2017 IEEE International Conference on Data Mining Workshops

Deep and Confident Prediction for Time Series at Uber

Lingxue Zhu*
Department of Statistics,
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: lzhu@cmu.edu

Nikolay Laptev
Uber Technologies
San Francisco, California 94103
Email: nlaptev@uber.com

***Abstract*—Reliable uncertainty estimation for time series prediction is critical in many fields, including physics, biology, and manufacturing. At Uber, probabilistic time series forecasting is used for robust prediction of number of trips during special events, driver incentive allocation, as well as real-time anomaly detection across millions of metrics. Classical time series models are often used in conjunction with a probabilistic formulation for uncertainty estimation. However, such models are hard to tune, scale, and add exogenous variables to. Motivated by the recent resurgence of Long Short Term Memory networks, we propose a novel end-to-end Bayesian deep model that provides time series prediction along with uncertainty estimation. We provide detailed experiments of the proposed solution on completed trips data, and successfully apply it to large-scale time series anomaly detection at Uber.**

***Keywords*—Bayesian neural networks, predictive uncertainty, time series, anomaly detection.**

1. Introduction

Accurate time series forecasting and reliable estimation of the prediction uncertainty are critical for anomaly detection, optimal resource allocation, budget planning, and other related tasks. This problem is challenging, especially during high variance segments (e.g., holidays, sporting events), because extreme event prediction depends on numerous external factors that can include weather, city population growth, or marketing changes (e.g., driver incentives) [1] that all contribute to the uncertainty of the forecast. These exogenous variables, however, are difficult to incorporate in many classical time series models, such as those found in the standard *R forecast* [2] package. In addition, these models usually require manual tuning to set model and uncertainty parameters.

Relatively recently, time series modeling based on the Long Short Term Memory (LSTM) model [3] has gained popularity due to its end-to-end modeling, ease of incorporating exogenous variables, and automatic feature extraction abilities [4]. By providing a large amount of data across numerous dimensions, it has been shown that an LSTM network can model complex nonlinear feature interactions [5], which is critical for modeling complex extreme events. A recent paper [6] has shown that a neural network forecasting model is able to outperform classical time series methods in cases with long, interdependent time series.

However, the problem of estimating the uncertainty in time-series predictions using neural networks remains an open question. The prediction uncertainty is important for assessing how much to trust the forecast produced by the model, and has profound impact in anomaly detection. The previous model proposed in [6] had no information regarding the uncertainty. Specifically, this resulted in a large false anomaly rates during holidays where the model prediction has large variance.

In this paper, we propose a novel end-to-end model architecture for time series prediction, and quantify the prediction uncertainty using Bayesian Neural Network, which is further used for large-scale anomaly detection.

Recently, Bayesian neural networks (BNNs) have garnered increasing attention as a principled framework to provide uncertainty estimation for deep models. Under this framework, the prediction uncertainty can be decomposed into three types: *model uncertainty*, *inherent noise*, and *model misspecification*. Model uncertainty, also referred to as epistemic uncertainty, captures our ignorance of the model parameters, and can be reduced as more samples being collected. Inherent noise, on the other hand, captures the uncertainty in the data generation process and is irreducible. These two sources have been previously recognized with successful application in computer visions [7].

The third uncertainty from model misspecification, however, has been long-overlooked. This captures the scenario where the testing samples come from a different population than the training set, which is often the case in time series anomaly detection. Similar ideas have gained attention in deep learning under the concept of adversarial examples in computer vision [8], but its implication in prediction uncertainty remains unexplored. Here, we propose a principled solution to incorporate this uncertainty using an encoder-decoder framework. To the best of our knowledge, this is the first time that misspecification uncertainty has been successfully applied to prediction and anomaly detection in

* *This work was done during an internship at Uber Technologies.*

a principled way.

In summary, this paper makes the following contributions:

- Provides a generic and scalable uncertainty estimation implementation for deep prediction models.
- Quantifies the prediction uncertainty from three sources: (i) model uncertainty, (ii) inherent noise, and (iii) model misspecification. The third uncertainty has been previously overlooked, and we propose a potential solution with an encoder-decoder.
- Motivates a real-world anomaly detection use-case at Uber that uses Bayesian Neural Networks with uncertainty estimation to improve performance at scale.

The rest of this paper is organized as follows: Section 2 gives an overview of previous work on time series prediction for both classical and deep learning models, as well as the various approaches for uncertainty estimation in neural networks. The approach of Monte Carlo dropout (MC dropout) is used in this paper due to its simplicity, strong generalization ability, and scalability. In Section 3, we present our uncertainty estimation algorithm that accounts for the three different sources of uncertainty. Section 4 provides detailed experiments to evaluate the model performance on Uber trip data, and lays out a successful application to large-scale anomaly detection for millions of metrics at Uber. Finally, Section 5 concludes the paper.

2. Related Works

2.1. Time Series Prediction

Classical time series models, such as those found in the standard *R forecast* [2] package are popular methods to provide an univariate base-level forecast. These models usually require manual tuning to set seasonality and other parameters. Furthermore, while there are time series models that can incorporate exogenous variables [9], they suffer from the curse of dimensionality and require frequent retraining. To more effectively deal with exogenous variables, a combination of univariate modeling and a machine learning model to handle residuals was introduced in [10]. The resulting two-stage model, however, is hard to tune, requires manual feature extraction and frequent retraining, which is prohibitive to millions of time series.

Relatively recently, time series modeling based on LSTM [3] technique gained popularity due to its end-to-end modeling, ease of incorporating exogenous variables, and automatic feature extraction abilities [4]. By providing a large amount of data across numerous dimensions, it has been shown that an LSTM approach can model complex extreme events by allowing nonlinear feature interactions [5], [6].

While uncertainty estimation for classical forecasting models has been widely studied [11], this is not the case for neural networks. Approaches such as a modified loss function or using a collection of heterogenous networks [12] were proposed, however they require changes to the underlying model architecture. A more detailed review is given in the next section.

In this work, we use a simple and scalable approach for deep model uncertainty estimation that builds on [13]. This framework provides a generic error estimator that runs in production at Uber-scale to mitigate against bad decisions (e.g., false anomaly alerts) resulting from poor forecasts due to high prediction variance.

2.2. Bayesian Neural Networks

Bayesian Neural Networks (BNNs) introduce uncertainty to deep learning models from a Bayesian perspective. By giving a prior to the network parameters W, the network aims to find the *posterior distribution* of W, instead of a point estimation.

This procedure is usually referred to as posterior inference in traditional Bayesian models. Unfortunately, due to the complicated non-linearity and non-conjugacy in deep models, exact posterior inference is rarely available; in addition, most traditional algorithms for approximate Bayesian inference cannot scale to the large number of parameters in most neural networks.

Recently, several approximate inference methods are proposed for Bayesian Neural Networks. Most approaches are based on variational inference that optimizes the variational lower bound, including stochastic search [14], variational Bayes [15], probabilistic backpropagation [16], Bayes by BackProp [17] and its extension [18]. Several algorithms further extend the approximation framework to α-divergence optimization, including [19], [20]. We refer the readers to [21] for a more detailed and complete review of these methods.

All of the aforementioned algorithms require different training methods for the neural network. Specifically, the loss function must be adjusted to different optimization problems, and the training algorithm has to be modified in a usually non-trivial sense. In practice, however, an out-of-the-box solution is often preferred, without changing the neural network architecture and can be directly applied to the previously trained model. In addition, most existing inference algorithms introduce extra model parameters, sometimes even double, which is difficult to scale given the large amount of parameters used in practice.

This paper is inspired by the Monte Carlo dropout (MC dropout) framework proposed in [13] and [22], which requires no change of the existing model architecture and provides uncertainty estimation almost for free. Specifically, stochastic dropouts are applied after each hidden layer, and the model output can be approximately viewed as a random sample generated from the posterior predictive distribution [21]. As a result, the model uncertainty can be estimated by the sample variance of the model predictions in a few repetitions. Details of this algorithm will be reviewed in the next section.

The MC dropout framework is particularly appealing to practitioners because it is generic, easy to implement, and directly applicable to any existing neural networks. However, the exploration of its application to real-world problems remains extremely limited. This paper takes an important step forward by successfully adapting this framework to conduct time series prediction and anomaly detection at large scale.

3. Method

Given a trained neural network $f^{\hat{W}}(\cdot)$ where $\hat{W}$ represents the fitted parameters, as well as a new sample x^*, our goal is to evaluate the uncertainty of the model prediction, $\hat{y}^* = f^{\hat{W}}(x^*)$. Specifically, we would like to quantify the prediction standard error, η, so that an approximate α-level prediction interval can be constructed by

$$[\hat{y}^* - z_{\alpha/2}\eta,\ \hat{y}^* + z_{\alpha/2}\eta] \tag{1}$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of a standard Normal. This prediction interval is critical for various tasks. For example, in anomaly detection, anomaly alerts will be fired when the observed value falls outside the constructed 95% interval. As a result, underestimating η will lead to high false positive rates.

In the rest of this section, we will present our uncertainty estimation algorithm in Section 3.1, which accounts for three different sources of prediction uncertainties. This framework can be generalized to any neural network architectures. Then, in Section 3.2, we will present our neural network design for predicting time series at Uber.

3.1. Prediction Uncertainty

We denote a neural network as function $f^W(\cdot)$, where f captures the network architecture, and W is the collection of model parameters. In a Bayesian neural network, a prior is introduced for the weight parameters, and the model aims to fit the optimal posterior distribution. For example, a Gaussian prior is commonly assumed:

$$W \sim N(0, I)$$

We further specify the data generating distribution $p(y \mid f^W(x))$. In regression, we often assume

$$y \mid W \sim N(f^W(x), \sigma^2)$$

with some noise level σ. In classification, the softmax likelihood is often used. For time series prediction, we will focus on the regression setting in this paper.

Given a set of N observations $X = \{x_1, ..., x_N\}$ and $Y = \{y_1, ..., y_N\}$, Bayesian inference aims at finding the posterior distribution over model parameters $p(W \mid X, Y)$. With a new data point x^*, the prediction distribution is obtained by marginalizing out the posterior distribution:

$$p(y^* \mid x^*) = \int_W p(y^* \mid f^W(x^*)) p(W \mid X, Y)\, dW$$

In particular, the variance of the prediction distribution quantifies the prediction uncertainty, which can be further decomposed using law of total variance:

$$\begin{aligned} \text{Var}(y^* \mid x^*) &= \text{Var}\left[\mathbb{E}(y^* \mid W, x^*)\right] + \mathbb{E}\left[\text{Var}(y^* \mid W, x^*)\right] \\ &= \text{Var}(f^W(x^*)) + \sigma^2 \end{aligned} \tag{2}$$

Immediately, we see that the variance is decomposed into two terms: (i) $\text{Var}(f^W(x^*))$, which reflects our ignorance over model parameter W, referred to as the *model uncertainty*; and (ii) σ^2 which is the noise level during data generating process, referred to as the *inherent noise*.

An underlying assumption for (2) is that y^* is generated by the same procedure. However, this is not always the case in practice. In anomaly detection, in particular, it is expected that certain time series will have unusual patterns, which can be very different from the trained model. Therefore, we propose that a complete measurement of prediction uncertainty should be a combination from three sources: (i) model uncertainty, (ii) model misspecification, and (iii) inherent noise level. The following sections provide details on how we handle these three terms.

3.1.1. Model uncertainty. The key to estimating model uncertainty is the posterior distribution $p(W \mid X, Y)$, also referred to as Bayesian inference. This is particularly challenging in neural networks because the non-conjugacy due to nonlinearities. There have been various research efforts on approximate inference in deep learning (see Section 2.2 for a review). Here, we follow the idea in [13] and [22] to approximate model uncertainty using Monte Carlo dropout (MC dropout).

The algorithm proceeds as follows: given a new input x^*, we compute the neural network output with stochastic dropouts at each layer. That is, randomly dropout each hidden unit with certain probability p. This stochastic feedforward is repeated B times, and we obtain $\{\hat{y}^*_{(1)}, ..., \hat{y}^*_{(B)}\}$. Then the model uncertainty can be approximated by the sample variance:

$$\widehat{\text{Var}}(f^W(x^*)) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{y}^*_{(b)} - \overline{\hat{y}}^*\right)^2 \tag{3}$$

where $\overline{\hat{y}}^* = \frac{1}{B}\sum_{b=1}^{B}\hat{y}^*_{(b)}$ [13]. There has been recent work done on choosing the optimal dropout probability p adaptively by treating it as part of the model parameter, but this approach requires modifying the training phase [12]. In practice, we find that the uncertainty estimation is usually robust within a reasonable range of p.

3.1.2. Model misspecification. Next, we address the problem of capturing potential model misspecification. In particular, we would like to capture the uncertainty when predicting unseen samples with very different patterns from the training data set. We propose to account for this source of uncertainty by introducing an encoder-decoder to the model framework. The idea is to train an encoder that extracts the representative features from a time series, in the sense that a decoder can reconstruct the time series from the encoded space. At test time, the quality of encoding of each sample will provide insight on how close it is to the training set. Another way to think of this approach is that we first fit a latent embedding space for all training time series using an encoder-decoder framework. Then, we measure the distance between test cases and training samples in the embedded space.

The next question is how to incorporate this uncertainty in the variance calculation. Here, we take a principled approach by connecting the encoder, $g(\cdot)$, with a prediction network, $h(\cdot)$, and treat them as one large network $f = h(g(\cdot))$ during inference. Figure 1 illustrates such an inference network, and Algorithm 1 presents the MC dropout algorithm. Specifically, given an input time series $x = \{x_1, ..., x_T\}$, the encoder $g(\cdot)$ constructs the learned embedding $e = g(x)$, which is further concatenated with external features, and the final vector is fed to the final prediction network h. During this feedforward pass, MC dropout is applied to all layers in both the encoder g and the prediction network h. As a result, the random dropout in the encoder perturbs the input intelligently in the embedding space, which accounts for potential model misspecification and gets further propagated through the prediction network. Here, variational dropout for recurrent neural networks [22] is applied to the LSTM layers in the encoder, and regular dropout [13] is applied to the prediction network.

Algorithm 1: MCdropout

```
Input: data $x^*$, encoder $g(\cdot)$, prediction network $h(\cdot)$,
    dropout probability $p$, number of iterations $B$
Output: prediction $\hat{y}^*_{mc}$, uncertainty $\eta_1$
1: for $b = 1$ to $B$ do
2:     $e^*_{(b)} \leftarrow$ VariationalDropout$(g(x^*), p)$
3:     $z^*_{(b)} \leftarrow$ Concatenate$(e^*_{(b)}, \text{extFeatures})$
4:     $\hat{y}^*_{(b)} \leftarrow$ Dropout $(h(z^*_{(b)}), p)$
5: end for
   // prediction
6: $\hat{y}^*_{mc} \leftarrow \frac{1}{B}\sum_{b=1}^{B} \hat{y}^*_{(b)}$
   // model uncertainty and misspecification
7: $\eta_1^2 \leftarrow \frac{1}{B}\sum_{b=1}^{B}(\hat{y}^*_{(b)} - \hat{y}^*)^2$
8: return $\hat{y}^*_{mc}$, $\eta_1$
```

3.1.3. Inherent noise. Finally, we estimate the inherent noise level σ^2. In the original MC dropout algorithm [13], this parameter is implicitly determined by a prior over the smoothness of W. As a result, the model could end up with drastically different estimations of the uncertainty level depending on this pre-specified smoothness (see [21], chapter 4). This dependency is undesirable in anomaly detection, because we want the uncertainty estimation to also have robust frequentist coverage, but it is rarely the case that we would know the correct noise level *a priori.*

Here, we propose a simple and adaptive approach that estimates the noise level via the residual sum of squares, evaluated on an independent held-out validation set. Specifically, let $f^{\hat{W}}(\cdot)$ be the fitted model on training data, and $X' = \{x'_1, ..., x'_V\}, Y' = \{y'_1, ..., y'_V\}$ be an independent validation set, then we estimate σ^2 via

$$\hat{\sigma}^2 = \frac{1}{V}\sum_{v=1}^{V}\left(y'_v - f^{\hat{W}}(x'_v)\right)^2 . \tag{4}$$

Note that (X', Y') are independent from $f^{\hat{W}}(\cdot)$, and if we further assume that $f^{\hat{W}}(x'_v)$ is an unbiased estimation of the true model, we have

$$\begin{aligned}\mathbb{E}(\hat{\sigma}^2) &= \sigma^2 + \frac{1}{V}\sum_{v=1}^{V}\mathbb{E}\left[f^{\hat{W}}(x'_v) - f^{W}(x'_v)\right]^2 \\ &= \sigma^2 + \text{Var}_{\text{TRN}}(f^{\hat{W}}(x'_v))\end{aligned} \tag{5}$$

where Var_{TRN} is w.r.t the training data, which decreases as the training sample size increases, and $\to 0$ as the training sample size $N \to \infty$. Therefore, $\hat{\sigma}^2$ provides an asymptotically unbiased estimation on the inherent noise level. In the finite sample scenario, it always overestimates the noise level and tends to be more conservative.

The final inference algorithm combines inherent noise estimation with MC dropout, and is presented in Algorithm 2.

Algorithm 2: Inference

```
Input: data $x^*$, encoder $g(\cdot)$, prediction network $h(\cdot)$,
    dropout probability $p$, number of iterations $B$
Output: prediction $\hat{y}^*$, predictive uncertainty $\eta$
   // prediction, model uncertainty and misspecification
1: $\hat{y}^*$, $\eta_1 \leftarrow$ MCdropout $(x^*, g, h, p, B)$
   // Inherent noise
2: for $x'_v$ in validation set $\{x'_1, ..., x'_V\}$ do
3:     $\hat{y'}_v \leftarrow h(g(x'_v))$
4: end for
5: $\eta_2^2 \leftarrow \frac{1}{V}\sum_{v=1}^{V}\left(\hat{y'}_v - y'_v\right)^2$
   // total prediction uncertainty
6: $\eta \leftarrow \sqrt{\eta_1^2 + \eta_2^2}$
7: return $\hat{y}^*$, $\eta$
```

3.2. Model Design

The complete architecture of the neural network is shown in Figure 1. The network contains two major components: (i) an encoder-decoder framework that captures the inherent pattern in the time series, which is learned during pre-training step, and (ii) a prediction network that takes input from both the learned embedding from encoder-decoder, as well as any potential external features to guide the prediction. We discuss the two components in more details below.

3.2.1. Encoder-decoder. Prior to fitting the prediction model, we first conduct a pre-training step to fit an encoder that can extract useful and representative embeddings from a time series. The goals are to ensure that (i) the learned

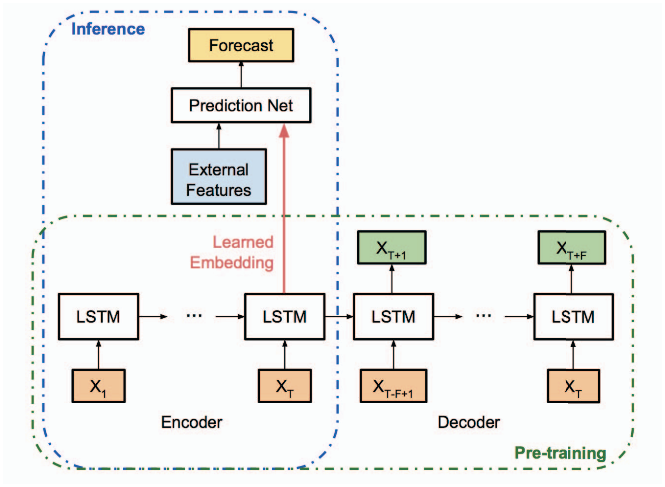


Figure 1. Neural network architecture, with a pre-training phase using a LSTM encoder-decoder, followed by a prediction network, with input being the learned embedding concatenated with external features.

embedding provides useful features for prediction and (ii) unusual input can be captured in the embedded space, which will get further propagated to the prediction network in the next step. Here, we use an encoder-decoder framework with two-layer LSTM cells.

Specifically, given a univariate time series $\{x_t\}_t$, the encoder reads in the first T timestamps $\{x_1, ..., x_T\}$, and constructs a fixed-dimensional embedding state. After then, from this embedding state, the decoder constructs the following F timestamps $\{x_{T+1}, ..., x_{T+F}\}$ with guidance from $\{x_{T-F+1}, ..., x_T\}$ (Figure 1, bottom panel). The intuition is that in order to construct the next few timestamps, the embedding state must extract representative and meaningful features from the input time series. This design is inspired from the success of video representation learning using a similar architecture [23].

3.2.2. Prediction network. After the encoder-decoder is pre-trained, it is treated as an intelligent feature-extraction blackbox. Specifically, the last LSTM cell states of the encoder are extracted as learned embedding. Then, a prediction network is trained to forecast the next one or more timestamps using the learned embedding as features. In the scenario where external features are available, these can be concatenated to the embedding vector and passed together to the final prediction network.

Here, we use a multi-layer perceptron as the prediction network. We will show in Section 4.1 that the learned embedding from the encoder successfully captures interesting patterns from the input time series. In addition, including external features significantly improves the prediction accuracy during holidays and special events (see Section 4)

3.2.3. Inference. After the full model is trained, the inference stage involves only the encoder and the prediction network (Figure 1, left panel). The complete inference algorithm is presented in Algorithm 2, where the prediction uncertainty, η, contains two terms: (i) the model and misspecification uncertainty, estimated by applying MC dropout to both the encoder and the prediction network, as presented in Algorithm 1; and (ii) the inherent noise level, estimated by the residuals on a held-out validation set. Finally, an approximate α-level prediction interval is constructed by $[\hat{y}^* - z_{\alpha/2}\eta,\ \hat{y}^* + z_{\alpha/2}\eta]$, where $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of a standard Normal.

Two hyper-parameters need to be specified in Algorithm 2: the dropout probability, p, and the number of iterations, B. As for the dropout probability, we find in our experiments that the uncertainty estimation is relatively stable across a range of p, and we choose the one that achieves the best performance on the validation set. As for the number of iterations, the standard error of the estimated prediction uncertainty is proportional to $1/\sqrt{B}$. We measure the standard error across different repetitions, and find that a few hundreds of iterations are usually suffice to achieve a stable estimation.

4. Evaluation

This section contains two sets of results. We first evaluate the model performance on a moderately sized data set of daily trips processed by the Uber platform. We will evaluate the prediction accuracy and the quality of uncertain estimation during both holidays and non-holidays. We will also present how the encoder recognizes the day of the week pattern in the embedding space. Next, we will illustrate the application of this model to real-time large-scale anomaly detection for millions of metrics at Uber.

4.1. Results on Uber Trip Data

4.1.1. Experimental settings. In this section, we illustrate the model performance using the daily completed trips over four years across eight representative large cities in U.S. and Canada, including Atlanta, Boston, Chicago, Los Angeles, New York City, San Francisco, Toronto, and Washington D.C. We use three years of data as the training set, the following four months as the validation set, and the final eight months as the testing set. The encoder-decoder is constructed with two-layer LSTM cells, with 128 and 32 hidden states, respectively. The prediction network has three fully connected layers with *tanh* activation, with 128, 64, and 16 hidden units, respectively.

Samples are constructed using a sliding window with step size one, where each sliding window contains the previous 28 days as input, and aims to forecast the upcoming day. The raw data are log-transformed to alleviate exponential effects. Next, within each sliding window, the first day is subtracted from all values, so that trends are removed and the neural network is trained for the incremental value. At test time, it is straightforward to revert these transformations to obtain predictions at the original scale.

4.1.2. Prediction performance. We compare the prediction accuracy among four different models:

TABLE 1. SMAPE of Four Different Prediction Models, Evaluated on the Test Data.

City	Last-Day	QRF	LSTM	Our Model
Atlanta	15.9	13.2	11.0	7.3
Boston	13.6	15.4	10.0	8.2
Chicago	16.0	12.7	9.5	6.1
Los Angeles	12.3	10.9	8.5	4.7
New York City	11.5	10.9	8.7	6.1
San Francisco	10.7	11.8	7.3	4.5
Toronto	15.2	11.7	10.0	5.3
Washington D.C.	13.0	13.3	8.2	5.2
Average	**13.5**	**12.5**	**9.2**	**5.9**

1) **Last-Day**: A naive model that uses the last day's completed trips as the prediction for the next day.
2) **QRF**: Based on the naive last-day prediction, a quantile random forest (QRF) is further trained to estimate the holiday lifts, i.e., the ratio to adjust the forecast during holidays. The final prediction is calculated from the last-day forecast multiplied by the estimated ratio.
3) **LSTM**: A vanilla LSTM model with similar size as our model. Specifically, a two-layer sacked LSTM is constructed, with 128 and 32 hidden states, respectively, followed by a fully connected layer for the final output. This neural network also takes 28 days as input, and predicts the next day.
4) **Our Model**: Our model that combines an encoder-decoder and a prediction network, as described in Figure 1.

Table 1 reports the Symmetric Mean Absolute Percentage Error (SMAPE) of the four models, evaluated on the testing set. We see that using a QRF to adjust for holiday lifts is only slightly better than the naive prediction. On the other hand, a vanilla LSTM neural network provides an average of 26% improvement across the eight cities. As we further incorporate the encoder-decoder framework and introduce external features for holidays to the prediction network (Figure 1), our proposed model achieves another 36% improvement in prediction accuracy. Note that when using LSTM and our model, only one generic model is trained, where the neural network is not tuned for any city-specific patterns; nevertheless, we still observe significant improvement on SMAPE across all cities when compared to traditional approaches.

Finally, Figure 2 visualizes the true values and our predictions during the testing period in San Francisco as an example. We observe that accurate predictions are achieved not only in regular days, but also during holiday seasons.

4.1.3. Uncertainty estimation. Next, we evaluate the quality of our uncertainty estimation by calibrating the empirical coverage of the prediction intervals. Here, the dropout probability is set to be 5% at each layer, and Table 2 reports the empirical coverage of the 95% predictive intervals under three different scenarios:

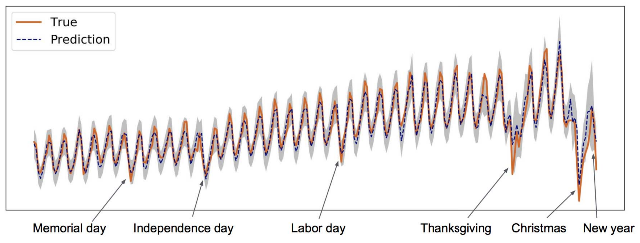


Figure 2. Daily completed trips in San Francisco during eight months of the testing set. True values are shown with the orange solid line, and predictions are shown with the blue dashed line, where the 95% prediction band is shown as the grey area. Exact values are anonymized.

TABLE 2. Empirical Coverage of 95% Predictive Intervals, Evaluated on the Test Data.

City	PredNet	Enc+Pred	Enc+Pred+Noise
Atlanta	78.33%	91.25%	94.30%
Boston	85.93%	95.82%	99.24%
Chicago	71.86%	80.23%	90.49%
Los Angeles	76.43%	92.40%	94.30%
New York City	76.43%	85.55%	95.44%
San Francisco	78.33%	95.06%	96.20%
Toronto	80.23%	90.87%	94.68%
Washington D.C.	78.33%	93.54%	96.96%
Average	**78.23%**	**90.59%**	**95.20%**

1) **PredNet**: Use only model uncertainty estimated from MC dropout in the prediction network, with no dropout layers in the encoder.
2) **Enc+Pred**: Use MC dropout in both the encoder and the prediction network, but without the inherent noise level. This is the term η_1 in Algorithm 2.
3) **Enc+Pred+Noise**: Use the full prediction uncertainty η as presented in Algorithm 2, including η_1 as in 2), as well as the inherent noise level η_2.

By comparing `PredNet` with `Enc+Pred`, it is clear that introducing MC dropout to the encoder network is critical, which significantly improves the empirical coverage from 78% to 90% by capturing potential model misspecification. In addition, by further accounting for the inherent noise level, the empirical coverage of the final uncertainty estimation, `Enc+Pred+Noise`, nicely centers around 95% as desired.

One important use-case of the uncertainty estimation is to provide insight for unusual patterns in the time series. Figure 3 shows the estimated predictive uncertainty on six U.S. holidays in the testing data. We see that New Year's Eve has significantly higher uncertainty than all other holidays. This pattern is consistent with our previous experience, where New Year's Eve is usually the most difficult day to predict.

4.1.4. Embedding features. As illustrated previously, the encoder is critical for both improving prediction accuracy,

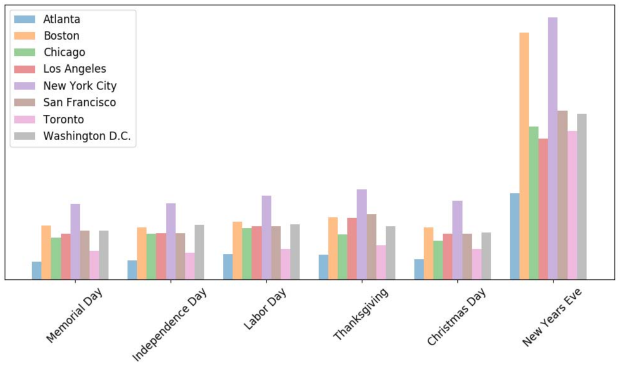


Figure 3. Estimated prediction standard deviations on six U.S. holidays during testing period for eight cities. Exact values are anonymized.

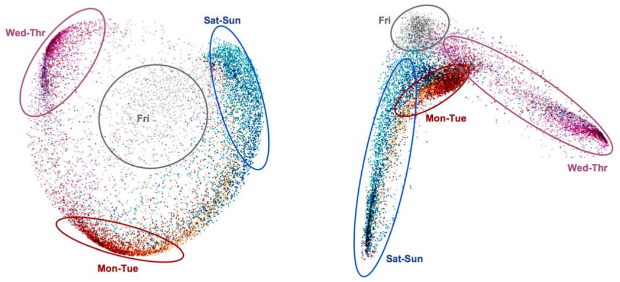


Figure 4. Training set of time series, visualized in the embedding space. Each point represents a 28-day segment, colored by the day of the week of the last day. We evaluate the cell states of the two LSTM layers, where the first layer with dimension 128 is plotted on the left, and second layer with dimension 32 is plotted on the right. PCA is used to project into 2D space for visualization.

as well as for estimating prediction uncertainty. One natural follow-up question is whether we can interpret the embedding features extracted by the encoder. This can also provide valuable insights for model selection and anomaly detection. Here, we visualize our training data, each being a 28-day time series segment, in the embedding space. We use the last LSTM cell in the encoder, and project its cell states to 2D for visualization using PCA (Figure 4). The strongest pattern we observe is day of the week, where weekdays and weekends form different clusters, with Fridays usually sitting in between. We do not observe city-level clusters, which is probably due to the fact all cities in this data set are large cities in North America, where riders and drivers tend to have similar behaviors.

4.2. Application to Anomaly Detection at Uber

At Uber, we track millions of metrics each day to monitor the status of various services across the company. One important application of uncertainty estimation is to provide real-time anomaly detection and deploy alerts for potential outages and unusual behaviors. A natural approach is to trigger an alarm when the observed value falls outside of the 95% predictive interval. There are two main challenges we need to address in this application:

- Scalability: In order to provide real-time anomaly detection at the current scale, each predictive interval must be calculated within a few milliseconds during inference stage.
- Performance: With highly imbalanced data, we aim to reduce the false positive rate as much as possible to avoid unnecessary on-call duties, while making sure the false negative rate is properly controlled so that real outages will be captured.

4.2.1. Scalability. Our model inference is implemented in Go. Our implementation involves efficient matrix manipulation operations, as well as stochastic dropout by randomly setting hidden units to zero with pre-specified probability. A few hundred stochastic passes are executed to calculate the prediction uncertainty, which is updated every few minutes for each metric. We find that the uncertainty estimation step adds only a small amount of computation overhead and can be conducted within ten milliseconds per metric.

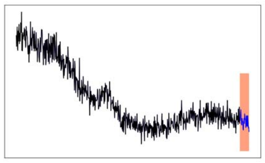
4.2.2. Performance. Here, we illustrate the precision and recall of this framework on an example data set containing 100 metrics with manual annotation available, where 17 of them are true anomalies. Note that the neural network was previously trained on a separate and much larger data set. By adding MC dropout layers in the neural network, the estimated predictive intervals achieved 100% recall rate and a 80.95% precision rate. Figure 5 visualizes the neural network predictive intervals on four representative metrics, where alerts are correctly fired for two of them. When applying this framework to all metrics, we observe a 4% improvement in precision compared to the previous ad-hoc solution, which is substantial at Uber's scale.

5. Conclusion

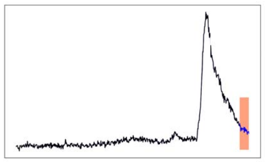
We have presented an end-to-end neural network architecture for uncertainty estimation used at Uber. Using the MC dropout technique and model misspecification distribution, we showed a simple way to provide uncertainty estimation for a neural network forecast at scale while providing a 95% uncertainty coverage. A critical feature about our framework is its applicability to any neural network without modifying the underlying architecture.

We have used the proposed uncertainty estimate to measure special event (e.g., holiday) uncertainty and to improve anomaly detection accuracy. For special event uncertainty estimation, we found New Year's Eve to be the most uncertain time. Using the uncertainty information, we adjusted the confidence bands of an internal anomaly detection model to improve precision during high uncertainty events, resulting in a 4% accuracy improvement, which is large given the number of metrics we track at Uber.

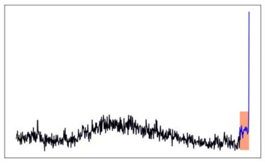
Our future work will be focused on utilizing the uncertainty information for neural network debugging during high error periods.



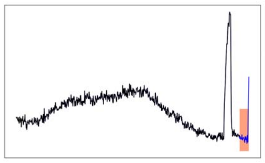
(a) Normal I



(b) Normal II



(c) Anomaly I



(d) Anomaly II

Figure 5. Four example metrics during a 12-hour span, and anomaly detection is performed for the following 30 minutes. All metrics are evaluated by minutes. The neural network constructs predictive intervals for the following 30 minutes, visualized by the shaded area in each plot. **(a)** A normal metric with large fluctuation, where the observation falls within the predictive interval. **(b)** A normal metric with small fluctuation, and an unusual inflation has just ended. The predictive interval still captures the observation. **(c)** An anomalous metric with a single spike that falls outside the predictive interval. **(d)** An anomalous metric with two consecutive spikes, also captured by our model.

References

[1] J. D. Horne and W. Manzenreiter, "Accounting for mega-events," *International Review for the Sociology of Sport*, vol. 39, no. 2, pp. 187–203, 2004.

[2] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: the forecast package for R," *Journal of Statistical Software*, vol. 26, no. 3, pp. 1–22, 2008.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] M. Assaad, R. Boné, and H. Cardot, "A new boosting algorithm for improved time-series forecasting with recurrent neural networks," *Information Fusion*, vol. 9, no. 1, pp. 41–55, 2008.

[5] O. P. Ogunmolu, X. Gu, S. B. Jiang, and N. R. Gans, "Nonlinear systems identification using deep dynamic neural networks," *arXiv preprint arXiv:1610.01439*, 2016.

[6] N. Laptev, J. Yosinski, E. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," *International Conference on Machine Learning*, 2017.

[7] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" *arXiv preprint arXiv:1703.04977*, 2017.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *International Conference on Learning Representations*, 2014.

[9] W. W.-S. Wei, *Time series analysis*. Addison-Wesley publ Reading, 1994.

[10] T. Opitz, "Modeling asymptotically independent spatial extremes based on laplace random fields," *Spatial Statistics*, vol. 16, pp. 1–18, 2016.

[11] R. D. Dietz, T. L. Casavant, T. E. Scheetz, T. A. Braun, and M. S. Andersland, "Modeling the impact of run-time uncertainty on optimal computation scheduling using feedback," in *Parallel Processing '97*, Aug, pp. 481–488.

[12] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," *arXiv preprint arXiv:1705.07832*, 2017.

[13] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 1050–1059.

[14] J. Paisley, D. Blei, and M. Jordan, "Variational bayesian inference with stochastic search," in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 1367–1374.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *The International Conference on Learning Representations*, 2014.

[16] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 1861–1869.

[17] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 1613–1622.

[18] M. Fortunato, C. Blundell, and O. Vinyals, "Bayesian recurrent neural networks," *arXiv preprint arXiv:1704.02798*, 2017.

[19] J. M. Hernández-Lobato, Y. Li, M. Rowland, D. Hernández-Lobato, T. Bui, and R. E. Turner, "Black-box α-divergence minimization," in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 1511–1520.

[20] Y. Li and Y. Gal, "Dropout inference in bayesian neural networks with alpha-divergences," *arXiv preprint arXiv:1703.02914*, 2017.

[21] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, PhD thesis, University of Cambridge, 2016.

[22] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1019–1027.

[23] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 843–852.