Modeling Uncertainty in RNNs for Time Series Forecasting

George Adam
University of Toronto
alex.adam@mail.utoronto.ca

Shadi Zabad
University of Toronto
shadi.zabad@mail.utoronto.ca

Tahmid Mehdi
University of Toronto
tfmehdi@cs.toronto.edu

Abhishek Tiwari
University of Toronto
a.tiwari@mail.utoronto.ca

Abstract

Modeling uncertainty in neural networks has recently become a central theme of much fruitful research in the machine learning literature. This surge of interest is partially motivated by the crucial role that uncertainties play in many practical applications, especially in areas such as Active and Reinforcement Learning. Despite this flurry of research, there has been relatively little work done on modeling uncertainty in Recurrent Neural Networks (RNNs) in the context of time series forecasting. RNNs present a unique challenge due to their optimization paradigm known as Backprop through time. A couple of approaches to representing uncertainty in RNNs have been proposed, but the analysis done in the corresponding publications is limited to epistemic uncertainty. We instead focus on three main types of uncertainty: optimization uncertainty, architecture uncertainty, and data uncertainty. The approach we advocate is quite general and can be used for domains outside of time series analysis. It may also be easily extended to other neural network architectures.[1]

1 Introduction

Over the past couple of decades, Gaussian Processes (GPs) gained tremendous traction in the machine learning community as an effective, non-linear method for modeling time series data. Formally, a GP defines a prior over functions by assuming observations are realizations of normally distributed random variables, any finite number of which have a joint Gaussian distribution [14]. GPs are parametrized by mean and covariance functions and the posterior naturally provides uncertainty estimates over the predictions. Despite these advantages, it is known that the performance of GPs depends crucially on the kernel function, the specification of which remains a difficult task. Additionally, GPs have traditionally lagged behind neural networks due to issues with computational cost and scaling to big datasets. Therefore, our work was motivated in part by the desire to replicate some of the attractive properties of GPs, especially with regards to expressing uncertainty, in the context of RNNs.

As advances in optimization procedures and hardware allow for deeper neural architectures to be efficiently trained, neural networks have become increasingly less interpretable, especially for tasks not related to vision. The prevalence of neural networks in everyday technologies, as well as their recent adoption for autonomous driving, has transformed these models from theoretical abstractions

[1] Data, code, and implementation details can be found on GitHub: https://github.com/shz9/csc2541-ml-project

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

of the brain to effective algorithms in many domains. Thus, it is important to understand when a model might fail based on its parameters and the features of a particular test example. Avoiding this question can have a spectrum of consequences ranging from inaccurate voice search to potentially deadly car accidents. Whatever the task may be, modeling uncertainty can warn users about when a machine learning model might make a poor decision.

One way of modeling uncertainty in neural networks is by using Bayesian neural networks (BNNs) [10]. However, BNNs can be computationally expensive for complex architectures and have only recently been studied in the context of recurrent architectures [4, 5]. It is also worth noting that these approaches focus on capturing epistemic uncertainty, which is the uncertainty in the learned parameters of a model. We instead focus on modeling 3 other types of uncertainty: optimization uncertainty, architecture uncertainty, and data uncertainty. Optimization uncertainty addresses the issue that gradient-based optimization methods are limited to finding local optima. Architecture uncertainty reflects our ignorance in choosing an optimal setting of hidden layers, hidden units, activation function, etc. Finally, data uncertainty is based on skepticism regarding the accuracy of the data collection process.

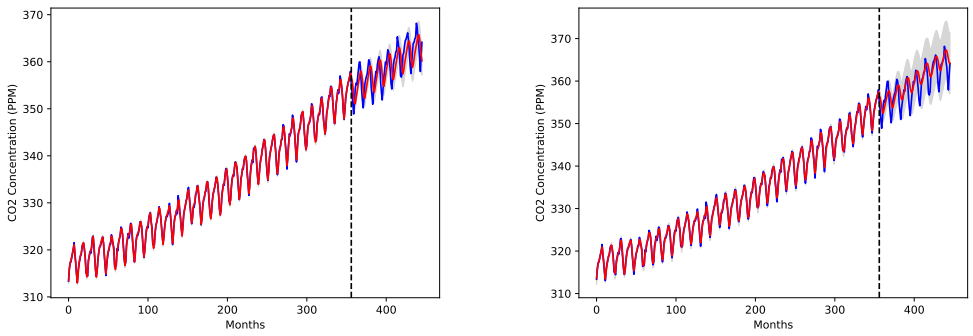
2 Related Work

Fortunato et al. [4] introduced Bayesian RNNs as an alternative technique to model uncertainty. They extend Bayes by Backprop (BBB) by employing forward propagation and backpropagation on minibatches of truncated sequences of the data and minimizing the variational free energy:

$$L(\theta) = -\mathbb{E}_{q(\theta)}\left[\log \prod_{b=1}^{B}\prod_{c=1}^{C} p(y^{(b,c)}|\theta, x^{(b,c)})\right] + w_{KL}^{(b,c)} KL[q(\theta)||p(\theta)]$$

where B and C are the number of minibatches and truncated sequences, respectively, and (b,c) represents the elements of the b$^{\text{th}}$ minibatch and c$^{\text{th}}$ truncated sequence. θ is the vector of RNN parameters, KL is a regularizer and $w_{KL}^{(b,c)}$ denotes the weight of the KL cost. For each minibatch, the parameters are sampled from $q(\theta)$ which is usually assumed to be Gaussian. This method essentially combines the epistemic uncertainty modeling capabilities of BNNs with the ability of RNNs to capture temporal contexts in data. Our work exploits the effectiveness of RNNs for forecasting and models uncertainty in optimization algorithms, the RNN architecture and the data.

Gal and Ghahramani [5] proposed modeling uncertainty in neural networks with dropout. We experimented with this approach and found that this method requires careful choosing of the dropout rate which has a significant effect on the confidence intervals generated around the predictions. The effect of dropout rate on confidence intervals can be seen in figure 1. Though this may be an effective way to model uncertainty in the model, it doesn't capture all dimensions of uncertainty that we will discuss below.



(a) Dropout rate of 0.1

(b) Dropout rate of 0.4

Figure 1: A comparison of how dropout rate can affect confidence intervals when using MC dropout.

The ensemble approach discussed in this paper was inspired by and builds on the bootstrapping method developed by Osband et al. [11] for deep exploration in Reinforcement Learning. We

demonstrate that the same principle applies in the context of recurrent architectures and show that it can be easily extended to other problem domains.

3 Our Approach

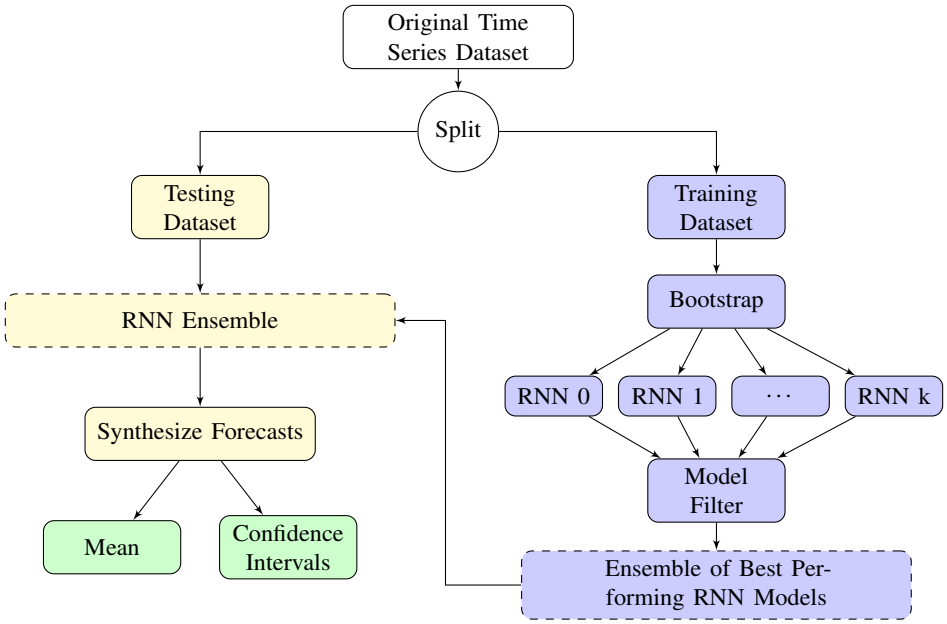


Figure 2: A schematic flowchart illustrating the bootstrap method proposed in this paper.

Figure 2 shows the general approach we employ to model various types of uncertainty in RNNs. To model the 3 dimensions of uncertainty discussed below, we create an ensemble of RNNs that are trained with different weight initializations, different hyperparameter settings, or by using bootstrapped samples of the data. This allows us to model uncertainty in the optimization procedure, architecture, and data respectively.

Algorithm 1 RNN Ensemble Algorithm

```
procedure RNN_FORECASTS(B, ensemble_type)
    models ← [ ]
    data, init_weights, hyperparams ← generate_bootstrap_samples(B, ensemble_type)
    for d, w, h in data, init_weights, hyperparams do
        m ← train_model(d, w, h)
        models.append(m)

    filtered_models ← filter_rnn_models(models)

    predictions ← [ ]
    for m in filtered_models do                    ▷ Generate predictions on test set
        pred ← m.predict(test_set)                 ▷ Pred is a list of forecasted values
        predictions.append(pred)
    μ ← mean(predictions)
    σ ← std(predictions)
    return μ, σ                                    ▷ μ and σ are both vectors
```

Algorithm 1 describes the general procedure for how we use ensembles of networks to obtain uncertainty estimates. Depending on the ensemble type, either the data, initial weights, or network hyperparameters become a list of different settings, while the other arguments stay the same. For example, if the ensemble type is "bootstrap", then the data becomes a list of B different bootstrap

samples, while the initial weights are randomly chosen and the hyperparameters are those determined to be the best via cross-validation. This results in B different models being trained on various bootstrap samples of the data. Alternatively, if the ensemble type is "initial weights", B different models are trained using various random initializations of the weights one the entire training dataset and with a fixed set of optimal hyperparameters. Lastly, if the ensemble type is "hyperparameters", B different models are trained using a list of various hyperparameter settings, each of which provides reasonable validation error. In all three cases, we take the resulting B models, make predictions on the test set, and use those predictions to obtain a mean and standard deviation.

3.1 Uncertainty in the Data

In many real-world applications, learning algorithms have to work with small datasets where factors such as measurement error and sampling error tend to have more pronounced effects. To make the learned model more robust to these effects, various regularization techniques, such as dropout and weight decay, have been proposed. However, many of these approaches end up learning a single representation of the data and are thus unequipped to express uncertainty in the predictions of the model.

To capture the uncertainty inherent in the data and express that uncertainty in model predictions, we turned to a classical approach in the field of statistics: The bootstrap. The bootstrap is a method that is traditionally used to estimate statistical parameters by means of re-sampling the data with replacement [2]. In this paper, we propose to use similar logic to quantify the uncertainty in the input data.

It should be noted, however, that the theory of the bootstrap assumes that the observations are independent and identically distributed [2]. Since we are interested in modeling and forecasting time series data, where observations are temporally correlated, we need to apply bootstrapping methods that preserve at least some of the inherent temporal structure. Over the past few decades, statisticians and econometricians developed a variety of bootstrap methods for time series data (see [8] for review). In our experiments, we tested both parametric as well as non-parametric bootstrap methods with varying degrees of success.

3.1.1 Non-parametric Bootstrap

We explored several non-parametric bootstrap approaches for time series data. The simplest approach is the Block Bootstrap (BB) method [9], which breaks the times-series into non-overlapping blocks of equal size and re-samples those blocks with replacement. Other popular varieties that build on the Block Bootstrap method include the Moving Block Bootstrap, the Stationary Block Bootstrap, and the Circular Block Bootstrap.

One of the difficulties we faced with Block-based bootstrap approaches is that they tend to produce samples with sharp discontinuities. This is problematic for our purposes, because we're modeling differences in the time series rather than the values themselves. To get around that, we also experimented with the Continuous-Path Block Bootstrap method [12].

Due to the non-stationary nature of the datasets that we experimented with, the non-parametric bootstrap approaches tended to produce poorer results than the parametric bootstrap approach that will be outlined in the next section.

3.1.2 Parametric Bootstrap

In the parametric bootstrap setting, a parametric model is fit to the data and the bootstrap samples are generated by sampling from that model instead. In our case, we adopted the approach outlined in [7], which uses Gaussian Process regression to fit a model to the data and generates samples from the posterior of the GP.

We experimented with two methods for generating samples from the posterior of the GP. The first method involved using time points that are slightly different from the ones in the original dataset. The second method involved generating samples for the same time points, given that appropriate Tikhonov regularization is applied to the GP model.

Although adopting this parametric bootstrap approach may seem to render the remainder of our ensemble method unnecessary, it should be noted that the GP bootstrap can produce reasonable samples even when using kernels that perform poorly in forecasting. That's primarily due to the fact our samples are generated for time points that are the same as or relatively close to the original time points.

3.2 Uncertainty in the Architecture

In the process of designing and building machine learning models, we are often uncertain about the structure that those models should take. The plethora of discussions on optimal design decisions for neural networks in technical blogs, Q/A forums, and many other mediums attest to the fact that this question cannot be settled by simple rules of thumb.

The preferred approach to reduce this uncertainty is to use cross-validation with grid or random search over the hyperparameter space. More recently, an approach based on Bayesian optimization [16, 17] was shown to be a promising alternative. Despite the effectiveness of these approaches in some practical settings, there is a lingering question as to whether they manage to eliminate the uncertainty about the architecture altogether or not.

Our approach to modeling uncertainty in model architecture involves building an ensemble of minimally viable architectures and using the variance in their predictions as proxy for that uncertainty. What is meant by minimally viable may differ from one problem domain to the next. In our case, where we are concerned with time series forecasting, we fixed an RMSE threshold that all models in the ensemble should cross.

3.3 Optimization Uncertainty

Gradient descent and its variants are all local optimization strategies when used for non-convex problems [15]. Furthermore, enhancements such as momentum [13] or stochastic optimization techniques (e.g. Adam [6]) can lead to different solutions than batch gradient descent when trained for the same number of epochs. This implies that the set of parameters learned is dependent on their initialization and thereby suggesting that the random initialization of parameters can be considered to be a source of uncertainty. Figure 3 shows how initializations affect gradient ascent solutions. It's important to notice that even when there is a global optimum that can be reached, it is not reachable from every weight initialization. Additionally, even if it is reachable from several initializations, the number of epochs required to reach the optimum from the various initializations can differ by orders of magnitude.

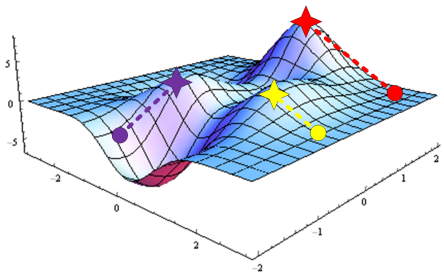


Figure 3: An illustration of how gradient ascent applied to a function with multiple maxima can result in different solutions.

Our approach for quantifying this uncertainty is to train an ensemble of neural networks with identical hyperparameters, but using different random weight initializations. The predictions of these neural networks can then be used to compute a mean prediction and standard deviation at each time step in the series. Optimization uncertainty is different from the epistemic uncertainty modeled by Bayesian Neural Networks which can be reduced as more data is obtained. It instead models the variability in the weight initializations of an optimization procedure.

4 Methods

4.1 Data Preprocessing

Figure 4 shows how the data was preprocessed for the RNN experiments. First, a sequence of values $S = \{x_1, ..., x_t\}$ is differenced to obtain a sequence of differences $D = \{d_1, ..., d_{t-1}\}$ where $d_i = x_{i+1} - x_i$. The differencing operation is required to make a time series stationary, and stationary time series are much easier to fit for both RNNs and traditional time series modeling techniques. Afterwards, the sequence of differences D is scaled to have values in the range $[-1, 1]$. The scaled sequence of differences is denoted by D'. Scaling inputs makes the training process more stable, since backpropagation derivatives with respect to weights are proportional to the magnitude of input values. Lastly, overlapping windows of data are created from D' to obtain a final set of differenced sequences called Final.

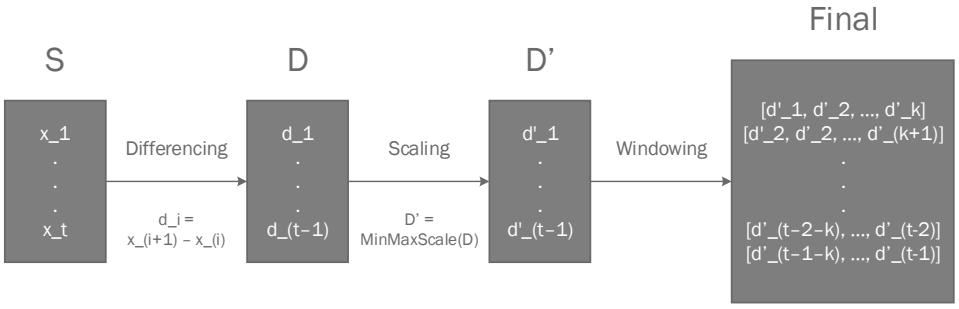


Figure 4: Preprocessing pipeline.

4.2 LSTM Setup

For the LSTM models, we tried two different training procedures: stateful and stateless. The stateful procedure gradually builds the LSTM hidden state until an entire time series dataset is processed. The stateless procedure uses windowed data and resets the LSTM hidden state after each batch is processed, thereby not taking into account relationships between batches.

4.2.1 Stateful LSTM Model

The input to the stateful model is a single difference at each time point. This is like using stochastic gradient descent since minibatches are not used. Referring back to the data processing pipeline, this corresponds to setting a window size of 1 for the windowing operation. Each difference is processed in order, and the state of the LSTM network is maintained and built upon until the very end of the entire time series dataset. This allows for modeling of long term dependencies. However, the size of the hidden state needs to be tuned in order for the hidden state not to simply get overwritten by the most recent observations. The stateful LSTM architecture can be seen in figure 5.

4.2.2 Stateless LSTM Model

The input to the stateless model is a batch of windowed data. Each window of size k in a batch is treated as being independent from the other windows in the batch and the remainder of the dataset. The independence arises from the fact that the LSTM states are reset after each sequence is processed. Thus, the weights learned by the network are more responsible for the predictions than the hidden state is since the hidden state can only represent the past k differences, unlike the stateful model. The stateless architecture can be seen in figure 6.

5 Results

In our experiments, we considered 2 time series datasets that exhibit complex temporal patterns. The first is the Mauna Loa atmospheric CO2 dataset, a canonical experimental dataset that is often used

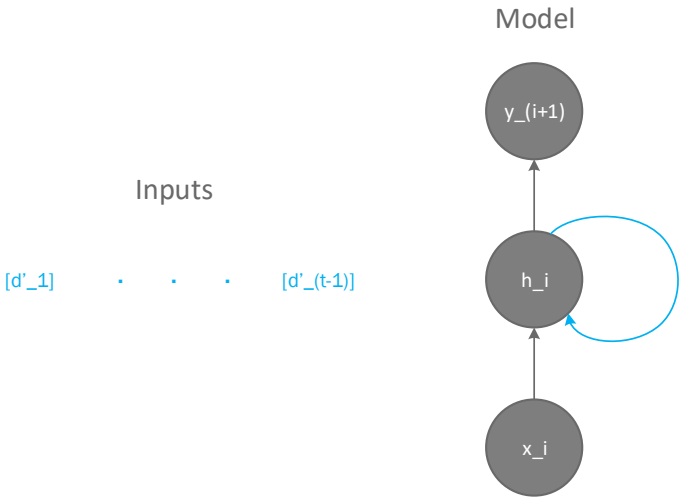


Figure 5: Stateful LSTM architecture. The input at each each iteration of the optimization procedure is a single difference d_i. The dependence between time steps is highlighted by the fact that the entire input series is coloured cyan. There is also a cyan feedback arrow in the hidden state of the model to suggest that the hidden state at any time point i depends on hidden states created by all previous inputs.

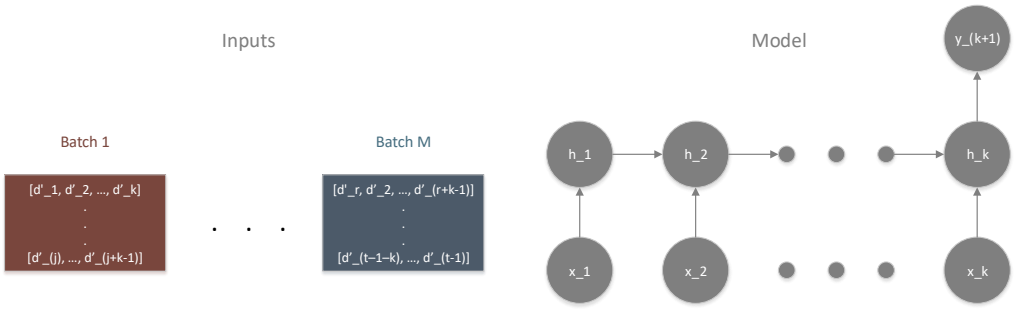
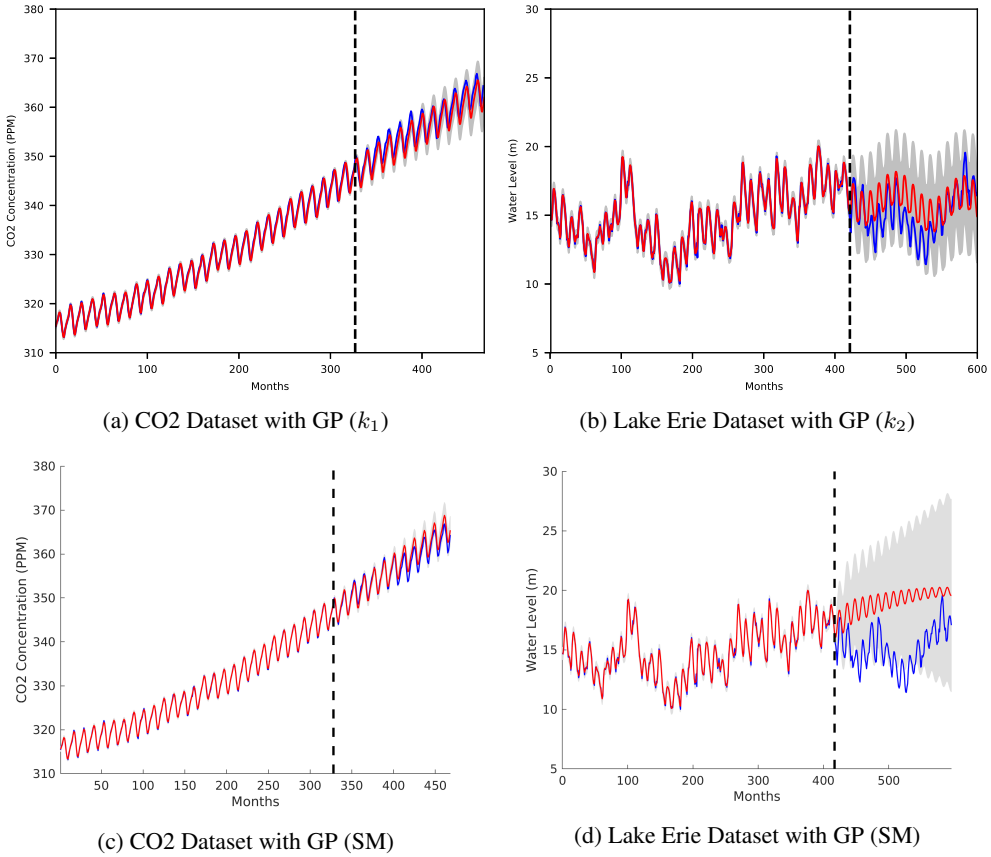


Figure 6: Stateless LSTM architecture. A batch of sequences of the past k time steps is fed as input, and the value of the $k+1$ time step is predicted in a many-to-one fashion for each sequence in the batch. The input batches have different colours to denote the fact that they are treated independently of one another since hidden states are reset after each batch. This independence is also highlighted in the model by having all arrows be of a single color that doesn't match the colors of the input batches.

in the GP literature and shows carbon dioxide concentrations reported at the Mauna Loa Observatory from 1959 to 1998. It has a non-stationary linear component as well as a seasonal component. The second is the Lake Erie dataset, which records the water level of lake Erie from 1921 to 1970. This dataset has a short-term seasonal component as well as a long term sinusoidal trend. For both datasets, all of our models were trained on the first 70% of observations and predicted the last 30%.

5.1 Experiments with GPs

In order to have a benchmark against which we can compare our results, we initiated the experimentation process by looking at the performance of GPs with various kernels on the 2 datasets. Figure 7 shows the result of applying GPs with hand-crafted kernels as well as the spectral mixture (SM) kernel [19].



(a) CO2 Dataset with GP (k_1)

(b) Lake Erie Dataset with GP (k_2)

(c) CO2 Dataset with GP (SM)

(d) Lake Erie Dataset with GP (SM)

Figure 7: Performances of GPs with custom kernels and the Spectral Mixture kernel on CO2 and Lake Erie datasets. The forecasts for both datasets are shown. Red denotes the mean of the predictions and gray represents the a confidence interval of 2 standard deviations. Actual observations are shown in blue.

For the CO2 dataset, a customized kernel k_1 from [14] was implemented. It can be expressed as:

$$k_1(x, x') = k_{RBF} + k_{RBF}k_{Per} + k_{RQ} + k_{RBF} + k_{White} \tag{1}$$

where $k_{RBF},\ k_{Per},\ k_{RQ}$ and k_{White} are the radial basis function, periodic, rational quadratic and white kernels, respectively. The GP model was able to capture both the linear trend and seasonality in the data with remarkable accuracy. The SM kernel obtained similar accuracy with slightly higher confidence (lower marginal variances). On the other hand, the GP models failed to capture the long and short term periodic trends in the Lake Erie dataset. A customized kernel k_2 was developed; it is given by:

$$k_2(x, x') = k_{RQ}(k_{Per} + k_{Per}) + k_{RBF} + k_{White} \tag{2}$$

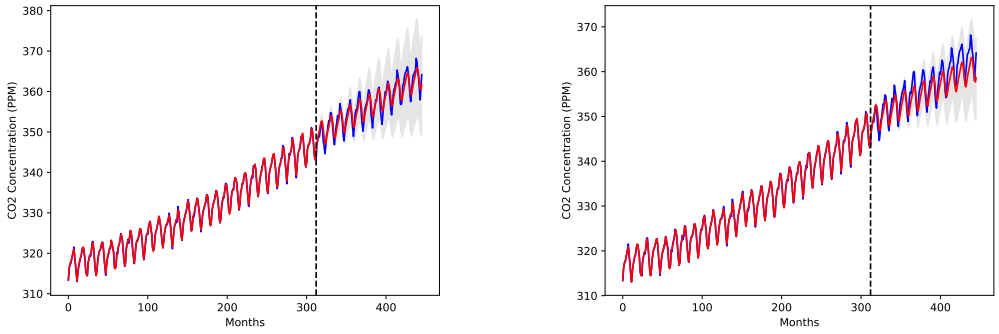
where the two periodic kernels have different hyperparameters. One of them is designed to model the long-term cycles and the other models short-term oscillations. Although the actual water level values are within the confidence bounds, the GP overestimated the values for most time points. The SM kernel produced short oscillations but failed to learn the long-term sinusoidal trend in the data and overestimated the values at all time points.

5.2 Experiments with RNNs

5.2.1 Optimization Uncertainty

Optimization uncertainty was modeled using the both stateful and stateless architectures, but no benefit was gained on the CO2 dataset using the stateful architecture, so the results in this section are only for the stateless architecture.

Figure 8 shows the results of 2 different ensembles each with 50 LSTMs in them corresponding to 2 types of random weight initializations: random normal and random uniform. The network architectures and hyperparameters are identical for the two ensembles: sequences of length 20 are used as input, a single LSTM layer of 200 hidden units is used to capture covariances in time steps, ReLU activations are used in the hidden layer, a combination of both L1 and L2 regularization both with weights 0.0001 is used to discourage overfitting, and the networks were trained for 100 epochs with a batch size of 25. These hyperparameters were optimized on a validation set (20 % of the final part of the training set) using grid search. Interestingly, there was no significant benefit gained from increasing the depth of the network by stacking multiple LSTM layers.



(a) Weights initialized with random normal distribution centered at 0 and diagonal covariance matrix with standard deviation of 0.25.

(b) Weights initialized with random uniform distribution where values range from -0.25 to 0.25.

Figure 8: Time series forecasts generated by the ensemble method for the CO2 dataset. The models in these figures were trained with different weight initializations. Red denotes the mean of the predictions and gray represents a confidence interval of 2 standard deviations. Actual observations are shown in blue.

As can be seen in figure 8, the difference in random weight initializations results in models that are diverse enough to give reasonable confidence intervals around the predicted mean. The CO2 dataset is perfect for this approach since it exhibits a clear increasing linear trend, as well as yearly periodicity. Unsurprisingly, the activation function used has a significant effect on the confidence intervals. We experimented with tanh activation as well, and it resulted in much smaller confidence intervals. A possible explanation for this is that the error function is smoother when using this activation function, thus potentially resulting in less local minima, and less of a difference between the minima.

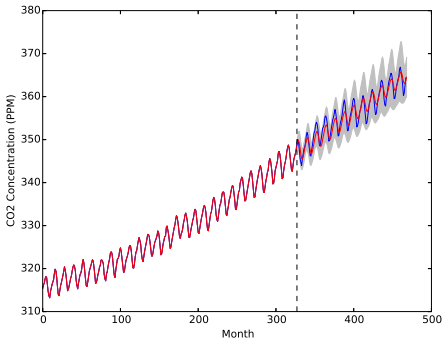
5.2.2 Data Uncertainty

The second experiment was aimed at modeling uncertainty in the time-series data. As discussed before, sub-samples of the data were generated using the GP bootstrap method, and those sub-samples were then used to train different stateful LSTM models. The models were then filtered based on their performance on the training data and the ensembles of best performing models were used to generate predictions. Figure 9(a) shows the results for the CO2 dataset and Figure 9(b) shows the results for the Lake Erie dataset. For both datasets, the LSTM models were able to capture the seasonal as well as the long-term trends, though this is less obvious in the case of Lake Erie.

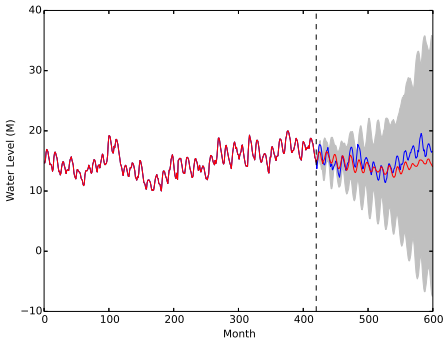
At a qualitative level, the confidence intervals for the CO2 dataset seem to display the desirable property of increasing as we get farther and farther away from the training data. The same is true for the Lake Erie dataset, though in this case the confidence intervals tend to become unreasonably large at a rate much faster than would be expected from the training data. This failure is indicative of the fact that at least 1 of the models in the ensemble failed to learn the long-term trend.

5.2.3 Architecture Uncertainty

The third experiment was aimed at capturing uncertainty in the model architecture. The figures in 10 were generated with the stateful LSTM models outlined above. The different hyper-parameters



(a) Mauna Loa CO2 Dataset with 17 LSTM Models



(b) Lake Erie Dataset with 5 LSTM Models

Figure 9: Time series forecasts generated by the ensemble method. The LSTM models in these figures were trained on different bootstrap samples of the original dataset. Red denotes the mean of the predictions and gray represents a confidence interval of 2 standard deviations. Actual observations are shown in blue.

Table 1: Ensembles for Model Uncertainty

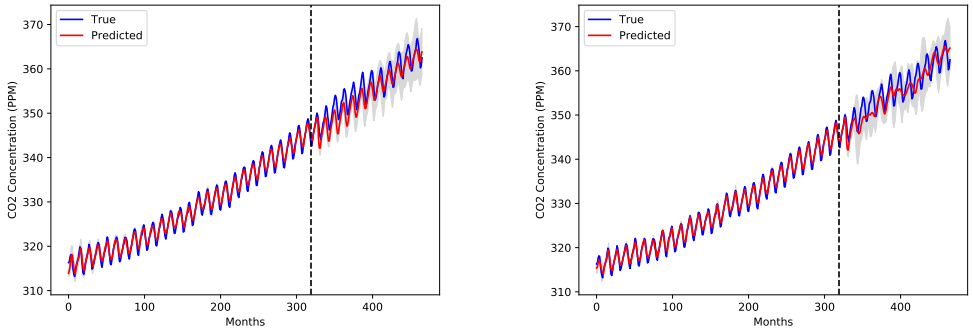
	LSTM		GRU	
	Epochs	Units	Epochs	Units
Model 1	100	40	100	50
Model 2	100	80	100	100
Model 3	100	100	200	50
Model 4	100	150	500	50
Model 5	100	200	500	100
Model 6	100	250	-	-
Model 7	200	40	-	-
Model 8	200	80	-	-
Model 9	200	100	-	-
Model 10	200	150	-	-
Model 11	200	200	-	-

considered are the number of training epochs and the number of LSTM units. The confidence intervals for the predictions steadily increase as we generate forecasts for time points that are far from the training data. The hyperparameters used for the models in the ensemble can be seen in Table 1. By comparing this against figure 8 we can see that using an ensemble of different optimal hyperparameters we can negate some of the uncertainty inherent in the architectures.

6 Discussion

In their recent paper on Gaussian Process Hybrid Deep Networks, Bradshaw et al. [1] echoed a growing concern in the deep learning community that many of the successful deep learning architectures simply "don't know what they don't know". As the authors point out, this is a serious issue in many applied domains, especially in decision-making systems. The approach advocated by Bradshaw et al. tries to tackle this problem by accounting for uncertainty using full or partial Bayesian architectures.

The ensemble approach outlined in this paper, and demonstrated in [11], is another way of tackling the problem of representing uncertainty in neural networks. In addition to its conceptual simplicity, this method also allows us to break down and model uncertainty in the 3 different dimensions from which it may arise: Uncertainty in the data, the architecture, and the optimization. And while our focus in this paper was on a specific deep learning model (RNNs) and a specific type of problem



(a) Ensemble of LSTM models

(b) Ensemble of GRU models

Figure 10: Time series forecasts generated by the ensemble method for the CO2 dataset. The models in these figures were trained with different hyperparameter settings. Red denotes the mean of the predictions and gray represents a confidence interval of 2 standard deviations. Actual observations are shown in blue.

(time series forecasting), we believe that this approach can be extended to other architectures and problem domains.

Despite these clear advantages, the ensemble approach has some theoretical as well as practical limitations. In what follows, we will discuss and address 3 of the most important issues and limitations that may arise with the ensemble approach.

6.1 Combining Uncertainties

We discussed the advantage of being able to analyze uncertainty in 3 different dimensions: Uncertainty in the data, architecture, and optimization. However, if our approach to prove useful for real-world systems, we need to be able to combine those different sources of uncertainty into one informative measure or quantity. The difficulty with producing such a measure is that it is not clear what sort of relationship holds between the different dimensions. Although we would have liked to explore this issue further, due to the limited amount of time allotted for this project we chose to leave it as an open question for future research.

6.2 Computational Cost

One of the major concerns with the approach outlined above is the fact that training an ensemble of RNNs is computationally expensive. Additionally, depending on the number of models that we choose to include in the ensemble after filtering, deploying it may prove challenging in some practical applications. In this section, we will argue that the challenge of training such an ensemble is not unique and will speculate on ways to make its deployment more feasible.

As for training, it should be noted that most machine learning practitioners employ model building, training, and criticism pipelines that make training an ensemble of models similar to the one proposed here inevitable. For example, when we perform cross-validation to determine the optimal set of hyperparameters, we do train a large number of models that are then thrown out in favor of a single, best performing model. What we propose is that instead of discarding all those models, perhaps they can be used to derive an estimate of the uncertainty in the predictions due to model architecture. A similar line of argument can be used for experimentation with weight initializations and optimization algorithms.

As for deploying the ensemble, parallel processing is a straightforward solution that we employed to ameliorate the computational bottleneck associated with generating predictions from a multitude of models. However, we acknowledge that this may not be feasible in some contexts where computational and storage resources are limited. In those domains, we speculate that a solution based on knowledge distillation and transfer learning, similar to the one proposed in [18], may provide an alternative to a large ensemble.

6.3 The Number of Bootstrap Samples: Theoretical Considerations

Perhaps a related consideration is the question of how many RNNs to include in the ensemble for the theoretical asymptotic guarantees of the bootstrap to hold. There is a debate among statisticians as to the minimum number of samples that would guarantee the correctness and exactness of the estimates derived from the bootstrap method [2]. What is clear, however, is that the more samples are used, the better the estimates will be.

Ultimately, we have to balance the considerations of computational cost discussed in the previous section on the one hand with the power loss due to the limited number of samples on the other. There are statistical methods that attempt to give a quantitative treatment to this question (see [3] for example). Such methods will prove useful in navigating this trade-off in different problem domains.

7 Conclusion

Time series data is a great fit for RNNs due to their ability to model long-term dependencies. The flexibility offered by RNNs spares data scientists from having to tinker with kernels for Gaussian processes. Although GPs outperformed RNN models on the Lake Erie dataset, this is likely due to information leak from the test set when designing the kernel. This shows the need for a data-driven time series modeling approach that is not plagued by human subjectivity, and is thus more consistent. We have added further benefits to using RNNs for time series forecasting by showing how to obtain reasonable confidence intervals for their predictions through straightforward training of multiple models. We explored three different sources of uncertainty, each of which provides insight into diverse aspects of RNN training and model selection. Such techniques can be transferred to other neural network architectures, and can help analyze the stability of regression models without using Bayesian inference.

Acknowledgments

We would like to thank professor Roger Grosse and our classmates in CSC2541 at the University of Toronto for fruitful discussions and advice.

Author Contributions

GA, SZ, TM, and AT analyzed the results and wrote the manuscript. TM performed experiments with GPs and various kernels. GA performed experiments to test optimization uncertainty with RNNs. AT performed experiments to test architecture uncertainty with RNNs. SZ performed experiments to test data uncertainty with RNNs.

References

[1] John Bradshaw, Alexander G. de G. Matthews, and Zoubin Ghahramani. Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks. pages 1–33, 2017.

[2] Michael R. Chernick. *Bootstrap methods: a guide for practitioners and researchers*. 2008.

[3] Russell Davidson and James G. MacKinnon. Bootstrap tests: how many bootstraps? *Econometric Reviews*, 19(1):55–68, 2000.

[4] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian Recurrent Neural Networks. *arXiv preprint arXiv:1704.02798v2*, pages 1–11, 2017.

[5] Yarin Gal and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In D D Lee, M Sugiyama, U V Luxburg, I Guyon, and R Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc., 2016.

[6] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015.

[7] Paul D W Kirk and Michael P H Stumpf. Gaussian process regression bootstrapping: Exploring the effects of uncertainty in time course data. *Bioinformatics*, 25(10):1300–1306, 2009.

[8] Jens-Peter Kreiss and Soumendra Nath Lahiri. Bootstrap Methods for Time Series. In *Time Series Analysis: Methods and Applications*, pages p. 3–26. 2012.

[9] Hans R. Kunsch. The Jackknife and the Bootstrap for General Stationary Observations. *The Annals of Statistics*, 17(3):1217–1241, 1989.

[10] Radford M. Neal. Bayesian Learning for Neural Networks. *Lecture Notes in Statistics*, 118:183, 1996.

[11] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4026–4034. Curran Associates, Inc., 2016.

[12] Efstathios Paparoditis and Dimitris N. Politis. The Continuous-Path Block-Bootstrap. In M. Puri, editor, *Asymptotics in Statistics and Probability*, pages 305–320. VSP Publications, 2001.

[13] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

[14] Carl Edward Rasmussen and Christopher K I Williams. *Gaussian processes for machine learning.*, volume 14. 2006.

[15] Sebastian Ruder. An overview of gradient descent optimization algorithms. pages 1–14, 2016.

[16] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.

[17] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2171–2180, Lille, France, 07–09 Jul 2015. PMLR.

[18] Zhiyuan Tang, Dong Wang, and Zhiyong Zhang. Recurrent neural network training with dark knowledge transfer. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904, 2016.

[19] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian Process Covariance Kernels for Pattern Discovery and Extrapolation. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, page 15, 2013.