpretation of dropout, and learning the dropout probability. We then replicate the results from the original Concrete Dropout paper [GHK17], namely the experiments on synthetic data, the UCI dataset, and the MNIST dataset. Due to limitations in computing and data access, we did not replicate the computer vision and reinforcement learning results from the original paper. Finally, we implemented an extension of Concrete Dropout in line with the variational interpretation of recurrent neural networks.

1 Introduction

Applying dropout to neural networks is a method of obtaining uncertainty estimates in model predictions, which is essential to many types of machine learning tasks. However, a good estimate of uncertainty requires a grid-search over dropout probabilities. In practice, performing manual tuning over dropout probabilities is extremely expensive: very similar models must be trained and evaluated with slight variations in dropout, thus wasting time and computing resources. As models increase in size, this grid-search approach is infeasible as each layer in the model can have a different dropout parameter, causing an exponential increase in the search space as the model becomes deeper. Finally, for applications such as reinforcement learning, in which data is collected in an online manner, the model should become more certain as more data arrives. Manual tuning of dropout is infeasible in this case, as the model will need to be retrained on the entire dataset every time new data arrives to obtain an updated estimate of dropout probability.

Given these limitations in the standard grid-search approach in tuning dropout, we selected this paper to learn about how dropout could be incorporated in a neural network as a trainable parameter. This is attractive because it eliminates the need for a tedious search over dropout probabilities, and instead the optimal dropout probability is learned as the model trains. We also chose this paper to gain more experience reasoning about variational approximations and understanding how such approximations could be implemented in code. More broadly, the variational interpretation of the dropout regularization technique represents a promising avenue of research in the field of Bayesian Neural Networks, which could play a key role in the development of interpretable AI.

In this writeup, we will first summarise the theory behind the variational approximation to dropout and concrete relaxation, which yields a loss function containing the dropout probability and allows gradients to flow to this dropout parameter so that it can be learned. Next, we will show and interpret our results when replicating the figures in the original Concrete Dropout paper [GHK17]. We will then describe our progress in extending the Concrete dropout concept to recurrent network architectures.

2 Background

2.1 Dropout as a measure of uncertainty

If we are to learn dropout probability p, we ideally will have an interpretation of p that allows us to check whether the result is sensible. Gal and Ghahramani [GG16a] offered such an interpretation of dropout probability as a proxy measure of model uncertainty. Model uncertainty may be divided into *epistemic uncertainty* and *aleatoric uncertainty* [GG16a], with dropout probability corresponding specifically to a measure of *epistemic uncertainty*, as we shall explain further.

2.1.1 Two types of uncertainty

To understand what epistemic uncertainty is, we will distinguish it from aleatoric uncertainty using a linear regression example.

In Fig. 1 below, the blue dots represents the distribution of $x \rightarrow y$ mappings. Suppose an engineer samples from this distribution and wants to fit a linear model to the data (in other words, perform linear regression). The best fit line found by the engineer (in red) is just one line that fits the data. There are multiple lines that could fit the data, depending on the sample of blue dots drawn, with each line having a different probability. The black region is a distribution of possible lines produced by linear regression and illustrates the uncertainty about which model is correct when data used for regression is limited. When making predictions using the linear model, the uncertainty in y due to this uncertainty of model is called epistemic uncertainty. As the amount of data increases, uncertainty about which model is correct decreases, and therefore epistemic uncertainty also decreases.

However, even with zero epistemic uncertainty (infinite amount of data), we will never be able to infinitely precisely predict y with a linear model due to inherent noise (or perhaps unobserved variables) in the distribution. The uncertainty in y due to this "inherent noise" is called aleatoric uncertainty.

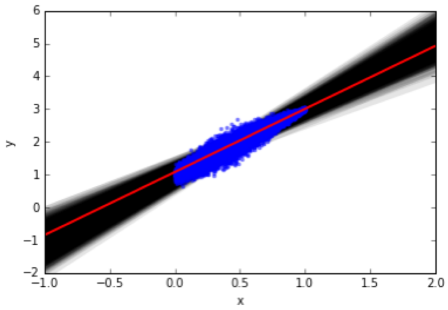


Figure 1: Epistemic uncertainty is uncertainty about which line is the true function describing the data. Aleatoric uncertainty is uncertainty due to noise in the inputs.

2.1.2 Variational interpretation of dropout

Having described the two types of uncertainty, we will now describe the variational interpretation of dropout.

When training a neural network model with dropout, a random set of nodes is dropped on each training pass. From a practitioner's perspective, this is a trick to efficiently train a bag of neural models – each with the same weights but different nodes zeroed out. This training process yields a single point estimate of these global weights.

From a probabilistic perspective, since our neural models can be parameterized by weights, models are represented as points in weight space. When no nodes are dropped, that model is represented by the furthest corner from origin of the weight-space hypercube (e.g. $[w_1, w_2, w_3 ... w_k]$). When

a node is dropped, the set of weights associated with that node is zeroed, and again this model is represented by some other corner of the same hypercube (e.g. $[0, 0, w_3...w_k]$ if weights w_1 and w_2 are zeroed). When dropout probability is small, more probability mass is concentrated on the furthest corner of the hypercube and less on the other corners.

Also defined in weight space are probability distributions over weights. Data $\mathcal{D}$ refines our prior distribution over weights $p(\mathbf{W})$ into a posterior distribution $p(\mathbf{W}|\mathcal{D})$.

Training with dropout is equivalent to fitting the discrete hypercube-like distribution to the continuous posterior distribution. We want most of the probability mass of the hypercube to coincide with where the posterior density is high. Since the hypercube distribution is parameterized by the furtherest corner, M and dropout probability p, fitting can be cast as a variational problem.

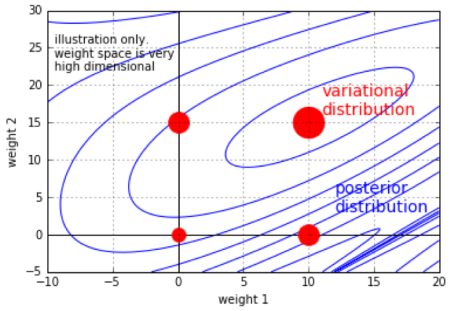


Figure 2: Dropout is a variational approximation to the posterior distribution of network weights.

If the posterior distribution has a very sharp peak – corresponding to low epistemic uncertainty – the optimal p is small so that most of the probability mass of the hypercube is concentrated on that furthest corner. But if the posterior distribution has broad peaks or ridges, or is multi-modal, the optimal p is higher to capture this spread in probability density.

This effect can also be specified mathematically: the loss function is the KL divergence between the variational distribution and the posterior distribution:

$$\mathcal{L}(\mathbf{M}, p) = KL[q_{M,p}(\mathbf{W})||p(\mathbf{W}|\mathcal{D})]$$

The variational distribution $q_{M,p}(\mathbf{W})$ is parametrized by the weight matrices prior to dropout $\mathbf{M}$ (i.e. the further corner in the weight-space hypercube), and a dropout probability p:

$$q_{M,p}(\mathbf{W}) = \sum_{z \sim Bernoulli(p)} p(z)\delta(\mathbf{W} = \omega)$$

where:

$$\omega = \mathbf{M} \cdot diag(z)$$

Essentially, the ω weight matrix is the original weight matrix $\mathbf{M}$ with certain columns zeroed out according to Bernoulli variable z, corresponding to the zeroing out of nodes during dropout.

Expanding out the loss function we obtain:

$$\mathcal{L}(\mathbf{M}, p) = KL[q_{M,p}(\mathbf{W})||p(\mathbf{W})] - \mathbb{E}_q[\log p(\mathcal{D}|W)] + \log p(\mathcal{D})$$

where the first term is the regularizer and second term is related to the likelihood.

Following [Gal16], the KL divergence between the variational distribution and the prior over weights can be approximated as:

$$KL[q_{M,p}(\mathbf{W})||p(\mathbf{W})] \approx \frac{l^2(1-p)}{2}|M|^2 - KH(p)$$

where K is the number of nodes in the layer and H is the entropy function.

The regularizer term pulls p towards larger values (up to 0.5) while the likelihood term pulls p towards 0. The more data there is, the larger the significance of the likelihood term and the smaller p will be.

2.2 Learning Dropout Probability

The core idea presented in the paper consists of including a regularization term dependent on the dropout probability p in the objective function of the model and including p in the set of model parameters to be optimized with gradient descent. Following the derivation in [Gal16], this regularization term can be approximated to be the entropy of p, which is the entropy of a Bernoulli random variable:

$$H(p) = -plog(p) - (1-p)log(1-p)$$

This is expected, since from an information theory standpoint, the KL divergence between two distributions $p(x)$ and $q(x)$ is the additional amount of information required to specify the value of x, as a result of using $q(x)$ instead of the true distribution $p(x)$ [Bis09] - and therefore in our case this additional information is represented by the entropy of the additional p variable. This entropy is maximal when $p = 0.5$ (i.e. when there is a small amount of data available and the prior dominates over the likelihood in the posterior) and minimal when $p = 0$ (i.e. when there is a larger amount of data available and the likelihood dominates the prior in the posterior). Conveniently, H is independent of the model weights, and can be omitted when the dropout probability p is not included in the objective function.

However, once we include the p dependency, we readily realize that we cannot perform gradient descent since we cannot take derivatives wrt p due to the discreteness of the Bernoulli random variable. The solution to this issue is a continuous relaxation of the Bernoulli distribution, the Concrete distribution [CJM16]. Instead of sampling from a Bernoulli distribution, we sample from a Concrete distribution with temperature t. The outcome will be a number in the interval [0,1] rather then being either 0 or 1. However, most of the probability mass is concentrated on the boundaries 0 and 1, which can be accomplished using a sigmoid transformation as shown in the following equation and in Fig. 3:

$$z = sigmoid\left(\frac{log(p) - log(1-p) + log(u) - log(1-u)}{t}\right)$$

where t is the temperature parameter, u is a uniformly distributed random variable, and p is the dropout probability. We observed that in 1-D, the Concrete distribution can be thought of as a distribution over all possible Bernoulli distributions, like the Beta distribution (with a similar extension to higher dimensions such as the Dirichlet distribution for K-dimensional categorial distributions). Despite the Concrete distribution having one extra parameter, the temperature t, compared to an equivalent Dirichlet, the two distributions have the same number of degrees of freedom. This is because the Concrete distribution imposes the constraint of all the non-temperature parameters having to sum to 1. As we will see in the following subsection, the Concrete distribution is preferred to the Dirichlet for computational cost reasons.

The temperature t controls the degree of the relaxation, or how close the Concrete distribution is to its discrete counterpart. In other words, it controls to what extent the probability mass should be shifted towards the extreme values in the [0,1] interval. For temperature $t = 0$, the probability mass is a delta function on the extremes values, letting the Concrete distribution collapse to a Bernoulli distribution in the 1-D case, or to a categorical distribution in the multi-dimensional case. Higher values of t yield a less steep sigmoid, as visualized in Fig. 3 and in the limiting case of $t \to \infty$, the sigmoid collapses to a flat line, yielding 0.5 for any value of u.

Using the Concrete Dropout objective, the temperature parameter t becomes a hyperparameter of the model, but it is valid for all layers of the network. As a result, Concrete dropout reduces the number of hyperparameters from K (one dropout probability for each layer) in standard droupout to 1. This is an ulterior advantage of using Concrete dropout over standard dropout, since by optimizing p for each layer of the network automatically, it reduces the number of tunable parameters, thus decreasing the risk of overfitting for the practitioner.

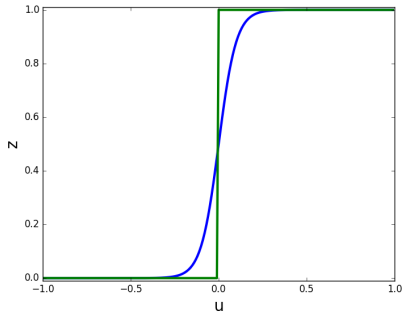


Figure 3: 1D illustration of the relationship between z and uniform noise u for a fixed dropout p in a concrete distribution (blue) and a Bernoulli distribution (green).

In all the experiments replicated and extended the temperature parameter was kept at $t = 0.1$. This is relatively close to $t = 0$ to ensure a balance between differentiability of the continuous relaxation without exceeding in the deviation from the original discrete distribution.

After relaxing the discrete distribution, we need to use the reparametrisation trick to allow gradients to reach the dropout p during backpropagation. This consists of reparametrising the stochastic binary-masked weights in standard dropout using the Concrete distribution, where the weight mask is a deterministic function of p and uniform noise u. As shown in Fig. 4, this allows us to extrapolate the randomness from the stochastic masked weights in standard dropout and redraw the architecture replacing stochastic nodes with deterministic nodes. In Concrete dropout, stochasticity is introduced by new stochastic nodes which solely depend on u and inform the mask nodes which depend deterministically on p and u. The gradient is now allowed to flow freely from the output labels to the input in the back-propagation algorithm, and consequently the parameter p can be optimized.

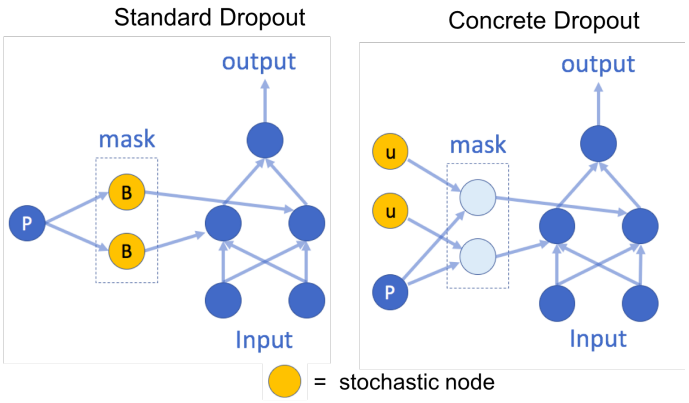


Figure 4: The reparameterisation trick moves stochasticity from the Bernoulli dropout mask into uniform noise nodes, thus allowing gradients to flow to the dropout mask.

2.3 A consideration of Dirichlet Dropout

The essence of the Concrete distribution is to relax a discrete categorical (in our case Bernoulli) distribution smoothly onto the simplex – a reparametrisation trick which allows gradients to flow through previously stochastic computation nodes and for the parameters of a discrete categorical distribution (dropout p) to be learnt.

This begs the question whether the same procedure could be done with the Dirichlet distribution with $0 < \alpha_i < 1$, which is a similar distribution over the simplex. We found the answer to be partially no because, to the best of our knowledge, a Dirichlet distribution with $0 < \alpha_i < 1$ is hard

to sample from. The following shows how to sample from a Dirichlet distribution and why the Concrete distribution is preferable.

Consider for example the binary Dirichlet distribution — the Beta distribution:

$$Beta(\alpha_1, \alpha_2) = \frac{1}{B(\alpha_1, \alpha_2)} x_1^{\alpha_1 - 1} x_2^{\alpha_2 - 1}$$

where $x_1, x_2 \in [0, 1]$, $x_1 + x_2 = 1$, $B(\alpha_1, \alpha_2) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)}{\Gamma(\alpha_1 + \alpha_2)}$, and $\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$

α_1 and α_2 need to be less than 1 for probability mass to be concentrated on the corners of the simplex. Sampling from a Beta (or Dirichlet) distribution involves first sampling from a Gamma distribution, which has a probability mass function given below:

$$Gamma(x_i; \alpha_i, 1) = \frac{x_i^{\alpha_i - 1} e^{-x_i}}{\Gamma(\alpha_i)}$$

For example, to sample from a Beta distribution, two Gamma distributed values are sampled.

$$x_1 \sim Gamma(x_1; \alpha_1, 1); \quad x_2 \sim Gamma(x_2; \alpha_2, 1)$$

They are then renormalised into a Beta distributed value.

$$\frac{x_1}{x_1 + x_2} \sim Beta(\alpha_1, \alpha_2)$$

However, the best known way to sample from a non-integer Gamma distribution involve rejection sampling [AD82]. Other sampling methods such as Inverse CDF are too computationally expensive due to the integral in the formula. While it may be possible to implement rejection sampling in a computation graph, it is not as efficient as sampling from a uniform distribution as we have done for the Concrete distribution.

3 Replicated Results

Provided with the original Concrete Dropout paper is an implementation of a Concrete Dropout layer wrapper in Keras and a demonstration of using Concrete dropout with synthetic data. We carefully worked through this code to understand its implementation, and then implemented our own pipelines for the remaining experiments.

3.1 Synthetic Data

Similar to the provided implementation, we generated synthetic data from the function $y = 2x + 8 + \epsilon$ with noise $\epsilon \sim N(0, 1)$. The training dataset ranged in size from 10 data points up to 10,000 points, and the test dataset contained 1000 points. The model contained 3 ReLU hidden layers that were 1024 units wide, and two 1D outputs: predicted mean and variance. The loss was therefore measured by $\frac{(m-y)^2}{v} + \log(v)$ where m is the predicted mean, v is the predicted variance, and y is the actual label. Each test input was evaluated on the trained model K times, to obtain K MC samples of predicted mean and variance. Then, epistemic uncertainty was measured as the variance in the predicted means across MC samples, and aleatoric uncertainty was the mean of the predicted variances across MC samples. Consistent with the original figure, we observed decreasing epistemic uncertainty as the number of data points increases. Intuitively, this suggests that model uncertainty is decreasing as more data is available. However, aleatoric uncertainty (after taking the square root) hovers around 1.0, which matches the standard deviation of the input noise. Predictive uncertainty is the sum of epistemic and aleatoric uncertainty. We also observed a trend in which dropout probability decreases as more training examples are available, and that dropout probability in the first layer is lower than that of subsequent layers. This suggests that the model learns to become confident in its transformation of the input, with remaining uncertainty in deeper layers.

For this experiment, our figures were replicated by directly running the provided implementation associated with the original paper. Interestingly, the results from this implementation do not exactly match those shown in the paper. This may be due to randomness involved in training the network (e.g. shuffling of data in each batch), as well as in generating the data samples and obtaining MC samples which leads to slightly differing results.

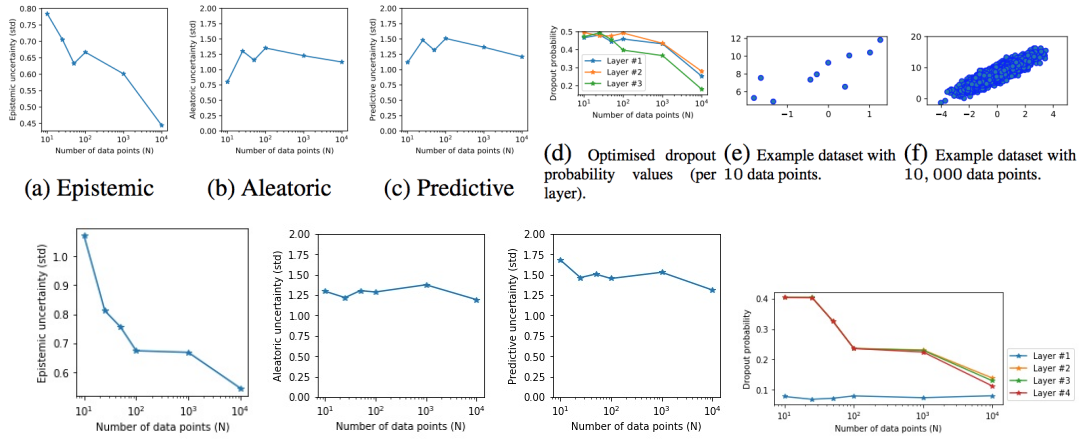


Figure 5: The original figure (top row) and our replication (bottom row) of epistemic, aleatoric, and predictive uncertainty, and dropout probability as a number of data points using synthetic data.

3.2 UCI Datasets

We proceeded in replicating the results of the Concrete dropout framework applied to UCI datasets. The UCI datasets are an interesting benchmark because they comprise a large variety of data – in terms of number of features, number of data points, and complexity of underlying structure – thus allowing experimenters to test their models in multiple settings.

The original authors were interested in comparing concrete dropout's performance on regression tasks against standard dropout and deep gaussian processes. However, the main characteristics that we were interested in testing were the consistency and stability of the optimal parameters p for the individual layers of the network, across the repetition of multiple experiments with equal settings. In other words, we wanted to verify that the optimization routine found consistent values of p to be optimal.

The network architecture is equivalent to the architecture used for the synthetic data, with 3 layers and 3 corresponding individual parameters p. The optimal values of p for the 3 layers are shown in Fig. 6, for 20 repetitions of the same experiments, in 9 different UCI datasets. The figure shows our results, compared to the original results reported in the paper. As can be seen in the plots, the optimal p parameters for the three layer seem to be consistent across repetitions of the same experiment for most experiments, with the three layers finding clearly distinct values of p.

For the Boston housing dataset we find similar results to those originally reported by the authors in terms of qualitative behavior of the p parameters, despite finding substantial quantitative differences in their individual values. Although the optimal values of p for the first layer are comparable and close to 0, the dropout probabilities p in the second and third layers differ by around 0.1 from their counterparts in the original paper. On the qualitative level, the difference resides in the higher variability in the parameter p for the second layer across multiple repetitions of the experiment.

Similar patterns are observed in the other datasets, in which the relative magnitude of p of a layer wrt the p of other layers is similar, despite the mismatch of the respective quantitative measures. In most cases, we observe similar variabilities of optimal p across repetitions of the same

experiment when comparing to the original result.

These differences likely result from different choices of parameters (temperature and weight prior lengthscale) in our experiments. However the model has shown to be robust, in terms of qualitative behavior, in finding the optimal values of the dropout probabilities p. There is a trend that p increases in deeper layers, suggesting that there could be a meaning or a possible theoretical interpretation of the relationship between the p parameters of consecutive layers in terms of propagation of uncertainties, which further research could elucidate.

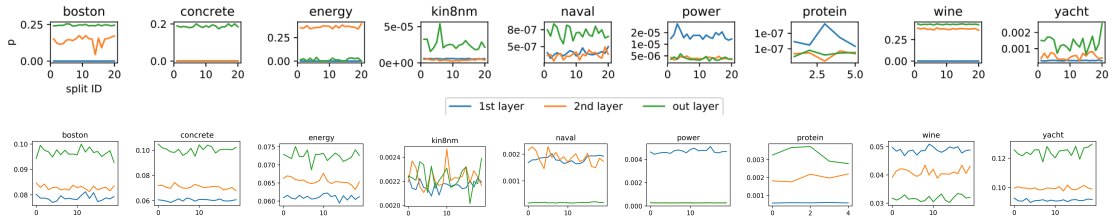


Figure 6: The original figure (top row) and our replication (bottom row) of dropout probability on UCI datasets. The blue lines correspond to the first layer, the yellow lines to the second layer and the green lines correspond to the third layer which outputs the predicted means.

3.3 MNIST

Following the model architecture described in the paper, we experimented with dropout probability as a function of number of training examples, and hidden layer dimension. For each increment in training set size or hidden layer dimension, we trained and evaluated 3 models – each a MLP with 3 hidden layers and ReLU activations – with different random initializations. The final output layer was a softmax layer over the 10 digits. We trained for 500 epochs as stated, and used a batch size of 150 (to obtain $2 \cdot 10^5$ total iterations with 60,000 training examples).

Similar to the original figure, we observe that dropout probability decreases as more data is provided. While the range of converged dropout values in the last two layers is smaller, they converge to zero in the first two layers. The dropout probability in the input layer converges to zero when enough examples are seen relative to model size, suggesting that the model becomes more confident in how to transform the input with more training points. A key difference in our replication was the behavior of the final layer, in which dropout probability decreases with increasing data, similar to the previous layers. Because we implemented this model based on what was stated from the paper, we may have used a slightly different model architecture. In particular, our final output layer was a 10-dimensional dense layer with softmax activation, and our loss function was cross entropy loss with additional weight and dropout regularization terms, but we do not know exactly how the model was implemented in the original paper. Secondly, differences could be due to our choices in parameters (prior lengthscale and temperature) and randomness in training and testing.

Next we experimented with changing the hidden layer size while consistently training on the entire training dataset. Fig. 8 shows that our results match the original figure fairly closely, but again slight differences may be due to our guesses of the model architecture, hyperparameter choice, and randomness in implementation.

4 Extension: RNNs

The results we have replicated here make the case that concrete dropout is both a principled and practically useful methodology to use with feed-forward networks performing regression and classification. An interesting follow-up question is thus whether concrete dropout can be applied to recurrent neural networks (RNNs) to similar effect. Gal and Ghahramani recently showed that dropout can be applied to RNNs at both input and recurrent connections to reduce overfitting [GG16b]. In this work, they interpret RNNs as probabilistic models and show that performing approximate variational inference in these Bayesian models, using a mixture of Gaussians as the

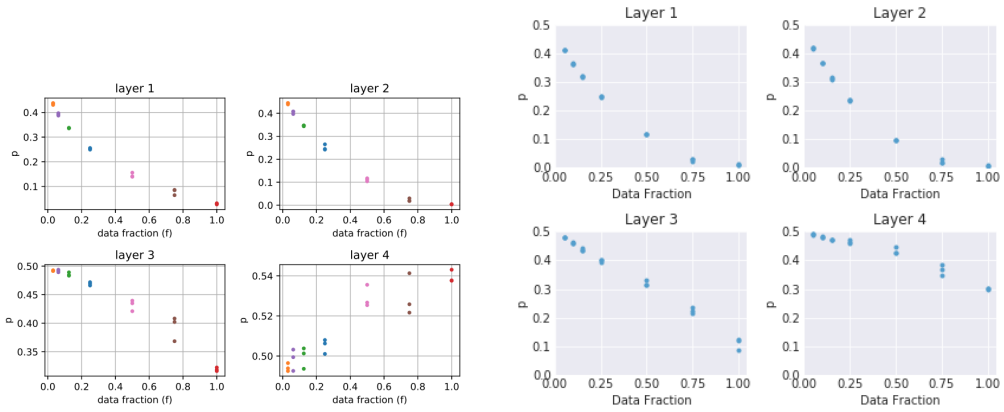


Figure 7: The original figure (left) and our replication (right) of dropout probability as a function of the number of training examples in a 3x512 MLP.

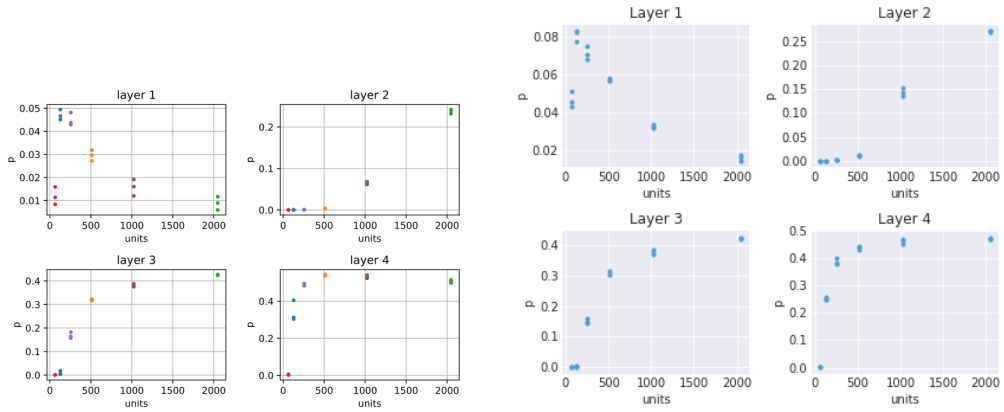


Figure 8: The original figure (left) and our replication (right) of of dropout probability as a function of the size of the hidden dimensions.

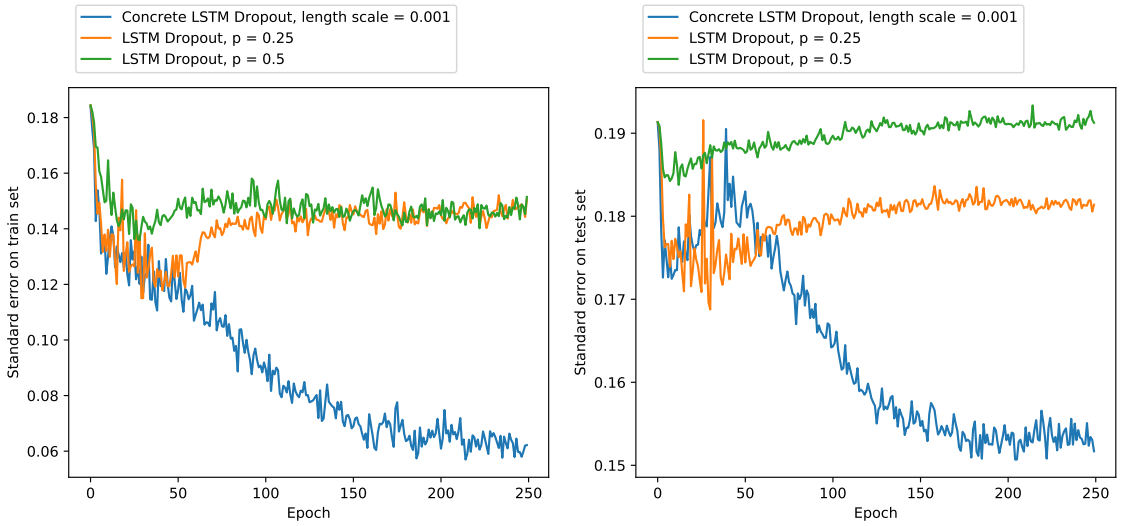
approximating posterior distribution, is identical to performing a new dropout variant in RNNs. This dropout variant repeats the same dropout mask at each time step for both inputs, outputs, and recurrent layers, rather than sampling a new dropout mask at each time step as was common before. This formulation of dropout resulted in improved performance on standard benchmark datasets and robustness to overfitting, particularly with the addition of dropout at the embedding layer.

For this extension, we applied concrete dropout to the LSTM model and evaluated this model against the LSTM with dropout model described in [GG16b]. To implement our model, we created a new Concrete Dropout LSTM layer wrapper and Concrete Dropout LSTM Cell class in Keras, which has two learnable parameters - p for input connection dropout and p_{rec} for recurrent connection dropout. In their paper, Gal and Ghahramani provided code for evaluating their LSTM on a sentiment classification task, and we adapted this code to evaluate our Concrete Dropout LSTM. The dataset is from the raw Cornell film reviews corpus collected by Pang and Lee [PL05] and composed of 5000 film reviews. Consecutive segments of T words are extracted from each review for $T = 200$, and the corresponding film score is the observed output y.

The model in [GG16b] is built from one embedding layer of dimensionality 128, one LSTM layer with 128 network units for each gate, and finally a fully connected layer applied to the last output of the LSTM, resulting in a scalar output. They use the Adam optimiser [KB14] throughout the experiments, with batch size 128, and MC dropout at test time with 10 samples. Our Concrete Dropout LSTM is built similarly, with the LSTM layer replaced with our Concrete Dropout LSTM layer. While the standard dropout LSTM requires setting hyperparameters p and p_{rec}, Concrete Dropout LSTM learns these parameters, only requiring a setting of the lengthscale hyperparameter

l. For other hyperparameters, we used the optimal settings reported in [GG16b], namely that weight decay = 0.001 and embedding layer dropout probability = 0.5.

The main results of these experiments are shown in Fig. 9, which compares Concrete Dropout LSTM against standard dropout LSTM with different dropout probabilities p (we set p_{rec} to be equal to p as in [GG16b]), and Fig. 10 which compares the performance of different hyperparameter settings for Concrete Dropout LSTM. We see from Fig. 9 that the networks all avoid overfitting, confirming the results of [GG16b]. Gal and Ghahramani report that using dropout probability $p = 0.25$ resulted in optimal performance for their LSTM dropout model, and we see this replicated in Fig. 9b. Interestingly, our Concrete Dropout LSTM seems to perform even better than their optimal model, at least on this sentiment classification task. It is possible that more fine-tuning of the dropout parameter in standard LSTM dropout would lead to performance matching that of the Concrete Dropout LSTM, but as discussed earlier, such grid search over dropout probabilities is computationally expensive and a pragmatic motivation for performing concrete dropout.



(a) Train error

(b) Test error

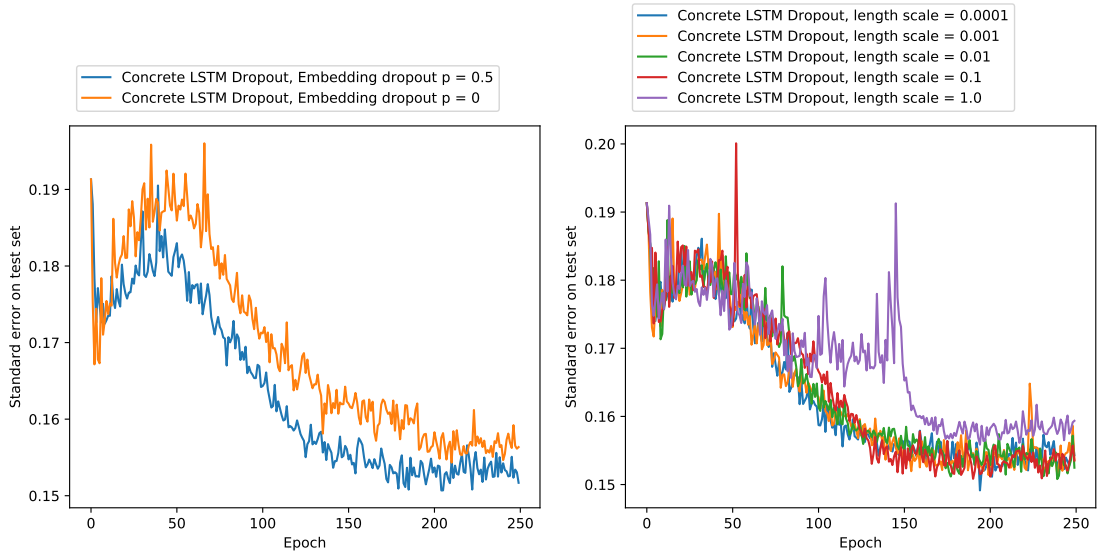
Figure 9: Train and test error for Concrete Dropout and standard dropout LSTMs

Gal and Ghahramani also report on the importance of including dropout at the embedding layer to reduce overfitting, since the embedding layer is often the largest layer in many networks for language applications. We see in Fig. 10a that performing dropout at the embedding layer does cause a non-trivial improvement to the performance of the model on test data. We compared embedding dropout with probability 0.5 and with probability 0 (no embedding dropout), but it would be interesting as a further extension to apply concrete dropout to the embedding layer and allow the dropout probability to be learned over training.

Although the Concrete Dropout LSTM model learns dropout probabilities over training, it does require setting a length scale hyperparameter l as described in the Concrete Dropout feed-forward models. Fig. 10b shows that the performance of our Concrete Dropout LSTM is reasonably robust to the choice of l, with length scales below 1 all performing similarly, and only slight performance degradation seen with $l = 1$.

Interestingly, across all parameter settings tested, the Concrete Dropout LSTM learned dropout parameters $p \approx 0.48$ and $p_{rec} \approx 0.49$. Interpreting the learned p as a measure of uncertainty, it appears that the network remains highly uncertain about its predictions. For this sentiment classification task, this persistent uncertainty may in fact be desirable, since the dataset is quite small - Gal and Ghahramani originally chose it as a test case for preventing overfitting on scarce labelled data [GG16b]. It would thus be interesting to test our Concrete Dropout LSTM on a much larger dataset and evaluate whether the uncertainty represented in p decreases as the model is exposed to more training data.

Overall, our results from applying concrete dropout to LSTM models appear promising. First, the concrete dropout framework can be implemented straightforwardly in RNN architectures without



(a) Test error with and without embedding dropout (b) Test error with different length scales

Figure 10: Effect of hyperparameter settings on Concrete Dropout LSTM

significantly increasing training time. Second, the improved performance we observe suggests that concrete dropout may be practically very useful for complex, multi-layered LSTM and GRU models which can require prohibitively expensive hyperparameter tuning with standard dropout.

5 Conclusion

In replicating the Concrete Dropout paper, we have gained an understanding of interpreting dropout as a variational distribution and the use of the Concrete distribution as a reparametrisation trick. We replicated several main results in the original Concrete Dropout paper, and then investigated a novel application of this technique to recurrent neural networks, also in line with the theory of dropout as a variational distribution. If time allowed, we would further investigate the influence of the remaining hyperparameters – temperature t and prior lengthscale l – which we left at a default value for the majority of the experiments.

6 Acknowledgements

We gratefully acknowledge the use of code from https://github.com/yaringal/ConcreteDropout/blob/master/concrete-dropout.ipynb and https://github.com/yaringal/BayesianRNN which we have adapted for use in this project.

References

[AD82] J. H. Ahrens and U. Dieter. Generating gamma variates by a modified rejection technique. *Commun. ACM*, 25(1):47–54, January 1982.

[Bis09] Christopher Bishop. Pattern recognition and machine learning. In *Textbook*, 2009.

[CJM16] Yee Whye Teh Chris J. Maddison, Andriy Mnih. The concrete distribution: A continuous relaxation of discrete random variables. In *Bayesian Deep Learning workshop, NIPS, 2016*, 2016.

[Gal16] Yarin Gal. Uncertainty in deep learning. In *PhD thesis*, 2016.

[GG16a] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[GG16b] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016.

[GHK17] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3584–3593, 2017.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[PL05] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005.