Modelling Uncertainty in Bayesian Neural Networks with Dropout

Ellis Brown[*], **Melanie Manko**[*], **Ethan Matlin**[*]
Department of Computer Science
Columbia University
New York, NY 10027
{ellis.brown, mm5475, esm2191}@columbia.edu

Abstract

While neural networks are quite successful at making predictions, these predictions are usually point estimates lacking any notion of uncertainty. However, when fed data very different from its training data, it is useful for a neural network to realize that its predictions could very well be wrong and encode that information through uncertainty bands around its point estimate prediction. Bayesian Neural Networks trained with Dropout are a natural way of modeling this uncertainty with theoretical foundations relating them to Variational Inference approximating Gaussian Process posteriors. In this paper, we investigate the effects of weight prior selection and network architecture on uncertainty estimates derived from Dropout Bayesian Neural Networks.

1 Introduction

While neural networks perform astonishingly well in predicting unseen data from the same distribution and are able to fit any arbitrary function (Cybenko (1989), Hornik (1991)), they output only a single point estimate, leaving it unclear how confident the model is in its prediction. Incorporating uncertainty[2] into a neural network's predictions is important for interpreting, understanding, and improving neural networks. First, uncertainty estimates allow a network to distinguish between out-of-distribution and in-distribution data–the model should be less confident about inputs that look very different from the training data than those which are similar. Second, estimates of uncertainty provide useful information to practitioners: seeing a distribution of predicted outcomes is far more informative than a single number.[3] Third, uncertainty estimates have applications in Reinforcement Learning: if a network knows that it has only limited data from certain parts of a distribution, it can spend more time training itself in unseen regions (Blundell et al. (2015)). These limitations have very practical consequences. A self-driving car would be better off notifying the driver that it is uncertain about conditions and needs human input than confidently making a dangerously wrong decision. A central banker setting interest rates should know that her neural network predicting good economic times may actually be quite uncertain about its prediction. Given how neural networks are treated as black boxes, it's even more important to understand model uncertainty.

One promising direction of research in modelling uncertainty with neural networks uses Bayesian techniques. The Bayesian paradigm is a natural one for modelling uncertainty as it treats the world as a series of distributions rather than fixed quantities. Bayesian techniques are used in a variety

[*]Authors listed alphabetically.

[2]This uncertainty includes both epistemic, i.e. the model knows what it doesn't know, and aleatoric, i.e. noisy data, uncertainty

[3]For example, economic surveys often ask people how uncertain they are about the future, not because they want a statistical confidence interval but because this belief uncertainty provides valuable information.

of contexts including in machine learning (Ghahramani (2015); he discusses how to incoprorate probabilistic uncertainty into a variety of models through Bayesian methods however doesn't discuss neural networks), biostatistics (Herzog and Ostwald (2013)), and economics (Herbst and Schorfheide (2015)).

Fortunately neural networks are already somewhat Bayesian already. For example, most networks randomly initialize weights. The key will be to connect Bayesian Neural Networks to more well-understood topics that allow us to get a better sense of what's going on and how to train them feasibly. We will show that a Bayesian Dropout Neural Network is equivalent to variational inference in Gaussian processes. Specifically, section 4.3.1 shows that an infinite-width neural network converges to a Gaussian process (Neal (1995)), section 4.3.2 provides two specific Gaussian process covariance functions corresponding to the Gaussian and Sigmoid activation functions, section 4.1 discusses how variational inference can be shown to be equivalent to the stochastic gradient descent method used to train neural networks (Kingma and Welling (2013) and Gal (2016)), section 4.2 discusses how Dropout, a commonly used regularization technique, is equivalent to variational inference (Srivastava et al. (2014), Hinton et al. (2012), Gal (2016)). Finally, section 4.3 shows how we can obtain a distribution of uncertainty from the neural network by making forward passes on our the input data we'd like to predict from with random network dropout.

We run a number of experiments. We begin with two small examples demonstrating why uncertainty information is useful, especially when a model is predicting on data unlike that which it has already seen (section 2.1). We then illustrate why probability estimates from a softmax output layer are insufficient measures of uncertainty in section 5.1, establishing why the Bayesian Neural Network paradigm is useful. In section 5.2 we examine the uncertainty that various prior distribution choices over weights can introduce to the network before training begins, and how this changes with different choices of activation function. We then explore how the choice of activation function affects our uncertainty bounds (i.e., the posterior distribution of the neural network) in section 5.3, especially as the model predicts on data unlike what it has seen. In section 5.4, we examine how the choice of prior distribution affects the posterior distribution of the network. In section 5.5, we compare the uncertainty obtained from various non-linearities, model sizes, and number of hidden layers. We observe that different non-linearities produce different uncertainty bounds and deeper networks produce more certain predictions, but that varying the model width does not have a significant impact. We then attempt to replicate the theoretical result of a Bayesian Neural Network approximating the Gaussian Process as network width increases, however show that in practice, we better approximate a Gaussian process with depth rather than width. Finally, we present a shortcoming of the Bayesian Neural network approach in section 5.7.

The paper proceeds as follows. Section 2 motivates exactly why uncertainty is necessary in neural networks through a few simple experiments and examples. Section 4 describes the theoretical results that link Bayesian Neural Networks to more well-understood concepts. Section 5 describes the numerical experiments we perform. Finally, section 6 concludes.

2 Problem Description

2.1 Neural Networks Need Better Notions of Uncertainty

To show why uncertainty matters, we begin with a simple experiment shown in figure 1. We would like our neural network to learn $1 - cos(x)$ (shown in green), however the network only sees training data for $x \in [-2, 2]$ (red, between the dotted vertical lines). The network confidently predicts wrong answers for new data outside of the training range. However, putting uncertainty bands around the network's mean prediction (these bands come from the dropout approach which will be explained further through the remainder of the paper) shows that the network is actually quite uncertain about its predictions, especially as it predicts on data farther from where it was trained.

Figure 1: Learning $1 - cos(x)$

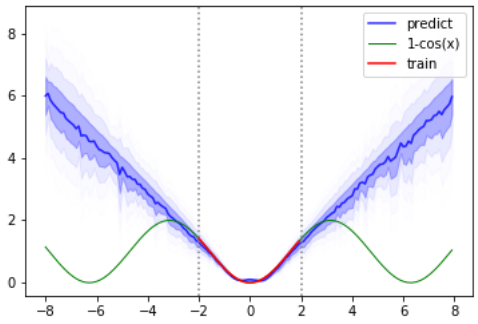


Figure Notes: The green line is the function we're trying to learn ($1 - cos(x)$). The red line is observed data we train on. The dashed vertical lines represent the training sample. The blue line is the model's predictions with shading corresponding to one- and two-standard deviation uncertainty bands. Bayesian Dropout ReLU Network with 5 hidden layers and 1024 neurons per layer. Standard deviations are the standard deviation of 1000 forward passes through the trained network with dropout probability 0.1.

The obvious solution, of course, is to simply train the network on the full range of data (in this example, training on $x \in [-8, 8]$ might yield a prediction more similar to the data-generating process). However, one can always come up with some other function that behaves normally in a large range of input values but then does something crazy as $x \to \pm\infty$.

This is also true in many real world situations: we may not have access to all of the data, datasets are often too small, we may not know the range of possible input values needed to train the network, and there are often unexpected random shocks (or "tail events"): an airplane experiencing some extraordinarily rare pressure zone or weather system, a self-driving car seeing a tumbleweed roll across the road (after having been trained only in urban places where only people, animals, and cars cross the road)[4], or an image classification system fed a picture of some alien species beyond our wildest imaginations. Especially as more critical systems depend on neural networks, this becomes increasingly relevant.

One specific example is in forecasting the recovery from the 2008 recessions. Figure 2 shows predictions from a Dropout ReLU network with 5 hidden layers and 1024 neurons per layer trained on macroeconomic data from 1985 to 2006[5]. The input-layer is 88 dimensional (one layer for each series, including the lagged variables) and the output layer (representing GDP) is 1-dimensional. We treat each year/quarter pair (e.g. 1984 Quarter 1) as a separate 88-dimensional input datapoint. Hence we're predicting GDP at time $t + 1$ as a function of time t variables (for simplicity the plot only shows our predictions as a function of time, however it's important to note that this is not a forecast in 2006 of the entire path of GDP through 2017, but rather individual one-quarter ahead forecasts for $t + 1$ using time t data as input points). The red line represents observed data used to train the model, the first vertical line represents the end of the training period (2006) and the second line represents the financial crisis in 2008.

[4]In fact, issues with self-driving cars have already emerged as demonstrated by the Tesla crash report NHTSA (2017). While estimates of uncertainty won't make self-driving cars not crash, it can be used to alert the driver that she should take over because the car isn't sure what is happening and needs human input.

[5]We use the following series (all of which are publicly available from the Federal Reserve Bank of St. Louis' FRED database (`https://fred.stlouisfed.org`) and their 1 quarter, 2 quarter, 3 quarter, and 4 quarter lags: Real GDP (lags only), Real gross domestic income, Nonfarm Business Sector Unit Labor Cost, Federal Reserve Bank of Philadelphia's Current General Activity Index, Export Price Index, Capacity Utilization, New Private Housing Units, PCE Price Index, PCE excluding Food and Energy, CPI excluding Food and Energy, CPI for Urban Consumers, Import Price Index, Real Disposable Personal Income, Industrial Production Index, Total New Privately Owned Housing Units STarted, Civilian Unemployment Rate, New One Family Houses Sold:, and Total Nonfarm Payrolls. These series comprise the publicly-accessible data used by in the Federal Reserve Bank of New York's Nowcast model used for forecasting GDP using a Vector Autoregressive Model (Bok et al. (2018) (the model's forecasts can be viewed in realtime here: `https://www.newyorkfed.org/research/policy/nowcast`.

Figure 2: Forecasting Gross Domestic Product

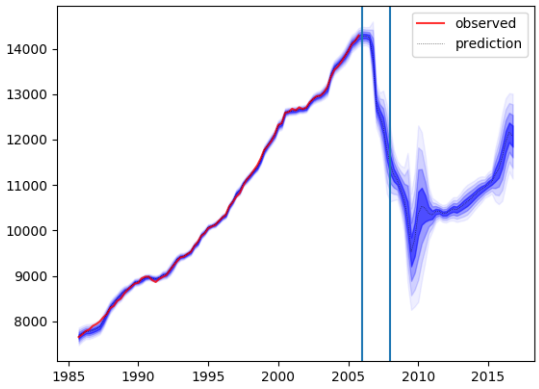


Figure Notes: The red line is is observed data on which the NN is trained. The blue line is the model's predictions with shading corresponding to uncertainty bands. The first vertical line represents the end of the training sample (2006). The second vertical line represents the beginning of the financial crisis in 2008.

While it's interesting that the neural network anticipates the 2008 recession, the main point of the plot is that the model has huge uncertainty during the period surrounding 2010 (and again, but less so in 2016 and 2017[6]). Why is this? The recovery from the 2008 recession, especially around 2010 didn't follow the rules as far as recoveries from recession have usually gone since the Great Depression (Cai et al. (2018) highlights a few reasons why). First, most previous recessions were followed by quick turnarounds while this recession was followed by a prolonged period of lagging GDP. Second, a recession of this magnitude should have (from the perspective of the previous data) caused the Federal Reserve to lower rates more than it actually did; in this case, however, the zero lower bound (inability to set negative interest rates) limited the Federal Reserve's ability to control monetary policy during the recession and recovery (another phenomenon that hadn't been present following previous recessions). Third the Phillips curve, a relationship between output and inflation that has historically held in the data seemed to have weakened or even disappeared. When economic conditions are novel, a neural network trained on past data may yield inaccurate predictions that if taken to heart with no uncertainty may lead policymakers astray.

One way to frame this problem more broadly is in terms of generalization: we want our NN to generalize well to unseen data. To a statistician, this can be thought of as interpolation vs. extrapolation. However, we usually think about it in terms of generalization to new test data *that comes from the same distribution*; in other words, so that we are not overfitting and can learn the true data-generating process instead of noise. One additional idea that we're adding here is of testing on in-distribution vs. out-of-distribution data (i.e. you train the model on data generated from one part of the input distribution but then predict on data from a qualitatively different part of the input distribution: the model should be highly uncertain about its prediction on data far from the data it was trained on).

We can also connect this problem area to the concept of interpretability. Part of interpreting a NN's prediction should involve assessing how uncertain it is about its predictions. Uncertainty estimates can provide a great deal of information about what a NN is actually doing and how confident it is in its predictions that simply looking at a point estimate would ignore. Consider a simple example where you can flip a coin and win either $1000 if heads or lose $999 if tails. The expected value is positive so anyone looking only at the mean, would make the bet. However, the Variance is quite large—it provides a great deal of additional information that would lead most responsible people not to play. For another very basic example of how uncertainty provides useful information, see Krzywinski and

[6]This smaller jump in uncertainty might correspond to what some of called a "mini"-recession in rural areas around 2016–another even that would have been novel from the perspective of previous data. See this NY Times article for more information:`https://www.nytimes.com/2018/09/29/upshot/mini-recession-2016-little-known-big-impact.html`

Altman (2013). Uncertainty estimates can also be useful for interpreting the decisions of complex decision-making systems such as self-driving cars (McAllister et al. (2017)). For example, a simple decision that a self-driving car makes (for example, swerve away from another car) is a function of many other smaller binary decisions (such as "is another car moving too quickly towards me?", "is there another car already where I want to swerve?", "if I swerve, will I hit the other car?", and so on. There is a great visualization of how ignoring the uncertainty inherent to each of these binary decisions can result in a suboptimal action on the second page of McAllister et al. (2017).

We are interested in uncovering the properties of various network architectures and initializations on uncertainty estimates derived from Bayesian Dropout Neural Networks. However, first we provide some background on subjects necessary to understand our results.

3 Background Topics

3.1 Bayesian Neural Networks

The Bayesian approach is a natural technique for modelling uncertainty in that it describes the world as distributions over parameters rather than fixed quantities. In a nutshell, the idea is to formulate beliefs on the distribution of a parameter[7] which we call the *prior*, then determine the *likelihood* of our data given values of the parameters. We use Bayes' rule to combine prior and likelihood, obtaining the *posterior* distribution of our parameters–our beliefs about the value of the parameter after having seen the data:

$$p(\theta|y, \mathcal{M}) = \frac{p(y|\theta, \mathcal{M})p(\theta)}{p(y|\mathcal{M})} \tag{1}$$

One of the first to propose Bayesian neural networks was MacKay (1992) who argued that they could act as a way of learning more about what the network was really doing and how well it might generalize. In the context of a neural network, our parameters are the weights and biases, $\theta = [w; b]$.

Of course, equation (1) is a challenging creature, especially in the context of models as complicated and nonlinear as neural networks. Since analytical expressions to draw from the posterior are unavailable, statisticians have turned to Monte Carlo posterior approximation (which uses an integral to approximate the posterior) and Variational Inference (which uses a derivative to minimize the difference between approximating distribution and posterior).[8] There are a number of Monte Carlo techniques (such as Gibbs Sampling, Sequential Monte Carlo, Hybrid Monte Carlo, Hamiltonian Monte Carlo, Metropolis Hastings)[9]) for sampling from the posterior, many of which are explained in the context of machine learning models by Neal (1993). Different methods have different advantages and disadvantages. For example, Gibbs Sampling requires knowing the conditional distributions of the parameter (which is never really the case in a complicated neural net); Metropolis Hastings struggles with multimodal distributions, takes serially correlated draws, and isn't parallelizable (Herbst and Schorfheide (2014)); Hybrid and Hamiltonian Monte Carlo require assumptions about the the system's dynamics; Etc. Neal (1995) uses a Hamiltonian Monte Carlo method to evaluate a Bayesian Neural Network's posterior which he claims performs relatively well. However, these methods are all quite slow and—in the context of today's large neural networks with millions of parameters—infeasible. Variational inference on the other hand may be far more helpful.

3.2 Variational Inference

When we don't have an analytical expression for the posterior, we can use a well-known, easy-to-evaluate distribution $q(\theta; \lambda)$ to approximate it (where θ is the vector of weight/bias parameters and λ is the parameters of the approximating distribution). By approximate, we mean to minimize the Kullback-Leibler (KL) divergence (Kullback and Leibler (1951), Kullback (1959)) between $q(\theta, \lambda)$

[7] For the sake of clarity and simplicity, we take the model as fixed in this example, however one can also formulate priors and conduct Bayesian inference on classes of models

[8] For additional information Goodfellow, Bengio and Courville (2016) chapters 17 and 19 discuss Monte Carlo sampling techniques as well as techniques similar to VI that approximate distributions via optimization (via derivative) rather than Monte Carlo approximation (via an integral)

[9] There are also Importance Sampling and Rejection sampling however these are obviously far too simple and time-consuming

and $p(\theta|y)$

$$KL(q(\theta,\lambda)||p(\theta|y)) = \int q(\theta,\lambda)\log\frac{q(\theta,\lambda)}{p(\theta|y)}d\theta \tag{2}$$

In other words, we minimize the expected value of the log difference between the approximating distribution and posterior with respect to the approximating distribution. This numerical approximation is called "Variational Inference" which is explained further in Blei, Kucukelbir and McAuliffe (2017).

Of course, minimizing equation [2] is infeasible–it requires computing the posterior (which is the point of all of this in the first place!). Instead, we maximize the Evidence Lower Bound (ELBO).[10] which is equivalent.[11]

$$\text{ELBO} = E_q(\log p(\theta, y)) - E_q(\log q(\theta)) \tag{3}$$

We minimize the sum (across all N datapoints) of the negative ELBOs (equation 3) which we can rewrite (see Appendix A.1 for why we can rewrite equation 3 as $\int q(\theta)\log p(y|\theta)d\theta - KL(q(\theta)||p(\theta))$) as

$$\mathcal{L} = -\sum_{i=1}^{N}\int q(\theta)\log p(y_i|f^\theta(x_i))d\theta + KL(q(\theta)||p(\theta)) \tag{4}$$

Equation 4 is the objective function for variational inference. We will come back to this in section 4.1

A number of papers have used variational inference to estimate neural networks. For example, Barber and Bishop (1998) use variational inference for a neural networks (however, they call it ensemble learning). Bishop et al. (1998) uses a related but alternative approach in which he approximates the neural network with mixtures of approximating distributions rather than just one. Graves (2011) shows a more efficient variational inference approximation of Bayesian neural networks and discuses tradeoffs between various approximating distributions. Finally Jordan et al. (1999) explains variational inference from a graphical model perspective which may be useful for any computer scientists[12]

3.3 Gaussian Processes

A Gaussian Process (GP) is an infinite-dimensional stochastic process such that every finite collection of random variables generated by the process is normally distributed. As with the familiar Gaussian distribution, GPs are exactly determined by their first and second moments. However, since the mean just moves around where the distribution is centered, we usually think of GPs as being exactly determined by their covariance functions.

To get a sense of what different GPs look like, we've plotted the posterior distribution[13] of GPs with a variety of commonly-used covariance function in figure 3. The red line indicates observed data on which the model was trained. The blue line indicates the model's predictions and the blue shading represents one and two standard deviation bands (darker is one standard deviation; lighter is two). Since our input data is only one-dimensional, the blue bands are really just showing the *variance* of the GP posterior corresponding to each covariance function.

[10] We call this the evidence lower bound because it is the lower bound on the log evidence of the model, $p(y)$. Since $\log p(x) = \log\int_\theta p(\theta, y) = \log\int_z p(x,z)\frac{q(\theta)}{q(\theta)} = \log(E_q(\frac{p(\theta,y)}{q(\theta)})) \geq E_q(\log p(\theta, y)) - E_q(\log q(\theta))$ by Jensen's inequality.

[11] The definition of KL divergence is $KL(q(\theta)||p(\theta|y)) = \int q(\theta)\log\frac{q(\theta)}{p(\theta|y)}$. Using log rules and the definition of a conditional probability, we can re-write this as $\int q(\theta)log(q(\theta) - (\int q(\theta)\log p(\theta, y) - \log p(\theta))$ which rearranged is $-(E_q(\log p(\theta, y)) - E_q(\log(\theta)) + \log(p(x))$. This is just $-\text{ELBO} + \log p(x)$. Since we are minimizing KL with respect to the variational distribution q and $p(x)$ does not depend on q, we see that maximizing ELBO is equivalent to minimizing KL. We will generally ignore the constant, $p(y)$, as it has nothing to do with our optimization problem; this is similar to the idea of a distribution's kernel: once we recognize the form of the prior times the likelihood, we immediately know $p(y)$ since the distribution must integrate to 1.

[12] However not economists like me who know nothing about graph theory!

[13] The data comes from the Mauna Loa dataset which is a dataset of C02 emissions over time from the Mauana Loa volcanco. It's the same dataset we will use later on for many of our experiments with NNs and will be descrbed more fully there.

These plots allow us to see some of the properties that different covariance functions imbue onto the GP posterior. Why do we care about these properties? As Williams and Rasmussen (2006) explains, the covariance of a Guassian Process is a kernel (and like a kernel, it measures the similarity between data points. And as for the strong connections between kernels and NNs (Tsuchida, Roosta-Khorasani and Gallagher (2017), Cho and Saul (2009)), there are also strong connections between covariance functions and neural network structure.

For example, the squared exponential, rational quadratic, and Ornstein-Uhlenbeck covariance functions are all stationary (they're also isotropic, however without diving too deeply into stochastic processes, this doesn't matter too much for our purposes). Stationarity means that the GP distribution shifts only for changes in the *distance between* input values, not the values themselves (i.e. its invariant to translations in input space)[14]. We can see this in how the bands are of a relatively constant width for different values of x (they're homoskedastic). The squared exponential covariance function is also infinitely differential which makes it very smooth as can be seen in figure 3.

The polynomial and linear covariance functions (as well as the arcsine covariance function, which isn't pictured but will show up later on in this paper), are examples of non-stationary covariance functions since they vary with translation (i.e. the location of the inputs affects the distribution as well as the distance between them), however the polynomial and linear covariance functions are invariant to rotations since they're *dot product kernels* (kernels for which the covariance only depends on x and x' through a dot product). In figure 3, we see that for these GPs, the variance is quite small (mostly because the σ's are relatively small). We also see that they're unable to fit the data very well since we need a more sinusoidal function (which combinations of squared exponentials can give us but lines and polynomials can't).

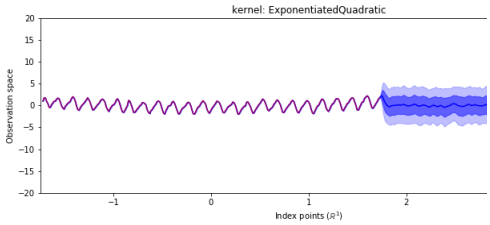
The exponential sine squared covariance function is an example of a periodic covariance function, meaning that the variance induced by it is period (as is easy to see in figure 3). In multiple dimensions, the period would be the same in every dimension.

[14]Usually we think of stationary in the the context of time series statistics however the concept also applies spatially. The dissimilarity between two points x and x' depends on their actual separation for a stationary distribution, however for a non-stationary distribution, their dissimilarity also depends where x and x' are located on the distribution.

Figure 3: Gaussian Processes with Various Covariance Functions

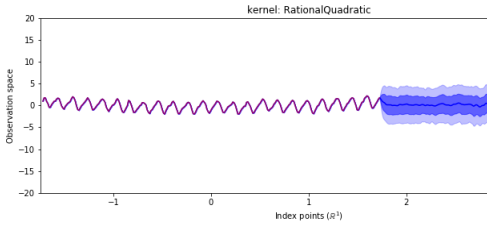
Squared Exponential

$$V(x, x') = e^{\left(\frac{-|x-x'|^2}{2l^2}\right)}$$



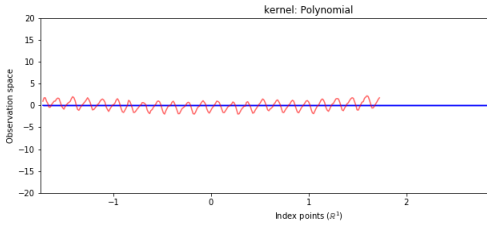
Rational Quadratic

$$V(x, x') = (1 + |x - x'|^2)^{-\alpha}, \alpha \geq 0$$



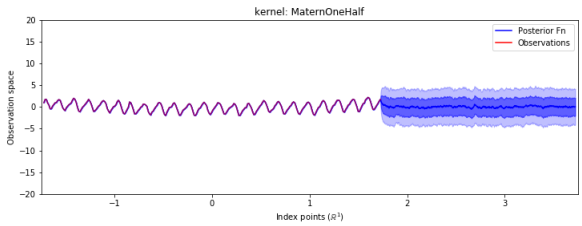
Polynomial

$$V(x, y) = \sigma_b^2 + \sigma_w^2 \left[(x - \text{shift}) \cdot (x' - \text{shift})\right]^d$$



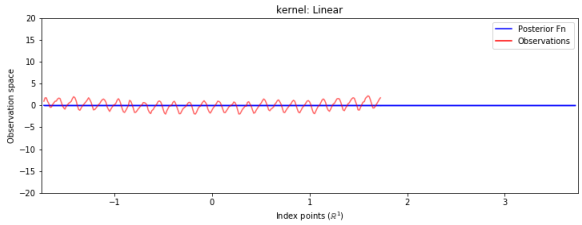
Ornstein-Uhlenbeck

$$V(x, x') = e^{(\frac{-|x-x'|^2}{l})}$$



Linear

$$V(x, x') = \sigma_b^2 + \sigma_w^2 (x - \text{shift}) \cdot (y - \text{shift})$$



Exponential Sine Squared

$$V(x, x') = e^{\left(-\frac{2\sin^2(\frac{x-x'}{2})}{l^2}\right)}$$

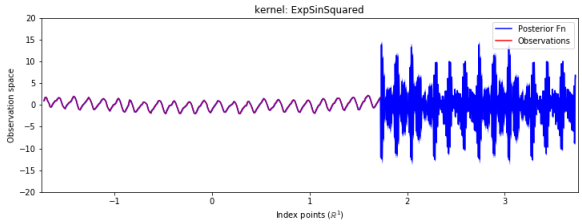


Figure Notes: Trained on the Mauna Loa dataset. Red is observed data. Blue is the model's predictions, shaded blue represents one and two standard deviations around the mean prediction.

4 Previous Theoretical Results

This section proceeds as follows. In section 4.1, we show that variational inference on a NN's posterior is equivalent to the stochastic gradient descent method used to train NNs. In section 4.2 we show that dropout regularization is equivalent to variational inference. In section 4.3, we show that we can also use dropout to generate a distribution of outputs by making taking forward passes through the dropout network. In sections 4.3.1 and **??**, we connect Bayesian NNs and GPs, a well-understood class of functions used in Bayesian statistics. Putting it all together, we get that Dropout on Bayesian Neural Networks gives us a distribution of uncertainty around our point estimate by conducting variational inference in a function space posterior defined by Gaussian processes.

4.1 Variational Inference and Dropout Neural Networks

In section 3.2, we showed that to conduct variational inference, we want to minimize equation 4 which we show again for your convenience:

$$\mathcal{L} = -\sum_{i=1}^{N} \int q(\theta) \log p(y_i|f^{\theta}(x_i))d\theta + KL(q(\theta)||p(\theta))$$

Minimizing this is feasible, however presents two challenges. First, the predictive likelihoods must be computed over the entire dataset, which is costly if the dataset is large. Second, the predictive likelihoods may be costly or infeasible to compute, especially in the context of models as complicated as neural networks.

To resolve the first problem, we can use sub-sampling/mini- batch optimization where we approximate [4] with

$$-\frac{N}{M} \sum_{i \in S} \int q_{\lambda}(\theta) \log p(y_i|f^{\theta}(x_i))d\theta + KL(q_{\lambda}(\theta)||p(\theta)) \tag{5}$$

where S is a random subset with M out of N datapoints

To resolve the second problem, we'll a Monte Carlo approximation of equation [4] called *Stochastic Gradient Variational Bayes (SGVB)* (see Kingma and Welling (2013)).[15] There are other Monte Carlo estimators one could use to approximate equation [4] (a few are explained in Gal (2016)), however this approach yields an optimization procedure that is almost identical to that used in neural networks so it will be easy to explain the connections between Variational Inference and Bayesian Neural Networks using this algorithm in the following section.

To minimize equation [4], we need to compute the derivative of the predictive log likelihood with respect to the parameters of the approximating distribution (λ).

The basic idea is to transform an expected value with respect to $q_{\lambda}(\theta)$ to an expected value with respect to a simpler "parameter-less" distribution.[16] To do so, we re-parametrise the model's parameters as $\theta = g(\epsilon, \lambda)$ where $g(\cdot)$ is differentiable and $\epsilon \sim p(\cdot)$ is an auxiliary variable.[17] We take a Monte Carlo average over L draws from the auxiliary variable's distribution.[18] Written out this is:

$$\begin{aligned} \frac{\partial}{\partial \lambda} \mathbb{E}_{q_{\lambda}(\theta)}[f(\theta)] &= \frac{\partial}{\partial \lambda} \mathbb{E}_{p(\epsilon)}[f(g(\lambda, \epsilon) \\ &\approx \frac{1}{L} \sum_{l=1}^{L} \frac{\partial}{\partial \lambda} f(g(\lambda, \epsilon^l)) \end{aligned}$$

where the first line shows the reparameterization (i.e. we can re-write the expectation with respect to $q_{\lambda}(x)$ to be with respect to $p(\epsilon)$) and the second line shows the Monte Carlo approximation that comes from sampling from $p(\epsilon)$.

[15] Another name for this algorithm used in Gal (2016) is the path-wise derivative estimator. Other names in the literature include the re-parametrisation trick, infinitesimal perturbation analysis, stochastic backpropogation

[16] By parameter-less, we mean something like the standard normal $\mathcal{N}(0, 1)$ $p(\theta)$

[17] One obvious question is how to pick $g(\cdot)$ and $p(\epsilon)$ appropriately so that our problem is actually easier. This is explained further in appendix A.2.

[18] Note that for notational ease, we're writing these derivations as if we were only interested in approximating the posterior distribution of our parameter. While the point of a neural network is obviously to make predictions (and we're therefore interested in the predictive posterior), once we have the posterior, we need only integrate over the input data to compute the predictive posterior.

Then if we start from the sub-sampling objective (equation [5]), we can reparametrise and approximate as follows

$$\begin{aligned}\mathcal{L} &= -\frac{N}{M}\sum_{i\in S}\int q_\lambda(\theta)\log p(y_i|f^\theta(x_i))d\theta + KL(q_\lambda(\theta)||p(\theta)) \\ &= -\frac{N}{M}\sum_{i\in S}\int p(\epsilon)\log p(y_i|f^{g(\lambda,\epsilon)})d\theta + KL(q_\lambda(\theta)||p(\theta)) \\ &\approx -\frac{N}{M}\sum_{i\in S}\frac{1}{L}\sum_{l=1}^{L}\log p(y_i|f^{g(\lambda,\epsilon)}) + KL(q_\lambda(\theta)||p(\theta))\end{aligned}$$

Removing the inner sum,[19] we obtain the minimization objective

$$\hat{\mathcal{L}} = -\frac{N}{M}\sum_{i\in S}\log p(y_i|f^{g(\lambda,\epsilon)}) + KL(q_\lambda(\theta)||p(\theta)) \tag{6}$$

whose derivative we can compute with respect to λ. Algorithm 1 (which is taken from Kingma and Welling (2013)) shows how we minimize $\hat{\mathcal{L}}$ in practice. It's just stochastic gradident descent on λ, the approximating distribution, so it's already easy to see how we will connect this to a neural network! Finally, note that the first term in equation (6) basically amounts to the loss without regularization while the second term is a regularizer that depends on $q_\lambda(\theta)$). We wil reference this structure in connection to Bayesian Neural Networks in the next section.

Algorithm 1 Variational Inference of Posterior using Pathwise Estimator as per Kingma and Welling (2013)

Input: data x_i, y_i, parameters θ, λ
repeat
  Randomly sample M datapoints from the N-sized dataset
  Randomly sample $\epsilon \sim p(\epsilon)$ (M times)
  Compute the gradient of our estimator

$$\nabla_\lambda\mathcal{L} = -\frac{N}{M}\sum_{i\in S}\frac{\partial}{\partial\lambda}\log p(y_i|f^{g(\lambda,\epsilon_i)}(x_i)) + \frac{\partial}{\partial\lambda}KL(q_\lambda(\theta)||p(\theta))$$

  Update λ with stochastic gradient descent $\lambda = \lambda + \eta\nabla\mathcal{L}$
until λ converged

Variational Inference has been used to estimate the parameters of a neural network through a variety of approaches including including Barber and Bishop (1998),Graves (2011), and Blundell et al. (2015). While it's more efficient than Markov Chain Monte Carlo, we want a faster technique.

4.2 Dropout Regularization in Neural Networks

Fortunately, there are more recent papers (Gal and Ghahramani (2015*a*), Gal and Ghahramani (2016*a*), Gal and Ghahramani (2016*b*) Gal and Ghahramani (2015*b*)) showing that Dropout Regularization in Bayesian Neural Networks is equivalent to Variational Inference. This is highly convenient since dropout is already widely used as a stochastic regularization technique (SRT). Hence, computing uncertainty bands via dropout requires only minimial additional computational time.

Dropout was first introduced as a stochastic regularization technique by Hinton et al. (2012) and Srivastava et al. (2014). The idea is to randomly "switch on" or "off" different neurons with probability p during training and testing.

[19] We remove the inner sum and $\frac{1}{L}$ since mini-batching already takes care of the Monte Carlo part (instead of averaging over L draws to compute the expected value, we're averaging over the M datapoints subsampled out of N; in fact in Kingma and Welling (2013), they set $L = 1$ in their experiments since they're already getting stochasticity from the sub-sampling).

For example, say we have a single hidden layer network with Q inputs and K hidden layer neurons. We sample $\{\epsilon_{1i}\}_{i=1}^{Q} \sim Bern(p_1)$ and $\{\epsilon_{2i}\}_{i=1}^{K} \sim Bern(p_2)$. This gives us output[20]

$$\hat{y} = \sigma(X\text{diag}(\boldsymbol{\epsilon_1})W_1 + b)\text{diag}(\boldsymbol{\epsilon_2})W_2$$

Letting $\tilde{W}_i = \text{diag}(\epsilon_i)W_i$ be our randomly dropped-out weights, we can rewrite this as

$$f^{\tilde{W}_1,\tilde{W}_2,b}(x) = \hat{y} = \sigma(x\tilde{W}_1 + b)\tilde{W}_2 \tag{7}$$

which looks like how we'd normally write a network (but with random weights denoted by the tilde's).

The objective function for a dropout network might look something like

$$\mathcal{L} = \frac{1}{M}\sum_{i\in S} E(f(x_i), y_i) + \lambda_1||\tilde{W}_1|| + \lambda_2||\tilde{W}_2|| + \lambda_3||b|| \tag{8}$$

which takes the form of an error function denoted by $E(\cdot)$ (perhaps quadratic loss) plus regularization terms (in $||\cdot||$).

For many cases (in particular, the Gaussian case for which we'll show in the next section that a Bayesian Neural Network approximates as $N_{\text{neurons}} \to \infty$), minimizing the quadratic loss is equivalent (up to a constant) to maximizing the likelihood (one way to think about this for a statistician is that Maximum Likelihood estimators for Normally-distributed random variables minimize quadratic loss).

So we can redefine our minimization objective as

$$-\log p(y|f^{\theta}(x)) + \text{const.} \tag{9}$$

and the regularized loss function as

$$\mathcal{L} = -\log p(y|f^{g(\theta,\epsilon_i)}) + \text{const.} + \lambda_1||\tilde{W}_1|| + \lambda_2||\tilde{W}_2|| + \lambda_3||b|| \tag{10}$$

which looks remarkably similar to equation [6]. Both equation [6]'s and [10]'s first term is something we want to maximize, and the following terms are regularizers. For equation 6, the regularization term depends on $q_\lambda(x)$, the approximating distribution, while for equation 10, the regularization terms depend on the random weights. Hence, different stochastic regularization techniques/different priors on weights (anything that controls the regularization terms in equation [10]) should correspond to different approximating distributions (which control the regularization term in equation [6]).

Specifically, the KL condition

$$\frac{\partial}{\partial\theta}KL(Q_\lambda(\theta)||p(\theta)) = \frac{\partial}{\partial\theta}\lambda_1||W_1|| + \lambda_2||W_2 + \lambda_3||b|| \tag{11}$$

shows when the regularization terms in equations [6] and [10] are the same. In that case, $\frac{\partial}{\partial\theta}L_{dropout}(\theta) = \frac{\partial}{\partial\theta}L_{MC}(\theta)$ and the optimal weights found using optimization of a dropout neural network are the same as those found using Variational Inference on a Bayesian NN of the same structure.

4.3 Generating the Distribution of a Neural Network's Output

In the previous section we showed that estimating a neural network with variational inference is equivalent to training a neural network using stochastic gradient descent. Next, we want to use this network to obtain estimates of predictive uncertainty.

To do so, we just need to integrate over the data which we do via Monte Carlo simulation. To get the mean of the predictive distribution, we make T forward passes throgh the network and compute:

$$\mathbb{E}(f_{NN}(x)) \approx \sum_{t=1}^{T} f_N N(x)$$

Of course, the mean is just a point estimate, something with which existing neural networks already do quite well.

[20] $\text{diag}(\mathbf{v})$ is an operator that constructs a diagonal Matrix with entries given by $\mathbf{v}$

We're interested in the second moment of the predictive distribution. Gal (2016) shows that we can approximate this by taking the Monte Carlo average over T forward passes through the network as $T \to \infty$

$$\sigma^2 \mathbf{I} + \frac{1}{T} \sum_{t=1}^{T} f^{\theta_t}(x)' f^{\theta_t}(x) \xrightarrow{T \to \infty} E_{q_\lambda(y|x)}[y'y] \tag{12}$$

where σ^2 represents the prior variance of the weights (we treat it as a scalar here and therefore multiply by the identity, however we could also have different σs for different weights—this would just make the math a bit more complicated).

We start out with the definition of the 2nd moment of the distribution approximated via variational inference (this is why the expected value is with respect to q_λ, the approximating distribution).

$$\begin{aligned}
\mathbb{E}_{q_\lambda(\mathbf{y}|\mathbf{x})}[\mathbf{y}^T \mathbf{y}] &= \int \left(\int \mathbf{y}^T \mathbf{y} p(\mathbf{y}|\mathbf{x}, \theta) d\mathbf{y} \right) q_\lambda(\theta) d\theta \\
&= \int \left(Cov_{p(\mathbf{y}|\mathbf{x},\theta)}[\mathbf{y}] + \mathbb{E}_{p(\mathbf{y}|\mathbf{x},\theta)}[\mathbf{y}]' \, \mathbb{E}_{p(\mathbf{y}|\mathbf{x},\theta)}[\mathbf{y}] \right) q_\lambda(\theta) d\theta \\
&= \sigma^2 \mathbf{I} + f^\theta(\mathbf{x})' f^\theta(\mathbf{x})) q_\lambda(\theta) d\theta
\end{aligned}$$

In the first equality, we use the integral definition of expected value (outer integral) and integral definition of the second moment (inner integral). The second line uses that $E(X^2) = Var(X) + [E(X)]^2$ (in vector form). The third line is true because the the prior variance is given by σ^2 and the mean of the network is just the network's prediction.

By definition of variance, we obtain the following estimator:

$$V(y^*) := \sigma^2 \mathbf{I} + \frac{1}{T} \sum_{t=1}^{T} f^\theta(x^*)' f^\theta(x^*) - \mathbb{E}(y^*)' \, \mathbb{E}(y^*) \xrightarrow{T \to \infty} Var_{q_\theta^*(y^*|x^*)}(y^*) \tag{13}$$

We can also compute other moments and quantiles of the predictive distribution via Monte Carlo approximation taking $T \to \infty$ forward passes through the network.

4.3.1 Neural Nets and Gaussian Processes

One important result central to Bayesian Neural Networks, and this paper in particular, comes from Neal (1995) who shows that with an infinite number of neurons (in a single hidden layer network), the prior over functions[21] induced by the Bayesian Neural Network (and therefore posterior over functions) converges to a Gaussian Process. Neal uses the following single hidden layer setup with H neurons in the hidden layer and I inputs:

$$f_k(x) = b_k + \sum_{j=1}^{H} \nu_{jk} h_j(x) \tag{14}$$

and

$$h_j(x) = \sigma(a_j + \sum_{i=1}^{I} u_{ij} x_i) \tag{15}$$

The nonlinearity $\sigma(\cdot)$ can be any bounded function. The input-to-hidden weights and hidden unit biases u_{ij} and a_j can have any distribution so long as they are i.i.d and the hidden-to-output weights ν_{jk} can have any zero-mean and finite-variance distribution.

Neal (1995) shows that the prior over output functions $f_k(x)$ induced by our prior over weights and biases is Gaussian. The basic idea is to show that each $v_{jk} h_j(x)$ in the sum is normally distributed with mean zero and bounded variance then use the Central Limit Theorem.

[21] We say "prior over functions" and "posterior over functions" because the output of a Bayesian Neural Network is a function and a Gaussian Process is a distribution over functions. However, we distinguish these from the "prior over weights" which is simply the distribution used to initialize weights. This dinstinction is important because the mapping from priors over weights to priors over functions induced by a NN are highly nonlinear (since the network is nonlinear) as we will explore in section **??**

First, the mean of each summand is $\mathbb{E}[\nu_{jk}h_j(x)] = \mathbb{E}[\nu_{jk}]\,\mathbb{E}[h_j(x)] = 0$. The first equality holds because ν_{jk} and $h_j(x)$ are independent and the second one holds because $\mathbb{E}[\nu_{jk}] = 0$ by assumption and $\mathbb{E}[h_j(x)] = \mathbb{E}[\sigma(a_j + \sum_{i=1}^{I} u_{ij}x_i)] = 0$ since the weights and biases are all mean zero.[22]

The CLT requires that the variance is bounded. This holds so long as the nonlinearity is bounded and the variances of the priors are bounded since $\mathbb{E}[(\nu_{jk}h_j(x))^2] = \mathbb{E}[\nu_{jk}^2]\,\mathbb{E}[h_j(x)^2] = \sigma_\nu^2\,\mathbb{E}[h_j(x)^2] = \sigma_\nu^2 V(x)$ if we call $V(x) = \mathbb{E}[h_j(x)^2]$) (as before, we can switch the expected values and the product since everything is i.i.d). The CLT tells us that as $H \to \infty$, the sum $\sum_{j=1}^{H} \nu_{jk}h_j(x)$ becomes distributed according to $\mathcal{N}(0, H\sigma_\nu^2 V(x))$. Therefore, $f_k(x) \sim \mathcal{N}(0, \sigma_b^2 + H\sigma_\nu^2 V(x))$ where σ_ν^2 and σ_b^2 are the variances of the weights and biases.

Hence, we've characterized the mean and variance of the Gaussian Process induced by the infinite-width NN with random weight initializations satisfying the i.i.d assumptions above. Note that since the network is infinite, the sums in equations 14 and 15 become infinite sums and $V(x) \to \infty$ (since $H \to \infty$), however Gaussian Processes are infinite objects such that any finite collection of them is Normally-distributed which equations 14 and 15 satisfy as shown above.[23]

4.3.2 Specific Covariance Functions

One shortcoming of Neal (1995) is that he only shows that there exists some Gaussian Process corresponding to an infinite-width single layer network, however he doesn't show which network architectures correspond to which Gaussian Processes (in other words, he doesn't show what $V(x)$ is). Williams (1998) extends Neal (1995)'s results by providing analytic expressions for the covariance functions defining GPs that correspond to NNs with sigmoid and Gaussian nonlinearities.

First, he shows that for a single layer neural network with a sigmoidal nonlinearity defined by the error function, as the number of hidden layers $\to \infty$, the network converges to a $\mathcal{N}\left(0, V_{\text{erf}}(x, x')\right)$ Gaussian Process where $V(x, x') = \mathbb{E}_u[\text{erf}(x; u)\,\text{erf}(x'; u)]$ is given by

$$V_{\text{erf}}(x, x') = \frac{2}{\pi} sin^{-1} \frac{2x^T \Sigma x'}{\sqrt{(1 + 2x^T \Sigma x)(1 + 2x'^T \Sigma x')}} \tag{16}$$

where ($u \sim \mathcal{N}(0, \Sigma)$ are the weights).

For reference, the error function (it looks like $\tanh(x)$; see figure 4) is:

$$\text{erf}\, x = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} \tag{17}$$

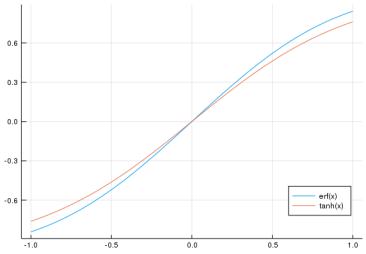


Figure 4: $\text{erf}\, x$ and $tanh(x)$

The mean is zero for the same reasons as in Neal (1995); the hard part is the covariance function.

The sketch of the proof is as follows. The appendix contains a more detailed proof which uses the outline from Williams (1998) (however we add additional explanation and exposition to the barebones outline that Williams (1998) provides).

[22] Note that this argument relies on the data being fixed rather than random which is reasonable since the data is fixed once we observe it. Alternatively, the proof also works if for random data we assume the data are independent from the weights u_{ij}.

[23] Often, we also scale the prior variance of the weights by the number of hidden units by setting $\sigma_\nu = \omega_\nu H^{-1/2}$ for some ω_ν then the prior for f_k converges to Gaussian with variance $\sigma_b^2 + \omega_\nu^2 V(x)$.

By definition of $V(\mathbf{x}, \mathbf{x}')$[24]

$$V_{\text{erf}}(\mathbf{x}, \mathbf{x}') = \frac{1}{(2\pi)^{\frac{d+1}{2}}|\Sigma|^{1/2}} \int \text{erf}(\mathbf{u}^T\mathbf{x}) \, \text{erf}(u^T\mathbf{x}') \cdot \exp(-\frac{1}{2}\mathbf{u}^T\Sigma^{-1}\mathbf{u}) d\mathbf{u} \tag{18}$$

where $u \sim \mathcal{N}(0, \Sigma)$ are the weights.

We simplify our integration problem by (1) using a few clever substitutions to rewrite equation [18] in a more convenient form without Σ and (2) change bases to convert from a $d+1$-dimensional to a 2-dimensional integral, ending up with a function of the form (this is quite similar to the first step of Cho and Saul (2009) when they're deriving the kernel that corresponds to certain neural networks)

$$\frac{1}{2\pi} \int \int \text{erf}(a_1^T\mathbf{u}') \, \text{erf}(a_2^T\mathbf{u}') \dot{e}xp\left(-\frac{u_1'^2 + u_2'^2}{2}\right) du_1' du_2' \tag{19}$$

We will evaluate this integral by constructing a "dummy function" $I(\lambda)$ equal to the integrand for $\lambda = 1$ and then "integrate by differentiating with respect to the parameter" λ (which is just an auxilliary variable).[25]

After integrating by parts and some very clever substitutions, we end up with $I'(\lambda) = \frac{2}{\pi}\frac{1}{\sqrt{1-\theta}}\frac{d\theta}{d\lambda}$ (θ is defined as $\theta = \frac{2x^T\Sigma x'}{\sqrt{(1+2x^T\Sigma x)(1+2x'^T\Sigma x')}}$. This should be easily recognizable as $\frac{d}{d\theta}\arcsin(\theta)$ (using the chain rule since θ is a function of λ. So $I = \frac{2}{\pi}\sin^{-1}(\theta)$, or since $\theta = \frac{2x^T\Sigma x'}{\sqrt{(1+2x^T\Sigma x)(1+2x'^T\Sigma x')}}$,

$$V_{\text{erf}}(\mathbf{x}, \mathbf{x}') = \frac{2}{\pi} sin^{-1}\left(\frac{2x^T\Sigma x'}{\sqrt{(1+2x^T\Sigma x)(1+2x'^T\Sigma x')}}\right) \tag{20}$$

Second, Williams (1998) shows that for a Gaussian nonlinearity, the covariance function is:

$$V_G(x, x') = \left(\frac{\sigma_e}{\sigma_u}\right)^d \exp\left(-\frac{\mathbf{x}^T\mathbf{x}}{2\sigma_m^2}\right) exp\left(-\frac{(\mathbf{x}-\mathbf{x}')^T(\mathbf{x}-\mathbf{x}')}{2\sigma_s^2)}\right) \exp\left(-\frac{\mathbf{x}'^T\mathbf{x}'}{2\sigma_m^2}\right) \tag{21}$$

where the σ terms are functions of the the variance of the priors. This proof is more complicated and relies on a long, tedious integration so we refer readers to the Williams (1998) paper.

This covariance function is a scaled squared exponential covariance function. If it were missing the final exponential term which, it would be just a squared exponential covariance function which would be a nice result since the squared exponential covariance function is stationary. However, since we're scaling by $e^{\frac{u^Tu}{2\sigma_m^2)}}$ here, we actually end up with a nonstationary covariance function.

5 Experiments

In addition to the two experiments presented section in 2 used to motivate our paper, we run a number of experiments exploring how model structure affects estimates of uncertainty obtained from a Bayesian Dropout Neural Network.

First, we demonstrate that the probabilities put out by a softmax layer do not capture uncertainty in a useful manner. Second, we explore a number of different weight initalizations and how these priors on weights map to priors over functions (i.e. priors on the neural network). Third, we explore different activation functions (which correspond to different covariance functions in GPs). Fourth, we investigate the effect of network width and depth on uncertainty estimates. Finally, we'll present one shortcoming of this approach for modeling uncertainty.

[24]The Gaussian distribution on weights gives us the exponential term and the denominator in front while the product of error functions comes from multiplying the nonlinearities as one does when computing a variance.

[25]$I(\lambda) = \frac{1}{2\pi}\int \text{erf}(\lambda a_1^T u') \, \text{erf}(a_2^T u) e^{-\frac{u_1'^2+u'^2_2}{2}}$

5.1 Probabilities Obtained via Softmax Output Layer are Insufficient

For classification tasks, one may argue that a neural network can easily model uncertainty by simply outputting a probability obtained via a softmax output layer.

We illustrate why this isn't the case using the following example.[26] Let $f(x) = \begin{bmatrix} sin(x) \\ cos(x) \end{bmatrix}$ be the output of some model (this can be a neural network or any other model). Now add noise to this output (the noise can either represent noise in the data or uncertainty about the proper functional form for the model). Let $\tilde{f}(x) = f(x) + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \end{bmatrix}$ where $\varepsilon_i \sim N(0, 1)$. In figure 5 panel A, we plot $f(x)$ and in panel B we plot the output of the softmax layer $\sigma(f(x))$ (for simplicity, we only show the first output $sin(x)$; solid line). In panel C, we add noise to each function to obtain a distribution (represented by the 95% bands) $\tilde{f}(x)$. We then recompute the softmax for the whole distribution of inputs which is shown in panel D. Although $\sigma(f(x))$ (the point estimate) gives almost 100% probability of belonging to the red category, if we look at the uncertainty bands, we see that there is actually quite a bit of uncertainty about this prediction (anywhere from 70 to 100% probability of belonging to the "red").

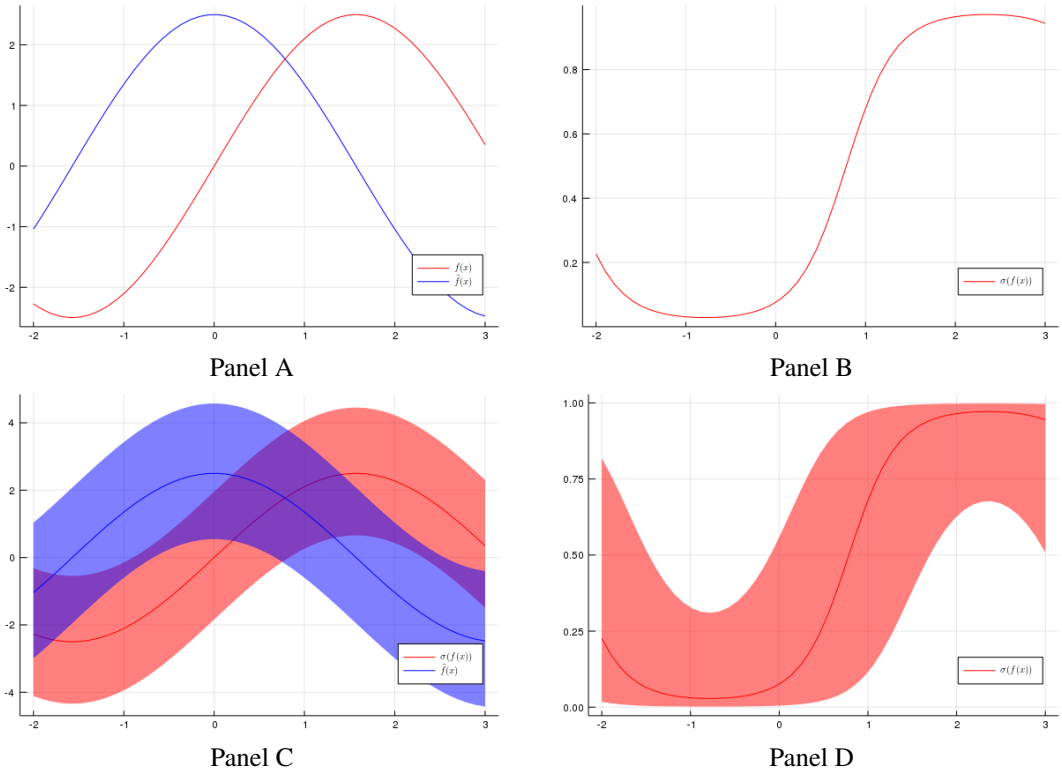


Figure 5: Panel A shows $f(x)$. Panel B shows $\sigma(f_1(x))$. Panel C shows $\tilde{f}(x)$. Panel D shows $\sigma(\tilde{f}_1(x))$

The point of this toy example is that while the probability of a category represents uncertainty to some extent, it misses how uncertain we are about that uncertainty/probability. A probability of some outcome is a parameter with a distribution just like any other parameter.

Where does this uncertainty come from and what does it represent? We might be uncertain about our model, the data used to train and test the model might be noisy, or the data we are using our model to predict with are far from the distribution of data the model was trained on. Softmax doesn't help us with out-of-distribution data—since it must assign probability to some outcome category, it will simply pick the category that is most likely relative to the others, even if that category is a bad fit. It also doesn't include uncertainty about the model

[26] A similar set of plots are shown in Gal and Ghahramani (2016*a*), however they present only a sketch of some imagined function without specifying the function. In this plot we give an example of an actual function for which this occurs

since it just takes a point estimate from the model as a given and evaluates that outcome relative to other possible outcomes.

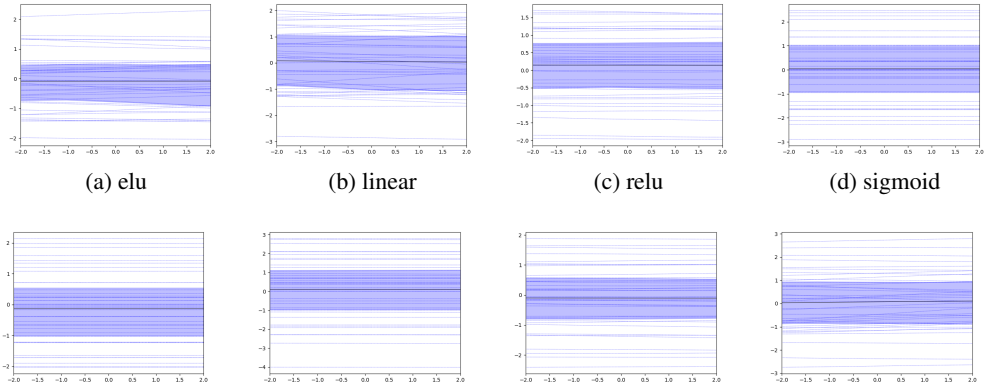
5.2 Prior Plots

Before looking at uncertainty estimates from the full posterior/trained NN, it's important to get a sense of the uncertainty each prior on weights induces onto the network for different activation functions.[27] Figures 6-9 show the priors on weights induced by combinations of different weight priors and activation functions. They help us to understand the uncertainty that our priors on weights put on the network before training on data. To generate each plot, we take 40 forward passes through the network (5 hidden layers, 1024 neurons per layer) for each input x then compute the 25th quantile, 75th quantile, and mean of the network output distribution. The bold black line is the mean, the blue shaded region is the interquartile range, and the light blue lines are the output from individual forward passes through the network.

Since, we don't know before seeing the data where in the data-space we want our model to be more and less uncertain, we would like the variance of our distribution to be relatively constant across values of x.[28]

Figures 6 and 7, which use the Glorot/Xavier Normalization (Glorot and Bengio (2010)) seem to satisfy this criterion the best. The basic idea of this procedure is to normalize the variance of our prior on weights to take into account the number of inputs and outputs into each layer. While Glorot and Bengio (2010) originally invented this technique to improve training by avoiding saturated activations, our new result shows that this initialization has a second advantage: it doesn't presuppose any uncertainty for different inputs before seeing the data. Naively drawing from a Normal or Uniform distribution, on the other hand however, seems to lead heteroskedastic (i.e. varying variance) prior uncertainty estimates (this is especially interesting because a standard deviation of $\sigma = 0.05$ is quite small). While certain activation functions like sigmoid, softmax, and softplus seem to suffer from this less, in general, it seems that a prior on weights that doesn't take into account the structure of the layers (i.e. the width of the previous and next layer) may be a poor choice of prior in terms of learning uncertainty. In appendix A.4, we show a number of other prior initializations.[29] In general, sigmoid, sotmax, and usually softplus seem to produce constant-uncertinaty NN-priors no matter what initialization prior is used.

Figure 6: $W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{N_{in}+N_{out}}}\right)$ (Glorot and Bengio (2010)))



(a) elu (b) linear (c) relu (d) sigmoid

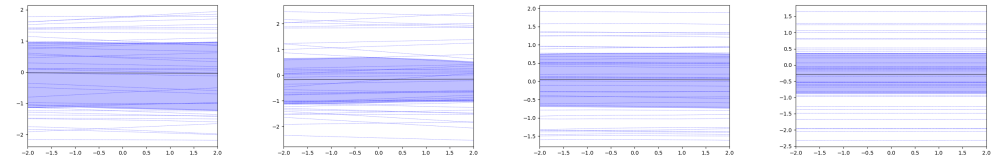
(e) softmax (f) softplus (g) softsign (h) tanh

[27] While the results of Williams (1998) (showing that certain activation functions correspond to different GP covariance functions which have known properties like stationarity, etc.) give us a sense of what to expect for certain activation functions (at least as network width $\to \infty$), it's not immediately obvious what each prior will look like with various activation functions since NNs are highly nonlinear.

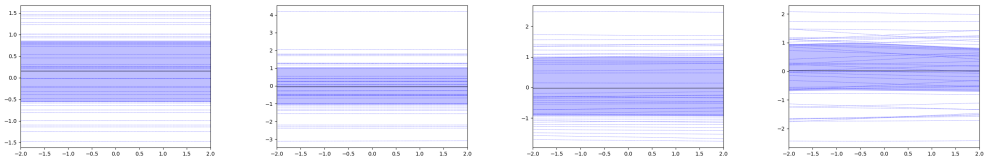
[28] For a stationary covariance function, this will always be the case.

[29] One is a method introduced by He et al. (2015) that they show works better for training networks using nonlinear rectifiers such as ReLU (however, as figures A.1 and A.2 show, they lead to heteroskedastic prior uncertainty). Another is an approach used in Klambauer et al. (2017) that they argue works well and provides nice training properties with Scaled Exponential Linear Units (which are quite similar to our ELU units). For our purposes, however, the Klambauer et al. (2017) normalization only provides homoskedastic prior uncertainty for the softmax, softplus, and sigmoid (notably, it doesn't for the eLU units). We also show results for a truncated Normal distribution (truncated at $> 2\sigma$) that theoretically should that should reduce the probability of drawing "outlier" weights from the prior, however this also produces heteroskedastic uncerainty (only for sigmoid, softmax, and softplus does it perform better).

Figure 7: $W \sim \text{Unif}\left[-0.05, 0.05\right]$ (Glorot and Bengio (2010))

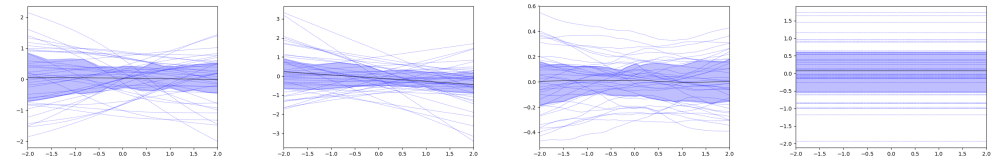


(a) elu (b) linear (c) relu (d) sigmoid

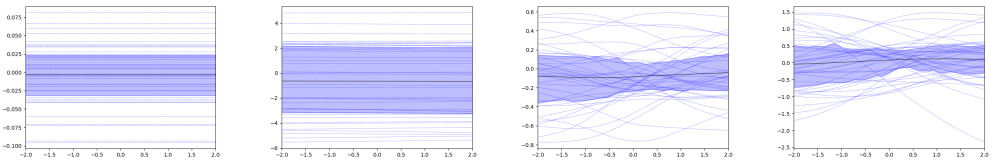


(e) softmax (f) softplus (g) softsign (h) tanh

Figure 8: $W \sim \mathcal{N}\left(0, 0.05\right)$

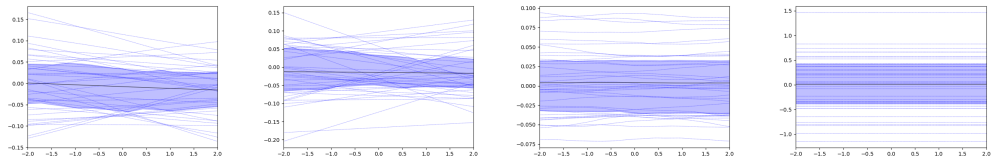


(a) elu (b) linear (c) relu (d) sigmoid

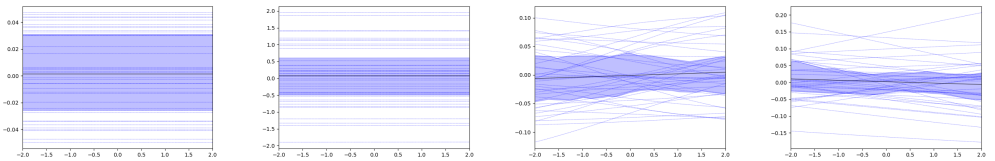


(e) softmax (f) softplus (g) softsign (h) tanh

Figure 9: $W \sim \text{Unif}\left[-0.05, 0.05\right]$



(a) elu (b) linear (c) relu (d) sigmoid



(e) softmax (f) softplus (g) softsign (h) tanh

5.3 Activation Fuctions and Uncertainty Estimates

We next explore how the choice of activation function affects our uncertainty bounds. This exercise is akin to examining the posterior distribution of the Neural Networks output function. As Williams (1998) showed, different nonlinearites in a single hidden layer network correspond to different Gaussian Process covariance functions (as network width $\rightarrow \infty$). For reference, appendix figure **??** illustrates what each activation function we use looks like.

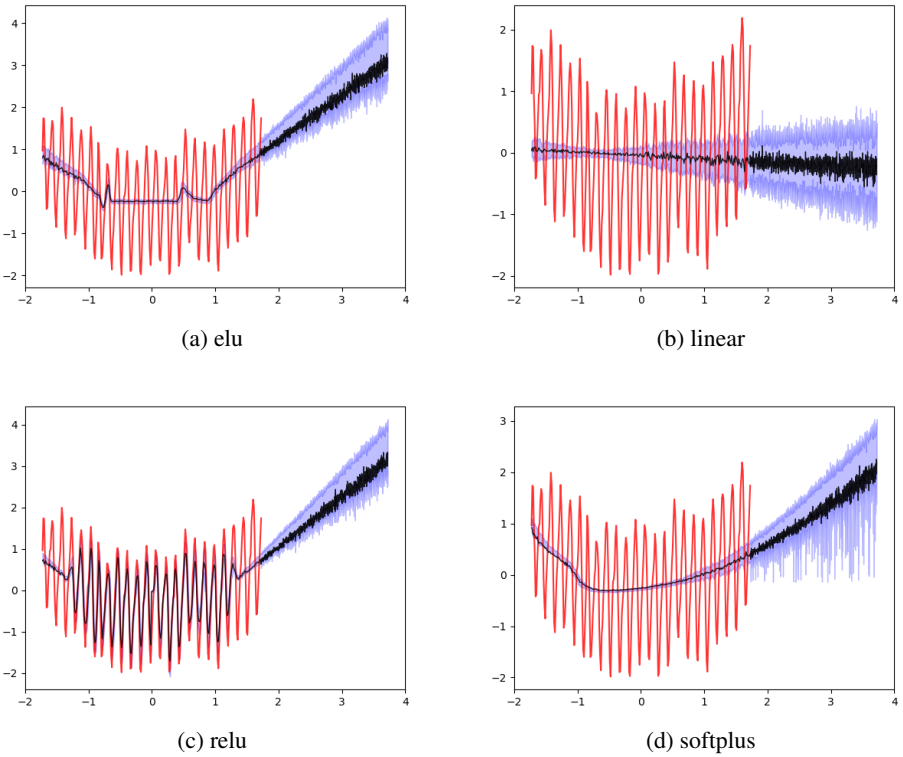
We evaluate the model's uncertainty predictions during extrapolation of the Mauna Loa CO2 concentrations dataset (Keeling (2004)). This dataset is a useful one for this purpose as it is relatively volatile and oscillates with high frequency. Hence it provides both a difficult function to learn that a NN is a prime candidate for (despite that it only has one input dimension) and also allows a natural conception of uncertainty that looking only at a point estimate wouldn't capture.

We train a neural network (5 hidden layers and 1024 hidden units) using various activation functions and dropout probability 0.1[30] on the data shown in red then test on the entire dataset. Model uncertainty estimates are created by sampling 100 forward passes from the network with different random hidden units dropped out each pass. We use this sample to calculate the mean (which approximates the point estimate output by an ordinary neural net i.e. what the output of our NN would be with dropout probability 0.0) and the model's uncertainty prediction. For the sake of this experiment, we have represented the uncertainty estimate via the middle 50% bands of the distribution in order to see the skew of the distribution as opposed to using the standard deviation which would always yield symmetric uncertainty around the mean. In some experiments (especially for the Sigmoid, Softmax, TanH, and Softsign activations), the prior-activation-hyperparameter combination caused the gradients to collapse.[31] In others, the network was not trained long enough to converge. As our primary concern is the behavior of the uncertainty estimates as the model observes data far from the data it has already seen, we leave these cases for further investigation with more computational resources. However, we show our current results in appendix figure A.7.

[30] We train for more than 10,000 epochs in all cases, however the exact number depends on the nonlinearity and how difficult the network was to train.

[31] Part of the reason some of these many not have trained properly is because of the choices of prior. For example, Glorot and Bengio (2010) shows that the sigmoid activation performs poorly with random weight initialization because the mean of random draws put through a sigmoid nonlinearity is nonzero, leading deeper neurons to saturate.

Figure 10: Different Activation Functions and Uncertainty Estimates



(a) elu

(b) linear

(c) relu

(d) softplus

Figure Notes: Comparison of various activation function selections on the uncertainty predictions. We also ran experiments using the sigmoid, softmax, and exponential activation functions, but have omitted them in this example as their gradients collapsed during training.

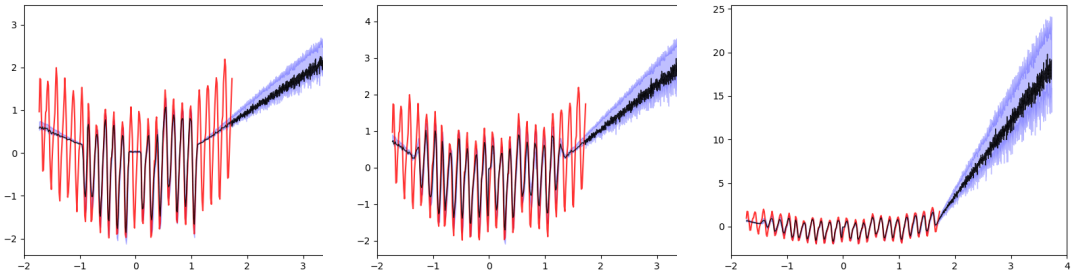
Figure 10 shows that uncertainty bounds can depend on the choice of activation function. In each plot, the black line represents the mean of the posterior (which is what we'd get as the point estimate from an ordinary NN) alone usually doesn't capture the high frequency oscillations. However with uncertainty bands, we at least capture the possibility that the NN's point estimate prediction may not be entirely accurate.

The linear (figure 10b), ReLU (figure 10c), and eLU (figure 10a) activation functions yield uncertainty bounds that fan out linearly. For the linear, ReLU, and eLU nonlinearities, this makes intuitive sense as linear is linear and ReLU and eLU contain linear components. Softplus yields uncertainty bounds that seem to fan out slightly more nonlinearly. In all cases, the uncertainty bands fan out as they move farther from the training data which is exactly what we'd want and expect.

5.4 Choice of Prior and its Effect on the Posterior

We've already shown in section 5.2 that the choice of prior on weights says something about the prior over functions. One example of how this manifests into the posterior estimates is shown in figure **??**. While the network trains makes reasonable predictions with both Glorot Normal and Glorot Uniform initialization, the mean and uncertainty bands both spike outside of the training range when initializing weights with a $\mathcal{N}(0, 0.05)$ prior. This experiment acts as a cautionary tale when selecting priors: it's important to be mindful of the prior otherwise, one may obtain strange results (in particular, one does not want to presuppose any kind of differing uncertainty for different types of input data unless there is a good justification for doing so).

(a) Glorot Uniform (b) Glorot Normal (c) $\mathcal{N}(0, .05)$



Finally, we conduct the same experiment for all cross products of [Glorot Normal, Glorot Uniform, and $\mathcal{N}(0, 0.05)$ prior weight initialization] and [Linear, eLU, ReLU, TanH, Softplus, and Softsign activations]. While the it's already a well-established result (see He et al. (2015), Klambauer et al. (2017), for example) that the choice of prior weight initialization matters a lot for training performance, A.7 provides additional evidence of this evidence for this. For example, the linear activation only trains with a Normal prior that uses Glorot variance normalization (top middle). A TanH network seems to train slightly better with a $\mathcal{N}(0, 0.05)$ prior (bottom right). Softplus only trains with a Normal prior that Glorot-normalizes the variance (bottom middle).

5.5 Depth and Width

We next evaluate how depth and width affect uncertainty bounds obtained via our Bayesian Dropout Neural Network. We compare networks with varying numbers of widths (K), and hidden layers (l).

Figure 12 column I shows the effect of depth (we only show the network's predictions, not the data here, however these networks are also trained on the Mauna Loa dataset introduced previously). We see that for all three nonlinearities the uncertainty bands tend to decrease as the number of hidden layers increases. This is good as we already know that deeper networks are more expressive (Telgarsky (2015), Telgarsky (2016), Yarotsky (2017), and others) therefore a neural network trained with more layers should be more certain that it is correctly approximating the function than a more shallow network.

In column II, however, we see that width has no effect on uncertainty. For single hidden layer networks with 100, 512, and 1024 neurons and TanH, softplus, and ReLU nonlinearities, we see that the uncertainty bands do not decrease as the width of the network increases.

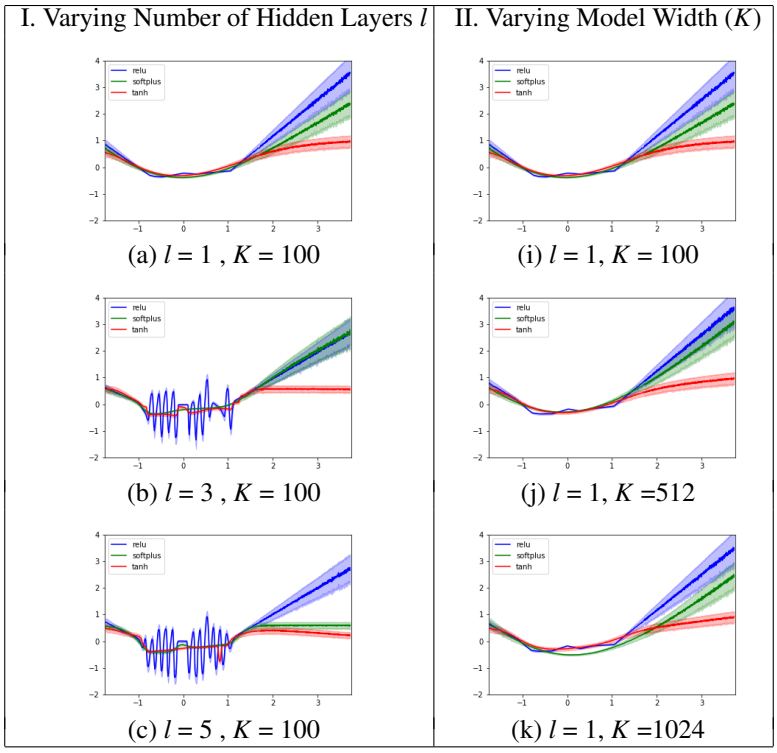


Figure 12: We evaluate model extrapolation with various non linearities, hidden layers (l), and model sizes (K). Shown are predictive mean (thick solid line) and predictive uncertainty (shaded area, showing 2 standard deviations).

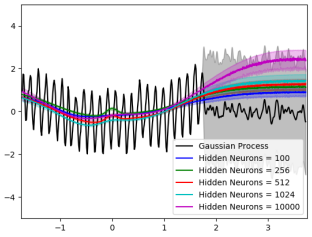
5.6 Depth, Width, and Convergence to Gaussian Processes

The results of Neal (1995) and Williams (1998) show that an infinite width neural network approaches a Gaussian Process with covariance function defined by the choice of activation function. However, in practice we only have finite neural networks. In the next experiment we investigate whether the result holds with finite neurons and whether increasing the width of the network converges to a Gaussian Process. While in the previous section we used ReLU, Softplus, and TanH, Williams (1998) shows that NN with Gaussian activation and infinite width converges to a GP with a scaled squared exponential covariance function. Hence, here we use a Gaussian activation (additionally, ReLU and Softplus are nonbounded so there's no guarantee that Neal (1995)'s result holds for them anyways).

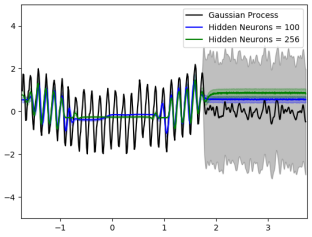
Figure 13 shows that with any number of neurons wide that is feasible to train on a standard laptop (up to 10,000 width), the single layer network doesn't approach a GP with squared exponential covariance. The black line represents a GP with squared exponential covariance trained on the same Mauna Loa dataset we've been using (for readability, the data is not shown on this plot). The other colored lines show predictions of networks with varying widths. This experiment indicates either that (1) you really need infinite neurons for this result to hold or (2) that the scaling in the covariance function Williams (1998) obtains is actually quite important. In either case, the result is not quite as nice since either neural networks that approach GPs are infeasible to use in practice or they approach a GP with a covariance function that is analytic but with less nice properties (such as stationarity because the scaling makes the scaled squared exponential covariance function non-stationary).

Depth on the other hand seems to help a lot. This result makes sense since, as a a number of recent papers demonstrate the expressive power of depth (Telgarsky (2015), Telgarsky (2016), Yarotsky (2017), Liang and Srikant (2016), and Eldan and Shamir (2015)). While proving the result of Neal (1995) and Williams (1998) may be more difficult in the case of deeper-than-one-layer networks, based on this numerical result, they seem to converge even faster than the wide networks Williams (1998) uses.

Figure 13: In practice, depth rather than width seems to yield convergence to a Gaussian process



(a) **NN** (Gaussian Nonlinearity, 1 Hidden Layer); **GP** (Squared Exponential Covariance)



(b) **NN** (Gaussian Nonlinearity, 5 Hidden Layers); **GP** (Squared Exponential Covariance)

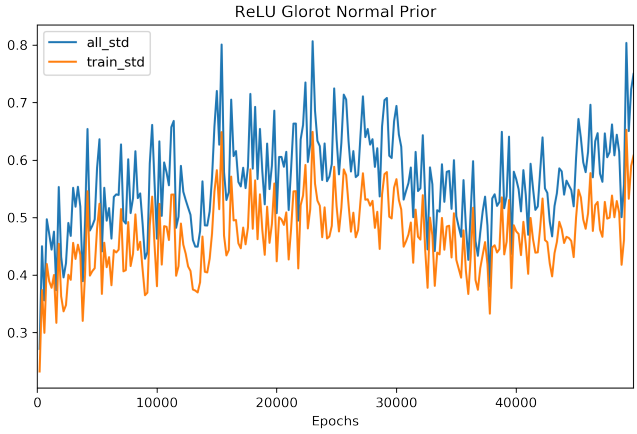
5.7 But is our network learning uncertainy as it trains?

So far, we've shown that the Dropout Bayesian Neural Network produces reasonable estimates of uncertainty in that as one moves away from the training data, uncertainty increases. Additionally, we seen that increasing the depth of the network seems to reduce uncertainty (since the network becomes more expressive) while width has no effect. Finally, we seen that different activation functions and prior distributions induce different shapes of uncertainty onto the neural network's predictions.

However it would also make sense that as a network is trained longer, it should become more confident about its predictions. To test this, we keep track of network uncertainty as the network trains. More precisely, after each 200 epochs of training, we feed 1000 forward passes through the dropout network using the weights of the network at that point in training. We then calculate the standard deviation of the output distribution for each input value x (we average these standard deviations over all inputs x). We continue in this fashion through the training process. While we might expect the neural network to become more confident as it trains more, figure 14 shows that this isn't actually the case.

It's not entirely clear why our network isn't becoming more certain as it trains more. One possibility is that the network knows that as it trains more it will overfit more and hence knows it will perform worse on new data. Another possibility is that the network isn't really *learning* uncertainty but rather that we are just imposing stochasticity onto the network that we choose to interpret as uncertainty because it has some nice properties. We leave these questions as future directions to explore.

Figure 14: Uncertainty Estimates as the Network Trains



6 Conclusion

Existing neural networks lack meaningful notions of uncertainty. This makes them less interpretable, generalizable, and in general worse at the jobs they are designed for. Not only does understanding the uncertainty of a prediction provide useful information about that prediction and the network that made it, it can prevent

catastrophic decisions made on the basis of bad predictions (after all, predictions with some kind of confidence bands are usually interpreted as 100% probability predictions).

In this paper, we first laid out some theoretical justification for modelling uncertainty using a dropout Bayesian neural network. We connected this network to Gaussian processes, a well-understand statistical model. We then showed that dropout is equivalent to variational inference and therefore the dropout neural network is approximating a Gaussian process posterior via variational inference with the Bernoulli distributions used for dropout.

Finally, we presented a number of experiments examining the effect of network structure and weight prior initialization on the uncertainty bands obtained from a dropout Bayesian neural network. In general, we find that the activation function affects the shape of the distribution (does uncertainty fan out linearly or nonlinearly and does the distribution have long or short tails?), greater depth reduces the networks uncertainty while width has no effect, and that the choice of prior can be very important both to train the network and appropriately model uncertainty.

A.1 Re-writing the ELBO

$$\begin{aligned}
E_q(\log p(\theta, y)) - E_q(\log q(\theta)) &= \int q(\theta) \log p(\theta, y) d\theta - \int q(\theta) \log q(\theta) \\
&= \int q(\theta)[\log p(y|\theta) + \log p(\theta)] d\theta \\
&\qquad - \int q(\theta) \log q(\theta) d\theta \\
&= \int q(\theta) \log p(y|\theta) - q(\theta) \log q(\theta) + \log p(\theta) \\
&= \int q(\theta) \log p(y|\theta) \\
&\qquad - \int q(\theta) \log q(\theta) - \log p(\theta) d\theta \\
&= \int q(\theta) \log p(y|\theta) - \int q(\theta) \log \frac{q(\theta)}{p(\theta)} d\theta \\
&= \int q(\theta) \log p(y|\theta) d\theta - KL(q(\theta)||p(\theta))
\end{aligned}$$

A.2 Picking the Transformation and Auxiliary Distribution for SGVB

According to Kingma and Welling (2013), there are three ways to go about this depending on which approximating distribution one uses.

1. For distributions that have a notion of a parameterless "standard" distribution with "location" and "scale" parameters (like the standard Normal),[32] we simply draw ϵ from the "standard" distribution (with location 0 and scale 1). Then $g(\cdot) = location + scale * \epsilon$.
2. If we know the inverse CDF of $q_\lambda(x)$, we can use the inverse transform method. Let $\epsilon \sim U(0, I)$ and $g_\lambda(\cdot)$ be the inverse CDF of $q_\lambda(\theta)$. This works for exponential, cauchy, logistic, rayleigh, pareto, weibull, reciprocal gompertz, gumbel and erland distributions
3. If a distribution is a composition of distributions in cases (1) and (2), we use the composition rule. For example, a Chi-squared distribution is the sum of squared standard Normal random variables so we would use this knowledge in combination with technique (1). Other examples include the Gamma (sum of exponentially-distributed r.v.s), Dirichlet (weighted sum of Gamma-distributed r.v.s), Beta (constructed from Gamma or χ^2 distributed r.v.s), F (ratio of χ^2-distributed rv.s), and Log Normal distributions.

A.3 Proof for Sigmoidal Ativation

This proof comes from Williams (1998) with significantly more explanation.

[32] This includes the Laplace, Elliptical, Student-t, Logistic, Uniform, Triangular, and Gaussian distributions.

Let C be the square root of the matrix Σ^{-1} (which means that $C^T = C$ and $\Sigma^{-1} = C^2$). Also let $\mathbf{u}' = C\mathbf{u}$ (this means that $\mathbf{u} = C^{-1}\mathbf{u}'$ and $\mathbf{u}^T = \mathbf{u}'^T C^{-1^T} = \mathbf{u}'^T C^{-1}$ and $d\mathbf{u} = \frac{d\mathbf{u}'}{C}$. Then,

$$\begin{aligned}
V_{\text{erf}}(\mathbf{x}, \mathbf{x}') &= \frac{1}{(2\pi)^{\frac{d+1}{2}}|\Sigma|^{1/2}} \int \text{erf}(\mathbf{u}^T\mathbf{x}) \,\text{erf}(u^T\mathbf{x}') exp(-\frac{1}{2}\mathbf{u}^T\Sigma^{-1}\mathbf{u}) d\mathbf{u} \\
&= \frac{C}{(2\pi)^{\frac{(d+1)}{2}}} \int \text{erf}(\mathbf{u}^T\mathbf{x}) \,\text{erf}(\mathbf{u}^T\mathbf{x}') \dot{exp}(-\frac{1}{2}\mathbf{u}^T\Sigma^{-1}\mathbf{u}) d\mathbf{u} \\
&= \frac{C}{(2\pi)^{\frac{d+1}{2}}} \int \text{erf}(\mathbf{u}'^T C^{-1}\mathbf{x}) \,\text{erf}(\mathbf{u}'^T C^{-1}\mathbf{x}') \dot{exp}(-\frac{1}{2}\mathbf{u}^T C^2\mathbf{u}) d\mathbf{u} \\
&= \frac{1}{(2\pi)^{\frac{d+1}{2}}} \int \text{erf}(\mathbf{u}'^T C^{-1}\mathbf{x}) \,\text{erf}(\mathbf{u}'^T C^{-1}\mathbf{x}') \dot{exp}(-\frac{1}{2}\mathbf{u}'^T C\mathbf{u}) d\mathbf{u}' \\
&= \frac{1}{(2\pi)^{\frac{d+1}{2}}} \int \text{erf}(\mathbf{u}'^T C^{-1}\mathbf{x}) \,\text{erf}(\mathbf{u}'^T C^{-1}\mathbf{x}') \dot{exp}(-\frac{1}{2}\mathbf{u}'^T\mathbf{u}') d\mathbf{u}' \\
&= \frac{1}{2\pi} \int\int \text{erf}(a_1^T\mathbf{u}') \,\text{erf}(a_2^T\mathbf{u}') \dot{exp}(-\frac{u'^2_1 + u'^2_2}{2}) du'_1 du'_2
\end{aligned}$$

The first line holds by definition of $V(x, x')$. The second line substitutes $C = \Sigma^{-1/2}$. The third line substitutes $\mathbf{u^T} = u'^T C^{-1}$ and $\Sigma^{-1} = C^2$. The fourth line substitutes $du = \frac{du}{C}$ and $u^T = u'^T C^{-1}$. The fifth line substitutes $u = C^{-1}u'$. The line holds by a change of bases (similar to what Cho and Saul (2009) do) such that the basis vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ lie in the plane defined by $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ and all other basis vectors are orthogonal to it. In this basis, $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ are defined only by a linear combination of $\mathbf{e}_1$ and $\mathbf{e}_2$ ($\tilde{\mathbf{x}} = a_{11}\mathbf{e}_1 + a_{12}\mathbf{e}_2$ and $\tilde{\mathbf{x}}' = a_{21}\mathbf{e}_1 + a_{22}\mathbf{e}_2$). Thus $\mathbf{u}'^T\tilde{\mathbf{x}} = a_{11}u'_1 + a_{12}u'_2$ and $u'^T\tilde{\mathbf{x}}' = a_{21}u'_1 + a_{22}u'_2$ (since $\mathbf{e}_1$ and $\mathbf{e}_2$ are just basis vectors). Note that $\mathbf{a}_1 = [a_{11}a_{22}^T]$ and $\mathbf{a}_2 = [a_{21}a_{22}]^T$. The idea here is that we're converting the $d+1$ dimensional integral over all the weights to a two dimensional one.

To compute this integral we're going to construct a slightly more complicated function and then "integrate by differentiating with respect to a parameter."[33].

Let $I(\lambda) = \frac{1}{2\pi}\int \text{erf}(\lambda a_1^T u') \,\text{erf}(a_2^T u) e^{-u'^2/2}$ where $u'^2 = u'^T u'$. Note that $I(1) = V_{\text{erf}}(x, x')$ (the expression given above).

$$\begin{aligned}
I'(\lambda) &= \frac{dI(\lambda)}{d\lambda} \\
&= \frac{1}{2\pi}\frac{2}{\sqrt{\pi}} \int a_1^T u' \,\text{erf}(a_2^T u') exp(-\frac{1}{2}u'^T(I + 2\lambda^2 a_1 a_1^T)u' du' \\
&\text{Let } B = I + 2\lambda^2 a_1 a_1^T, \tilde{u'} = B^{1/2}u', c_1 = B^{-1/2}a_1, c_2 = B^{-1/2}a_2 \\
&= \frac{1}{\pi^{3/2}|B|^{1/2} \int c_1^T\tilde{u'} \,\text{erf}(c_2^T\tilde{u'}) e^{-\tilde{u'}^2/2}} d\tilde{u'} \\
&= \frac{1}{\pi^{3/2}|B|^{1/2} \int (c_{11}\tilde{u'}_1 + c_{12}\tilde{u'}_2) \,\text{erf}(c_2^T\tilde{u'}) e^{-\tilde{u'}^2/2}} d\tilde{u'} \\
&= \frac{1}{\pi^{3/2}|B|^{1/2}} \int (c_{11}\tilde{u'}_1) \,\text{erf}(c_2^T\tilde{u'}) e^{-\tilde{u'}^2/2} d\tilde{u'} + \int (c_{12}\tilde{u'}_2)) \,\text{erf}(c_2^T\tilde{u'}) e^{-\tilde{u''}^2/2} d\tilde{u}
\end{aligned}$$

which we can integrate by parts to get

$$\begin{aligned}
&= \frac{2c_1^T c_2}{\pi^2|B|^{1/2}} \int exp(-\frac{1}{2}\tilde{u'}^T)(I + 2c_2c_2^T)\tilde{u'} d\tilde{u'} \\
&= \frac{4c_1^T c_2}{\pi} \frac{1}{|B|^{1/2}|I + 2c_2c_2^T|^{\frac{1}{2}}} \\
&= \frac{4}{\pi} \frac{a_1^T B^{-1} a_2}{|I + 2\lambda^2 a_1 a_1^T + 2a_2 a_2^T|^{\frac{1}{2}}}
\end{aligned}$$

since $c_1c_2 = B^{-1/2}a_1B^{-1/2}a_2 = a_1^T B^{-1}a_2$ in the numerator and $B = I + 2\lambda^2 a_1 a_1^T$ (and then multiplying out) in the denominator.

[33]This technique popularixed by Feynman is explained in these lecture notes: `https://kconrad.math.uconn.edu/blurbs/analysis/diffunderint.pdf`

Through some very clever substitutions Williams (1998) makes (many of these are tricks that I don't fully understand so I'll refer the reader to Williams' paper), we obtain

$$I'(\lambda) = \frac{4}{\pi} \frac{a_1^T a_2}{1 + 2\lambda^2 a_1^2) \Delta^{\frac{1/2}{]}}}$$

where $\Delta = 1 + 2\lambda^2 a_1^2 + 2a_2^2 + 4\lambda^2 [a_1^2 a_2^2 - (\mathbf{a_1}^T \mathbf{a_2})^2$. With the very clever substitution $\theta = \frac{2\lambda a_1^T a_2}{\sqrt{1+2\lambda^2 a_1^2}\sqrt{1+2a_2^2}}$, we can write a much simpler expression for $I'(\lambda)$

$$I'(\lambda) = \frac{2}{\pi} \frac{1}{\sqrt{1-\theta^1}} \frac{d\theta}{d\lambda} \tag{22}$$

which contains the derivative of sin^{-1} with respect to θ (using the chain rule). So we take the antidrivative and obtain $I(\lambda) = \frac{2}{\pi} \sin^{-1}(\theta)$. Plugging back in $\lambda = 1$ and unsubstituting for θ, he obtains

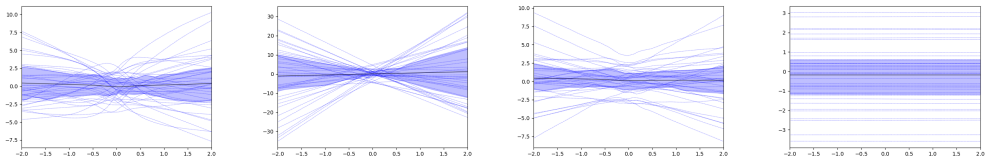
$$V_{\text{erf}}(x, x') = \frac{2}{\pi} \sin^{-1} \frac{2a_1^T a_2}{\sqrt{1 + 2a_1^2}\sqrt{1 + 2a_2^2}} \tag{23}$$

The final step is to convert back to the original space using $x^T x = a_1^2$, $x^T x' = \mathbf{a}_1^T \mathbf{a}_2$, $x'^T x' = a_2^2$, $x = C^{-1}x$, $x' = C^{-1}x'$

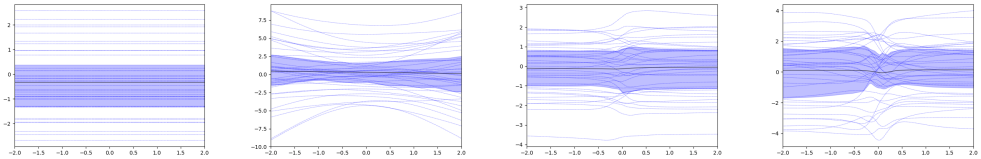
$$\frac{2}{\pi} sin^{-1} \frac{2x^T \Sigma x'}{\sqrt{(1 + 2x^T \Sigma x)(1 + 2x'^T \Sigma x'}} \tag{24}$$

A.4 Additional Experiments with Prior Weight Initialization for Various Kernels

Figure A.1: $W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{N_{in}}}\right)$ (He et al. (2015))

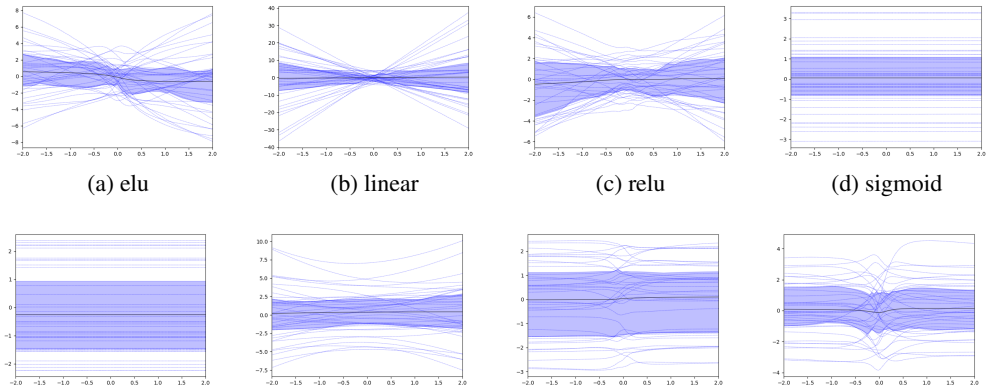


(a) elu (b) linear (c) relu (d) sigmoid



(e) softmax (f) softplus (g) softsign (h) tanh

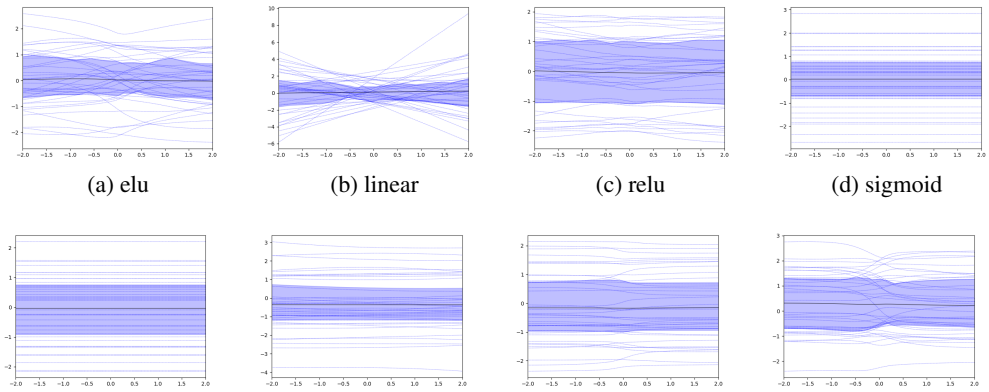
Figure A.2: $W \sim \text{Unif}\left[-\sqrt{\frac{2}{N_{in}}}, \sqrt{\frac{2}{N_{in}}}\right]$ (He et al. (2015))



(a) elu (b) linear (c) relu (d) sigmoid

(e) softmax (f) softplus (g) softsign (h) tanh

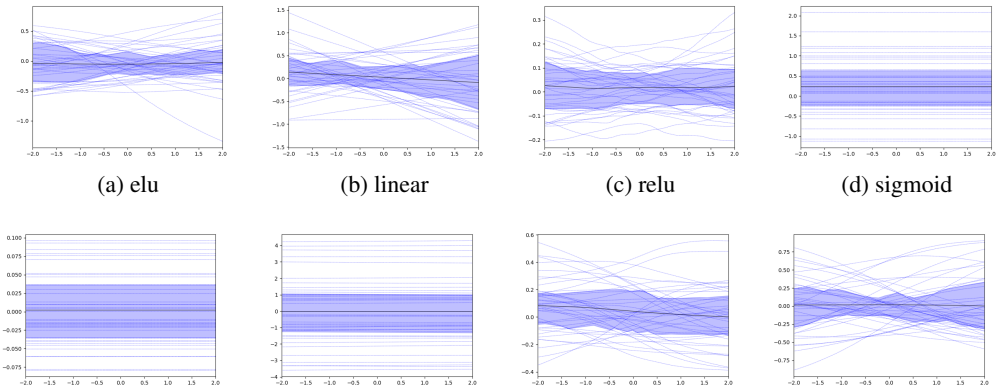
Figure A.3: $W \sim \mathcal{N}\left(0, \sqrt{\frac{1}{N_{in}}}\right)$ (Klambauer et al. (2017))



(a) elu (b) linear (c) relu (d) sigmoid

(e) softmax (f) softplus (g) softsign (h) tanh

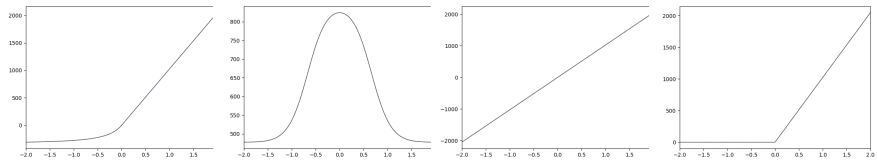
Figure A.4: Truncated ($\leq< 2\sigma$) $\mathcal{N}(0, 0.05)$ (truncated at more than two standard deviations away



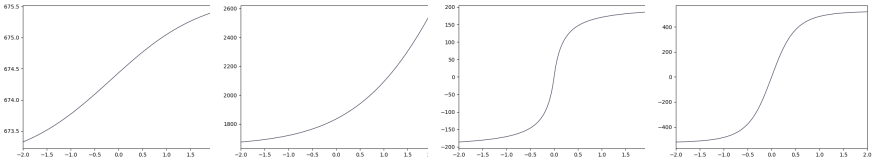
(a) elu (b) linear (c) relu (d) sigmoid

(e) softmax (f) softplus (g) softsign (h) tanh

A.5 Visualization of Each Activation Function



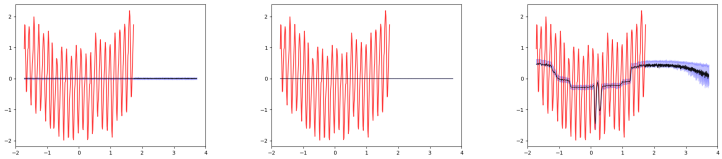
(a) Elu (b) Guassian (c) Linear (d) ReLu



(e) Sigmoid (f) Softplus (g) Softsign (h) TanH

A.6 Additional Posterior Plots (Those that were difficult to train)

Figure A.6: Activation Functions



(a) sigmoid (b) softmax (c) softsign

A.7 The Combined Influence of Prior and Activation Function

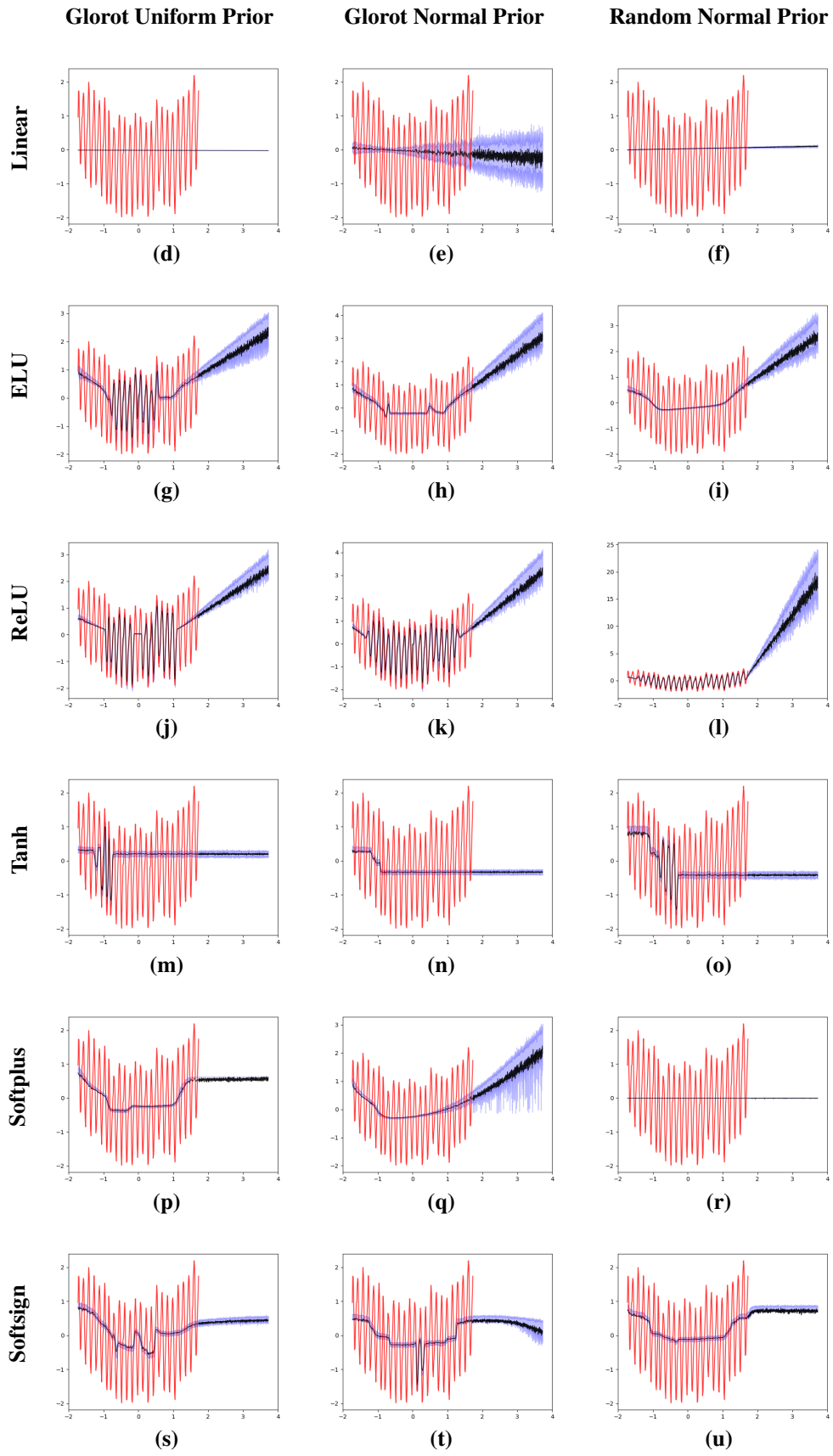


Figure A.7: Activation function v.s. Weight Prior

References

Barber, David, and Christopher M Bishop. 1998. "Ensemble learning in Bayesian neural networks." *Nato ASI Series F Computer and Systems Sciences*, 168: 215–238.

Bishop, Christopher M, Neil D Lawrence, Tommi Jaakkola, and Michael I Jordan. 1998. "Approximating posterior distributions in belief networks using mixtures." 416–422.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe. 2017. "Variational inference: A review for statisticians." *Journal of the American Statistical Association*, 112(518): 859–877.

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

Bok, Brandyn, Daniele Caratelli, Domenico Giannone, Argia M Sbordone, and Andrea Tambalotti. 2018. "Macroeconomic nowcasting and forecasting with big data." *Annual Review of Economics*, 10: 615–643.

Cai, Michael, Marco Del Negro, Marc P Giannoni, Abhi Gupta, Pearl Li, and Erica Moszkowski. 2018. "DSGE forecasts of the lost recovery." *FRB of New York Staff Report*, , (844).

Cho, Youngmin, and Lawrence K Saul. 2009. "Kernel Methods for Deep Learning." In *Advances in Neural Information Processing Systems 22*. , ed. Y Bengio, D Schuurmans, J D Lafferty, C K I Williams and A Culotta, 342–350. Curran Associates, Inc.

Cybenko, G. 1989. "Approximation by superpositions of a sigmoidal function." *Math. Control Signals Systems*, 2(4): 303–314.

Eldan, Ronen, and Ohad Shamir. 2015. "The Power of Depth for Feedforward Neural Networks."

Gal, Yarin. 2016. "Uncertainty in deep learning." PhD diss. PhD thesis, University of Cambridge.

Gal, Yarin, and Zoubin Ghahramani. 2015*a*. "Bayesian convolutional neural networks with Bernoulli approximate variational inference." *arXiv preprint arXiv:1506.02158*.

Gal, Yarin, and Zoubin Ghahramani. 2015*b*. "On modern deep learning and variational inference." Vol. 2.

Gal, Yarin, and Zoubin Ghahramani. 2016*a*. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." 1050–1059.

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

Ghahramani, Zoubin. 2015. "Probabilistic machine learning and artificial intelligence." *Nature*, 521(7553): 452.

Glorot, Xavier, and Yoshua Bengio. 2010. "Understanding the difficulty of training deep feedforward neural networks." 249–256.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

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

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." 1026–1034.

Herbst, Edward, and Frank Schorfheide. 2014. "Sequential Monte Carlo sampling for DSGE models." *Journal of Applied Econometrics*, 29(7): 1073–1098.

Herbst, Edward P, and Frank Schorfheide. 2015. *Bayesian estimation of DSGE models.* Princeton University Press.

Herzog, Stefan, and Dirk Ostwald. 2013. "Experimental biology: sometimes Bayesian statistics are better." *Nature*, 494(7435): 35.

Hinton, Geoffrey E, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580*.

Hornik, Kurt. 1991. "Approximation capabilities of multilayer feedforward networks." *Neural Netw.*, 4(2): 251–257.

Jordan, Michael I, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. 1999. "An introduction to variational methods for graphical models." *Machine learning*, 37(2): 183–233.

Keeling. 2004. "Atmospheric CO2 concentrations (ppmv) derived from in situ air samples collected at Mauna Loa Observatory, Hawaii." `https://www.esrl.noaa.gov/gmd/ccgg/trends/data.html`.

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

Klambauer, Günter, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. "Self-normalizing neural networks." 971–980.

Krzywinski, Martin, and Naomi Altman. 2013. "Points of significance: Importance of being uncertain."

Kullback, S, and R A Leibler. 1951. "On Information and Sufficiency." *Ann. Math. Stat.*, 22(1): 79–86.

Kullback, Solomon. 1959. *Information Theory and Statistics.* New York:John Wiley and Sons, Inc.

Liang, Shiyu, and R Srikant. 2016. "Why Deep Neural Networks for Function Approximation?"

MacKay, David JC. 1992. "A practical Bayesian framework for backpropagation networks." *Neural computation*, 4(3): 448–472.

McAllister, Rowan, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Vivian Weller. 2017. "Concrete problems for autonomous vehicle safety: advantages of Bayesian deep learning." International Joint Conferences on Artificial Intelligence, Inc.

Neal, Radford M. 1993. "Probabilistic inference using Markov chain Monte Carlo methods."

Neal, Radford M. 1995. *Bayesian learning for neural networks.* Vol. 118, Springer Science & Business Media.

NHTSA. 2017. "PE 16-007: Tesla Crash Preliminary Evaluation Report." U.S. Department of Transportation, National Highway Traffic Safety Administration.

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

Telgarsky, Matus. 2015. "Representation Benefits of Deep Feedforward Networks."

Telgarsky, Matus. 2016. "Benefits of depth in neural networks."

Tsuchida, Russell, Farbod Roosta-Khorasani, and Marcus Gallagher. 2017. "Invariance of Weight Distributions in Rectified MLPs."

Williams, Christopher KI. 1998. "Computation with infinite neural networks." *Neural Computation*, 10(5): 1203–1216.

Williams, Christopher KI, and Carl Edward Rasmussen. 2006. *Gaussian processes for machine learning.* Vol. 2, MIT Press Cambridge, MA.

Yarotsky, Dmitry. 2017. "Error bounds for approximations with deep ReLU networks." *Neural Netw.*, 94: 103–114.