Reliable training and estimation of variance networks - NeurIPS 2019 paper reproduction for IFT 6269 Project

Lawrence Abdulnour Isaac Ahouma Emilio Botero Ramon Emiliani Louis-François Préville-Ratelle

Abstract

Neural networks have been shown to perform remarkably well across a wide array of tasks, among which regression is common. The paper "Reliable training and estimation of variance networks", published in NeurIPS by (Detlefsen et al., NeurIPS 2019), proposes a set of tools for predicting the variance over a regression output. For this project, we decided to take the NeurIPS reproducibility challenge and attempted to reproduce the paper's baselines and proposed methodology in their regression tasks. We successfully implemented and tested their method on several regression datasets, achieving similar results.

1. Introduction

Understanding why neural networks work as well as they do is an active area of research. Deep neural networks are often able to take high dimensional data and make accurate predictions by learning hierarchical representations. Although neural networks outputs often achieve state of the art results on a lot of tasks, their outputs are not very interpretable, which in certain settings such as healthcare or autonomous driving is not acceptable. It is therefore desirable to know how confident a neural network is of its predictions: this amounts to estimating the variance over a prediction. In regression, for instance, if the outputs are assumed to follow a distribution, the output is typically the mean. The paper we aim to replicate proposes several complementary methods to additionally estimate the variance over the mean for regression tasks.

In this project, we not only successfully implement the paper's proposed method, but also their baselines, such that we can perform the same comparisons and experiments done by the authors. We focus on the papers regression results, which are based on a toy dataset and two regression datasets from real world data. We obtain very similar results as the paper, which we present in this report.

IFT 6269 project report - Université de Montréal

2. Related work

2.1. Bayesian Neural Networks

Bayesian Neural Networks (BNNs) (MacKay, 1992) assume a prior over the networks' weights. This allows modeling of the uncertainty directly. However, choosing an appropriate prior over the weights is not immediately straightforward, and furthermore running Markov Chain Monte Carlo algorithms over deep neural networks is not computationally tractable.

2.2. Neural networks for uncertainty

The first approaches to variance prediction using neural networks come from the work done by (Nix & Weigend, 1994). This approach simply uses two separate neural networks for predicting both the mean and the variance, where the networks are trained jointly using the negative log-likelihood of the Gaussian distribution as the objective function. Two main common approaches to predict the variance are used. The first approach consists in training a single network to predict the variance in addition to the usual mean-regression model. Thus in that approach, the network predicts two things. The second approach consists in training a separate network (referred to as a variance network) to predict the variance in addition to the usual mean-regression model. Each one of these approaches has different benefits, such as its number of parameters, its capacity, etc. For our work, we use the approach consisting of training two separate networks with the same architecture. The only difference in the networks lies in the choice of the final activation function. The variance network uses the softplus activation in order to enforce positive variance predictions (the variance needs to always be positive).

2.3. Deep ensembles

This method was proposed by (Lakshminarayanan et al., 2016), deep ensembles consist of an ensemble of neural networks with different initializations, where each set of networks predict the mean and the variance separately. At test time, an average of the ensembles' predictions is taken and combined as $p(y \mid \boldsymbol{x}) = M^{-1} \sum_{m_1}^{M} p(y \mid \boldsymbol{x}, \theta_m)$ where M denotes the number of trained models.

2.4. Monte Carlo dropout

Monte Carlo dropout methods (Gal & Ghahramani, 2015) use variational methods to demonstrate that dropout weights before each layer minimize the Kullback-Leibler divergence between am approximate distribution and the posterior of a deep Gaussian process marginalized over it's finite rank covariance matrix. Hence, using dropout before the layers of a neural network yields the posterior distribution, allowing us to estimate the distribution of each weight. To obtain the model's uncertainty, we use dropout over the network's neurons during multiple forward passes. The mean is taken to be the average of the forward passes. For the variance, we estimate the variance of a sample and add to it a constant determined by the number of points, the weight decay coefficient and other hyper-parameters. The authors suggest the method performs better when the data is normalized. For our experiments we restrict the baseline to only use dropout activation at test time and to estimate the mean and variance from the samples.

2.5. Gaussian Processes

Gaussian processes (Murphy, 2012) are well suited and offer a robust alternative for regression problems in a low data regime. Indeed, they provide competitive performance with the advantage of having few hyper-parameters and built-in estimates of uncertainty. However, they are computationally intractable in a large data regime because the computation of the covariance matrix becomes expensive. In fact, these models are non-parametric and thus their computation cost grows with the number of samples. Sparse Gaussian Processes, on the other hand, attempt to partially solve this problem by selecting a subset of the dataset, where samples selected are inducing points. However, this method becomes problematic and impractical when dealing with high-dimensional data (Snelson & Ghahramani, 2005; Quiñonero Candela & Rasmussen, 2005). Furthermore, other parametric models, such as neural networks, still prove to have better mean predictions.

3. Methodology

The paper assumes that the datasets are of the form $\mathcal{D} = \{\mathbf{x_i}, y_i\}_{i=1,\dots,N}$ where the data points are i.i.d. with inputs $\mathbf{x_i} \in \mathcal{R}^d$ and targets $y_i \in \mathcal{R}$. Moreover, the targets are assumed to be conditionally Gaussian given $\mathbf{x}$:

$$p_\theta(y|\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})),$$

where $\mu(\mathbf{x}), \sigma^2(\mathbf{x})$ are continuous functions, and in particular neural networks. Furthermore, $\theta = (\theta_\mu, \theta_{\sigma^2})$. As mentioned previously, the main goal of this paper is to approximate the function $\sigma^2(\mathbf{x})$. Since $\sigma^2(\mathbf{x})$ is continuous, this function shouldn't change too much in a close neighborhood of x. Of course, the size of that neighborhood depends on how much $\sigma^2(\mathbf{x})$ varies around x. These two functions are obtained using maximum likelihood (MLE).

The authors start by mentioning that their approach is a local likelihood model. More precisely, they state that the local likelihood estimation problem at a point $(\mathbf{x}_i, y_i)$ is equal to

$$\log \tilde{p}_\theta(y_i|\mathbf{x}_i) = \sum_{j=1}^{N} w_j(\mathbf{x}_i) \log p_\theta(y_j|\mathbf{x}_j), \qquad (1)$$

where w_j is a similarity function (weakly positive) that declines as $||\mathbf{x}_j - \mathbf{x}_i||$ increases. First, the LHS depends on y_i but the RHS doesn't. Second, it turns out that the left hand side is not what the local likelihood estimation really is. After looking at one of the two references of this equation given by the authors (see page 60 of (Loader, 1999)), we realized that the correct equation should be:

$$\mathcal{L}_x(\theta) = \sum_{j=1}^{N} w_j(\mathbf{x}) \log p_\theta(y_j|\mathbf{x}_j), \qquad (2)$$

where the weights $w_j(\mathbf{x}) \geq 0$ decrease as $||\mathbf{x}_j - \mathbf{x}_i||$ increase and sum to 1. Basically, $\mathcal{L}_x(\theta)$ counts the (weighted) average local likelihood around the point x given the parameters θ.

3.1. The locality sampler

The paper first notes a problem with the standard mini-batching scheme when training neural networks. At each iteration, a random sample of points in the dataset is taken, over which the forward and backward passes are computed. The authors note that if there isn't enough data around a point in the mini-batch, while it is possible to have a good idea of the local mean, getting a good approximation of the variance is difficult. They propose a local sampling scheme that we describe next.

The idea is then to sample a random point $\mathbf{x}$ from the dataset, and then construct the mini-batch using the $k-$nearest neighbors of $\mathbf{x}$. However, to allow for more variability in the mini-batches, first m points are sampled randomly, and then for each of these, n points are sampled from their $k-$ nearest neighbors. Note that n and k are two other hyper-parameters that need to be decided also. This will be explained later, but it turns out that the hyper-parameters (m, n, k) are different for training $\mu(x)$ than the ones for training $\sigma^2(x)$. In the code of the authors, these are hard coded. This seems to be a limitation of the algorithms presented in this paper.

For each mini-batch, since we assume a Gaussian distribution for the y's given any x, the log-likelihood of this sampling procedure would be

$$\sum_{\mathbf{x}_j \in \mathcal{O}} \Big[-\frac{1}{2} \log(\sigma^2(\mathbf{x}_j)) - \frac{(y_j - \mu(\mathbf{x}_j))^2}{2\sigma^2(\mathbf{x}_j)} \Big], \qquad (3)$$

where $\mathcal{O}$ is a mini-batch. However, as mentioned by the authors, there is a problem with this formula. Some points will be sampled more often than others. To correct this, they apply the Horvitz-Thompson algorithm (Horvitz & Thompson, 1952), which is putting a weight π_j on each term of the log likelihood that is equal to the inverse of the probability that each point $\mathbf{x}_j$ is sampled. This weighted log likelihood function is given by

$$\sum_{\mathbf{x}_j \in \mathcal{O}} \frac{1}{\pi_j} \Big[-\frac{1}{2} \log(\sigma^2(\mathbf{x}_j)) - \frac{(y_j - \mu(\mathbf{x}_j))^2}{2\sigma^2(\mathbf{x}_j)} \Big], \quad (4)$$

where the inclusion probability π_j is equal to

$$\pi_j = \frac{m}{N} \sum_{i=1}^{N} \frac{n}{k} \mathbf{1}_{j \in \mathcal{O}_k(i)}, \quad (5)$$

and $\mathcal{O}_k(i)$ is the set of the k closest points to $\mathbf{x}_i$.

3.2. Training

The training procedure consists of first optimizing θ_μ after fixing a small value for σ^2. Then, they optimize θ_μ and θ_{σ^2} alternately. More precisely, they optimize θ_{σ^2} while μ is fixed and vice versa for θ_μ and σ^2. To justify this process, the authors mention that it is better than optimizing both parameters at the same time because if there is a single point in a region by example, it is impossible to train both μ and σ^2 simultaneously.

The paper imposes a prior over the variance, assuming it is distributed as an inverse Gamma. This choice is justified since this is the conjugate prior of the variance when the data is assumed to follow a Gaussian distribution. To estimate $\sigma(x)^2$, they thus train two neural networks, one for $\alpha(x)$ and the other for $\beta(x)$, where α and β are the parameters of the inverse-Gamma distribution. The log likelihood is then:

$$\begin{aligned} &\log p_\theta(y_i) \quad (6) \\ &= \log \int \mathcal{N}(y_i|\mu_i, \sigma_i^2) \frac{\beta^\alpha}{\Gamma(\alpha)} (\sigma_i^2)^{-\alpha+1} \exp\big(-\frac{\beta_i}{\sigma_i^2}\big)\, d\sigma_i^2 \\ &= \log t_{\mu_i, \alpha_i, \beta_i}(y_i), \end{aligned}$$

where $\frac{\beta^\alpha}{\Gamma(\alpha)} (\sigma_i^2)^{-\alpha+1} \exp\big(-\frac{\beta_i}{\sigma_i^2}\big)$ is the probability density of the inverse-Gamma distribution with parameters α and β. It is important to note here that $\mathbf{x}_i$ doesn't appear to alleviate notation of the formula, but it is implicit everywhere. They justify using this approach because in low data regime, it is better to be Bayesian and find a distribution for σ^2 instead of a point estimate. They also say that few data points locally would lead to underestimating σ^2 due to the left skewness of the gamma distribution of $\hat{\sigma}_i^2 = (y_i - \mu(x_i))^2$.

Here, we need to explain one thing that we did differently. While training, the authors take 50 samples from the inverse-Gamma distribution for each data point in a mini-batch, and train their networks using equation 4, where the sampling operation is differentiable through the reparametrization trick. We instead train our networks using equation 6.

3.3. Extrapolation trick

To bound the variance away from data points, they impose the variance to tend to a chosen hyperparameter η when going away from the data points. The authors mention that they are inspired by Gaussian processes, where the covariance function is stationary. Similarly to sparse GP, they obtain some centroids $\{c_i\}_{i=1}^L$ using K-means. Let $\delta(x_0) = min_i||c_i - x_0||$. This is the smallest distance from x_0 to any of the centroids. Let

$$\hat{\sigma}^2(x_0) = (1 - \nu(\delta(x_0)))\hat{\sigma}_\theta^2 + \eta\nu(\delta(x_0)), \quad (7)$$

where ν is the scaled and translated sigmoid function $\nu(x) = \text{sigmoid}((x + a)/(\gamma))$, γ is a parameter that is optimized during training and $a = -6.9077\nu$ to get $\nu(0) \approx 0$. The authors also mention that the c_i are optimized during training. But after looking at their code for some of the methods, we realized that the centroids are obtained using K means and are then fixed for the rest of the algorithm, which was surprising.

3.4. Datasets

3.4.1. Toy regression

We test our implementation of the paper's method, as well as their baselines, on three of the datasets used in the paper[1]. The first one is a toy dataset of the form

$$y = x \sin(x) + \epsilon_1 + x\epsilon_2$$

Where $\epsilon_1, \epsilon_2 \sim \mathcal{N}(0, 1)$. As in the paper, we sample 500 points drawn uniformly at random in the interval $[0, 10]$ for training, and test on 5000 points drawn uniformly at random in the interval $[-4, 15]$.

3.4.2. Weather data

To show that their method does not only work on the previous toy dataset, the paper experiments on a weather dataset consisting of daily maximum temperature measurements taken over 130 years[2]. They take a "ground truth" of the variance each day as being the sample variance for each day

[1] Our code can be found here: https://github.com/ramonemiliani93/uncertainty. Note that some experiments are on different branches.

[2] https://mrcc.illinois.edu/CLIMATE/Station/Daily/StnDyBTD.jsp

of the year over the 130 years. As training data, for each day of the year the authors randomly sample a max temperature value from the historical data. However, it happens that some times the sampled values is not a number (NaN), in that case it is removed. In the end, the training data consists of between 360 to 366 values.

3.4.3. Boston House Prices (UCI)

The Boston House Prices features data collected by the U.S Census Service concerning housing in the area of Boston (including suburbs or towns). The task is to predict the median value of a house based on 13 features, among which are per capita crime rate by town, average number of rooms per dwelling, weighted distances to five Boston employment centers, and others. The dataset features 506 samples, which are split into a training/testing set using a 80/20 split.

3.5. Implementation details

All neural network based models were implemented in Pytorch (Paszke et al., 2019) and trained using a simple multilayer perceptron with 1 hidden layer and 50 hidden neurons, except the Gaussian Processes methods (GP and SGP) and Bayesian Neural networks method (BNN) which were implemented using separate libraries. We trained our models for 50000 epochs with the Adam optimizer (Kingma & Ba, 2014). In contrast to the paper, which uses sigmoid units and leaky ReLUs as non-linearities for different datasets, we use the sigmoid consistently throughout all datasets and methods.

4. Results and discussion

In figure 1 we show our results on the toy regression dataset (the first 2 rows) and then the paper's results. More precisely, the red dots represent the data points, the blue line is the true mean and the black line indicates the mean predicted by the algorithm. The green lines indicate the true uncertainty (defined as the area under curve between $\pm 2 \times \sigma_{true}$) and the shaded region shows the uncertainty predicted by the algorithm. As we observe, we have similar results and the mean seems to be always well predicted. As contrary however, only the ensemble of NNs, NN's and combined predict well the uncertainty.

Figure 2 shows the true standard deviation for the toy regression given the homoscedastic and heteroscedastic noise. The true deviation is equal to $\text{Var}(y) = 0.3\sqrt{1 + x^2}$. We can see that the methods that best adjust the standard deviation inside the regime of training data are, ensembles (ENS-NN), the uncertainty aware neural network (which we refer to as NN) and the authors proposed method (we refer to it as Combined). All of the other methods underestimate the variance. Furthermore, the other methods' predictions seem

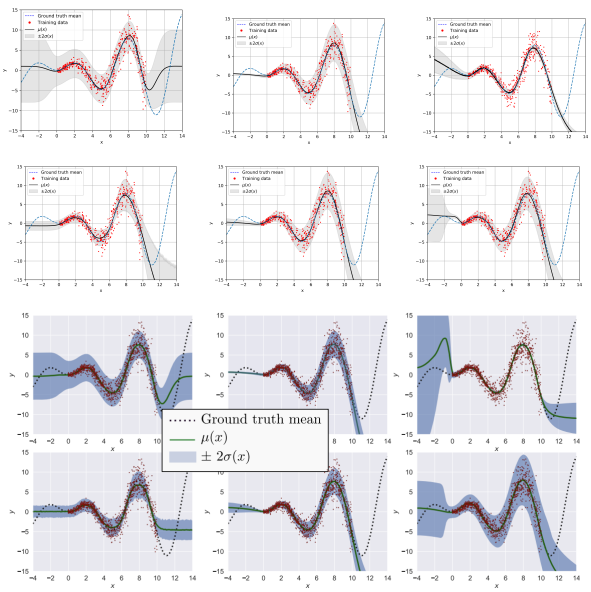

Figure 1. Regression on toy dataset. The first two rows correspond to our results, and the last two to the paper's results. For each set of rows, from top left to bottom right the methods are: Gaussian Process, Neural Network, Bayesian Neural Network, Monte Carlo Dropout, Ensemble of NNs, Combined.

not to follow the actual trend of the true standard deviation. This could raise problems for example on tasks such as safe imitation learning where the switch between an expert and a learner depends on the uncertainty, leading to crashing (in the context of autonomous vehicles). The other conclusion to draw from the figure is that the variance for the combined method is actually bounded, mimicking the behavior of Gaussian processes. Even though the bound is a hyperparameter to adjust, by knowing its value out of distribution, further action can be taken when getting near the bound (again, in the context of autonomous vehicle the learning agent could simply stop). The baselines we developed have a very similar behavior to the ones the authors use, which suggests that the results are correct. Finally, it should not be concluded that the method proposed in the paper works better than the baselines, since some methodologies that are suggested on the original work of the baselines was not included in the author's experiments.

The first two rows in figure[3] show how our implementations of the different methods performed on the weather dataset. As we observe, all methods are able to predict the mean relatively well, however the ensemble method and the combined method have better performances on the uncertainty task. For each method, we also compute an error

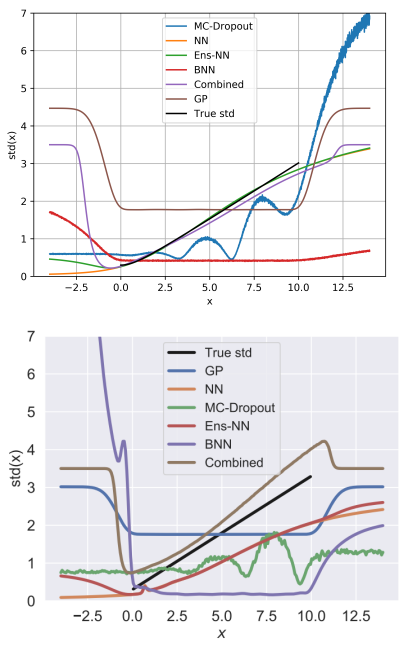

Figure 2. Standard deviation estimates as a function of $\mathbf{x}$ for the toy regression experiment. The first row is our results, the second row is the corresponding figure from the original paper.

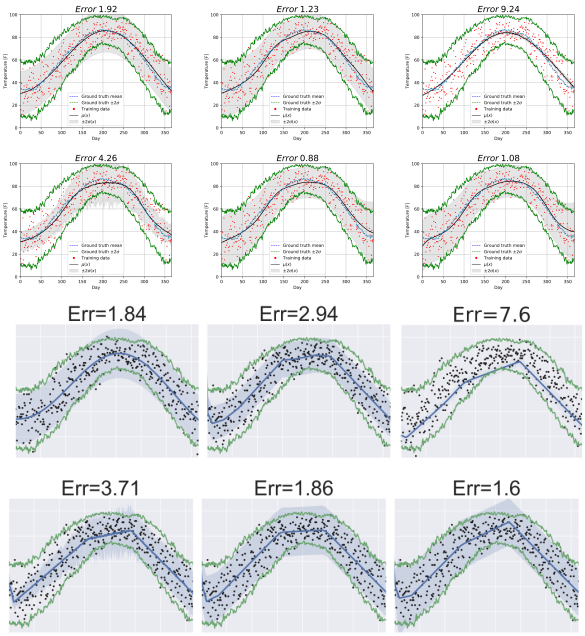

Figure 3. Regression on weather dataset. The first two rows correspond to our results, and the last two to the paper's results. For each set of rows, from top left to bottom right the methods are: Gaussian Process, Neural Network, Bayesian Neural Network, Monte Carlo Dropout, Ensemble of NNs, Combined.

which is defined by the authors as:

$$Err = \frac{1}{N}\sum_{i=1}^{N} = |\sigma^2_{true}(x^{(i)}) - \sigma^2_{estimate}(x^{(i)})|$$

but in fact, it is computed and reported by them as:

$$Err = \frac{1}{N}\sum_{i=1}^{N} = |\sigma_{true}(x^{(i)}) - \sigma_{estimate}(x^{(i)})|$$

In our experiment, the ensemble method has the lowest error (even lower than the author's method), contradicting the claim made by the paper.

Looking more closely at the computed errors, we can see that they differ from those published in the paper. We can explain this by the fact that we performed the experiment with the sigmoid activation instead of the leaky relu activation as in the paper. Surprisingly, it also appears that the sigmoid activation works better for this dataset, hence why some of the errors reported in figure 3 are smaller than the ones reported in the paper.

Mean Test log-likelihoods on the Boston UCI Regression Dataset		
	Ours	**Paper**
sGP	**-1.93**	-1.85
GP	-2.01	**-1.76**
NN	-4.34	-3.64
BNN	-	-2.59
MC-Dropout	-4.23	-2.51
Ens-NN	-4.22	-2.45
Combined	-3.53	-2.09

Table 1. Comparison of the log-likelihoods achieved on the Boston UCI Regression

Table 1 summarizes the log-likelihoods obtained on the Boston UCI Regression dataset by our implementations and those of the authors. When comparing our test log-likelihoods with the paper's results, we see that we have coherent and similar findings. We observe that both GP and SGP's perform best on the Boston UCI dataset, as they have the highest test log-likelihoods. This is unsurprising since, these are powerful models in low-data settings (the boston dataset only has 506 samples). This observation is true for both our log-likelihoods and the paper's log-likelihoods. However, as we have mentioned earlier, these models are intractable when there is a lot of samples available and less

well suited for mean prediction. The combined model then comes into play in this situation, confirming the author's contribution to uncertainty prediction. Indeed, we obtain that after GP's, the combined model, which is the paper's method, performs better in comparison to all of the other baselines. Even though we obtain different numbers than those of the author's experiments, due to different hyperparameter settings, these numbers still manifest logical conclusions. For instance, we notice that the NN baseline performs worse than all of the other models, which is expected as it possesses less capacity.

5. Conclusion

In conclusion, we have successfully reproduced the paper's method and baselines on the main regression tasks. This paper proposes the use of four complementary mechanisms to achieve uncertainty prediction. These mechanisms include a locally aware mini-batch gradient descent, the mean-variance split training, the inverse-Gamma estimation of variance and the extrapolation architecture. When out of distribution, the technique proposed aims to replicate the Gaussian processes behaviors by bounding the variance.

While the authors seemingly provide an improvement over previous approaches for uncertainty estimation on the datasets presented, for the reasons stated previously, we can't conclude with certainty that their proposed method is better than the baselines. It is also important to mention that there are still a few limitations. The first one is that the authors didn't test on higher dimensional data with a low data regime. In that case, their proposed model could be a bit problematic, as sampling higher dimensional data is a harder problem. Secondly, this paper also focuses on both epistemic and aleatoric uncertainty altogether without distinguishing the two types of uncertainty, which is less beneficial for real life applications.

References

Detlefsen, N. S., Jørgensen, M., and Hauberg, S. Reliable training and estimation of variance networks, NeurIPS 2019.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

Horvitz, D. G. and Thompson, D. J. A generalization of sampling without replacement from a finite universe. *J. Amer. Statist. Assoc.*, 47:663–685, 1952. ISSN 0162-1459.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2014.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. *ArXiv*, abs/1612.01474, 2016.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Loader, C. *Local regression and likelihood*. New York: Springer-Verlag, 1999. ISBN 0-387-9877.

MacKay, D. J. C. A practical bayesian framework for backpropagation networks. *Neural Comput.*, 4(3):448–472, May 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.3.448. URL http://dx.doi.org/10.1162/neco.1992.4.3.448.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

Nix, D. A. and Weigend, A. S. Estimating the mean and variance of the target probability distribution. *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, 1:55–60 vol.1, 1994.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlch Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Quiñonero Candela, J. and Rasmussen, C. E. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, December 2005. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1046920.1194909.

Snelson, E. and Ghahramani, Z. Sparse gaussian processes using pseudo-inputs. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pp. 1257–1264, Cambridge, MA, USA, 2005. MIT Press. URL http://dl.acm.org/citation.cfm?id=2976248.2976406.