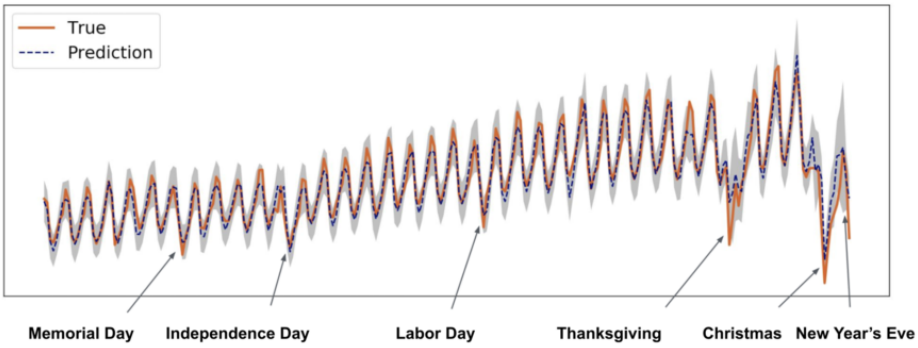
Engineering

Engineering Uncertainty Estimation in Neural Networks for Time Series Prediction at Uber

Lingxue Zhu and Nikolay Laptev September 6, 2017



Accurate time series forecasting during high variance segments (e.g., holidays and sporting events) is critical for anomaly detection, resource allocation, budget planning, and other related tasks necessary to facilitate optimal Uber user experiences at scale. Forecasting these variables, however, can be challenging because extreme event prediction depends on weather, city population growth, and other external factors that contribute to forecast uncertainty.

In recent years, the Long Short Term Memory (LSTM) technique has become a popular time series modeling framework due to its end-to-end modeling, ease of incorporating exogenous variables, and automatic feature extraction abilities. [1,2] By utilizing a large amount of data across numerous dimensions, an LSTM approach can model complex nonlinear feature interactions, which is critical for forecasting extreme events.[3]

Uncertainty estimation in deep learning remains a less trodden but increasingly important component of assessing forecast prediction truth in LSTM models. Through our research, we found that a neural network forecasting model is able to outperform classical time series methods in use cases with long, interdependent time series.

While beneficial in other ways, our new model did not offer insights into prediction uncertainty, which helps determine how much we can trust the forecast. Prediction uncertainty blindness also has a profound impact in anomaly detection; in Uber's case, this might result in large false anomaly rates during holidays where model prediction has large variance.

In this article, we introduce a new end-to-end Bayesian neural network (BNN) architecture that more accurately forecasts time series predictions and uncertainty estimations at scale. We also discuss how Uber has successfully applied this model to large-scale time series anomaly detection, enabling us to better accommodate rider demand during high-traffic intervals.[4]

Using BNNs at Uber

Engineering

Under the BNN framework, prediction uncertainty can be categorized into three types: model uncertainty, model misspecification, and inherent noise. Model uncertainty, also referred to as epistemic uncertainty, captures our ignorance of the model parameters and can be reduced as more samples are 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 vision.[5]

Long overlooked by most researchers, model misspecification captures the scenario where testing samples come from a different population than the training set, which is often the case in time series anomaly detection. Similar concepts have gained attention in deep learning under the concept of adversarial examples in computer vision, but its implication in prediction uncertainty remains relatively unexplored.[6]

In the following sections, we propose a principled solution to incorporate this uncertainty using an encoder-decoder framework. To the best of our knowledge, Uber's use case is one of the first times that misspecification uncertainty has been successfully applied to prediction and anomaly detection in a principled way.

Before we detail our use case, we discuss how we capture prediction uncertainty in BBN frameworks and its three types (model uncertainty, inherent noise, and model misspecification).

Calculating prediction uncertainty with BNNs

For the purpose of our model, 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 BNN, 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 as $p(y \mid f^W(x))$. In regression, we often assume: $y \mid W \sim N(f^W(x), \sigma^2)$ with some noise level, σ^2. In classification, the softmax likelihood is often used.

Given a set of *N* observations, $X = \{x_1, ..., x_N\}$ and $Y = \{y_1, ..., y_N\}$, Bayesian inference aims to find the posterior distribution over model parameters $P(W \mid X, Y)$. Finally, given a new data point x^*, the prediction distribution is obtained by marginalizing out the posterior distribution: $p(y^* \mid x^*) = \int p(y^* \mid f^W(x^*))\, p(W \mid X, Y)\, dW$.

In particular, the variance quantifies the prediction uncertainty, which can be broken down using the law of total variance: $Var(y^* \mid x^*) = Var(f^W(x^*)) + \sigma^2$. Immediately, we see that the variance is decomposed into two terms: $Var(f^W(x^*))$, which reflects our ignorance regarding the specifications of model parameter *W*, referred to as the model uncertainty, and σ^2, which represents the inherent noise.

An underlying assumption for the model uncertainty equation is that y^* is generated by the same procedure, but this is not always the case. In anomaly detection, for instance, it is

Engineering

The following three sections address how Uber handles BNN model uncertainty and its three categories when calculating our time series predictions.

Model uncertainty

The key to estimating model uncertainty is the posterior distribution $p(W|X,Y)$, also referred to as Bayesian inference. This is particularly challenging in neural networks because of the non-conjugacy often caused by nonlinearities. There have been various research efforts on approximate inference in deep learning, which we follow to approximate model uncertainty using the Monte Carlo dropout (MC dropout) method.[7,8]

The algorithm proceeds as follows: given a new input x^*, we compute the neural network output $\hat{y}^*$ with stochastic dropouts at each layer; in other words, randomly drop out each hidden unit with certain probability *p*. The 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 $Var(f^W(x^*)) \approx \frac{1}{B}\sum_{b=1}^{B}(\hat{y}^*_{(b)} - \bar{y}^*)^2$, where $\bar{y}^* = \frac{1}{B}\sum \hat{y}^*_{(b)}$.[9] In recent years, research has been conducted 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.[10]

In practice, we find that the uncertainty estimation is usually robust within a reasonable range of *p*.

Model misspecification

Next, we address the problem of capturing potential model misspecification through BNNs. We tackle this challenge by seeking to capture the uncertainty when predicting unseen samples with very different patterns from the training data set and aim to account for this source of uncertainty by training an encoder that extracts the representative features from a time series. At test time, the quality of encoding each sample will provide insight into how close it is to the training set.

Another way to frame this approach is that we must first fit a latent embedding space for all training time series using an encoder-decoder framework. From there, we are able measure the distance between test cases and training samples in the embedded space.

The next question we must address is how to combine this uncertainty with model uncertainty. Here, we take a principled approach by connecting the encoder-decoder network with a prediction network, and treat them as one large network during inference, as displayed in Algorithm 1 below:

Engineering

1: **for** $b = 1$ to B **do**
2: $e^*_{(b)} \leftarrow VariationalDropout(g(x^*), p)$
3: $z^*_{(b)} \leftarrow \text{Concatenate}(e^*_{(b)}, \text{extFeatures})$
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** $\hat{y}^*_{mc}, \eta_1$

Algorithm 1: Our MC dropout algorithm is used to approximate both model uncertainty and model misspecification.

Algorithm 1, above, illustrates such an inference network using the MC dropout algorithm. Specifically, given an input time series $x = \{x_1, ..., x_T\}$, the encoder $g(\cdot)$ constructs the learned embedding vector $e = g(x)$, which is further treated as feature input to the prediction network *h*.

During this feedforward pass, MC dropout is applied to all layers in both the encoder $g(\cdot)$ and the prediction network $h(\cdot)$. As a result, the random dropout in the encoder intelligently perturbs the input in the embedding space, which accounts for potential model misspecification and is further propagated through the prediction network. Here, variational dropout for recurrent neural networks is applied to the LSTM layers in the encoder, and regular dropout is applied to the prediction network.[11,12]

Inherent noise

Finally, we estimate the inherent noise level, σ^2. In the original MC dropout algorithm, this parameter is implicitly inferred from the prior over the smoothness of *W*. As a result, the model could end up with a drastically different estimation of the uncertainty level depending on the prespecified prior.[13] This dependency is undesirable in anomaly detection because we want the uncertainty estimation to also have robust frequentist coverage, yet it is rarely the case that we would know the correct noise level.

In this scenario, we propose a simple but adaptive approach by estimating the noise level via the residual sum of squares, evaluated on an independent held-out validation set. Specifically, let $f^{\widehat{W}}$ 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 by using the formula $\widehat{\sigma}^2 = \frac{1}{V}\sum_{v=1}^{V}(y'_v - f^{\widehat{W}}(x'_v))^2$.

Note that (X', Y') are independent from $f^{\widehat{W}}$. If we further assume that $f^{\widehat{W}}$ is an unbiased estimation of the true model, we have $E(\widehat{\sigma}^2) = \sigma^2 + Var_{TRN}(f^{\widehat{W}}(x'_v))$ where the bias term is $Var_{TRN}(\cdot)$ with respect to the training data, which decreases as the training sample size increases, and the bias approaches 0 as the training size N approaches ∞. Therefore, $\widehat{\sigma}^2$ provides an asymptotically unbiased estimation on the inherent noise level if the model is unbiased. Under finite sample scenario, $\widehat{\sigma}^2$ can only overestimate the noise level and tends to be more conservative.

The final inference algorithm in our BNN model combines inherent noise estimation with MC dropout and is presented in Algorithm 2, below:

Engineering

// prediction, model uncertainty and misspecification

$\hat{y}^*, \eta_1 \leftarrow MCdropout\ (x^*, g, h, p, B)$

// Inherent noise

for x'_v **in** validation set $\{x'_1, ..., x'_V\}$ **do**

$\hat{y'}_v \leftarrow h(g(x'_v))$

end for

$\eta_2^2 \leftarrow \frac{1}{V} \sum_{v=1}^{V} \left(\hat{y'}_v - y'_v\right)^2$

// total prediction uncertainty

$\eta \leftarrow \sqrt{\eta_1^2 + \eta_2^2}$

return $\hat{y}^*, \eta$

Algorithm 2: Our inference algorithm combines our inherent noise estimation and MC dropout algorithms.

In the following section, we take our understanding of BNNs and apply it to Uber's use case by introducing our time series prediction model.

BNN model design at Uber

The complete architecture of Uber's neural network contains two major components: (i) an encoder-decoder framework that captures the inherent pattern in the time series and is learned during pre-training, and (ii) a prediction network that receives input both from the learned embedding within the encoder-decoder framework as well as potential external features (e.g., weather events). This robust architecture is depicted in Figure 1, below:

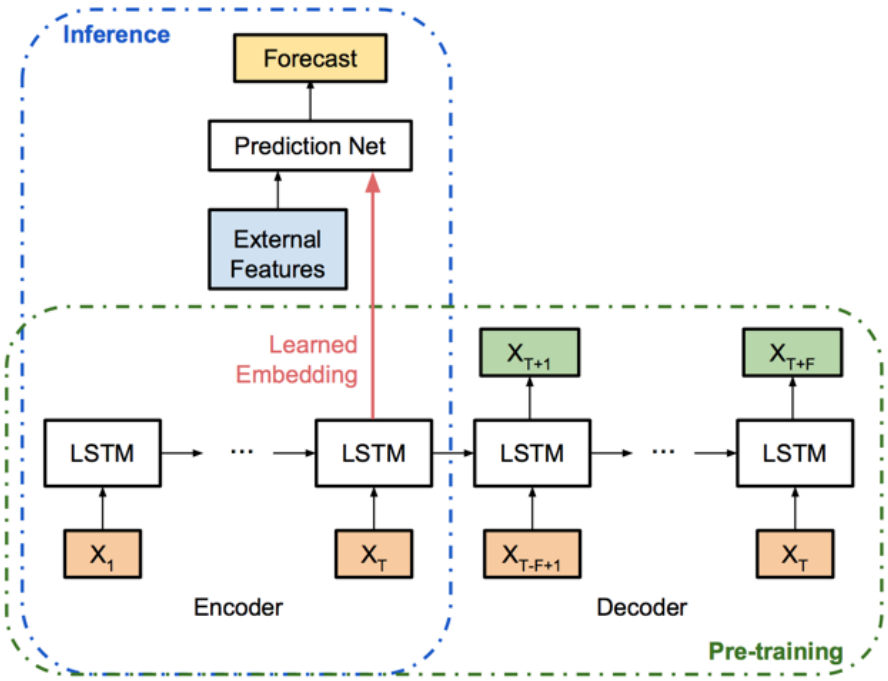


Figure 1: Our neural network architecture incorporates a pre-training phase using a LSTM encoder-decoder, followed by a prediction network. Input for this neural network includes learned embedding concatenated with external features.

Prior to fitting the prediction model, we first conduct pre-training to fit an encoder that can extract useful and representative embeddings from a time series. The goals are twofold: (i) to ensure that the learned embedding provides useful features for prediction and (ii) to certify that unusual input can be captured in the embedded space, which will further propagate to the prediction network.

guided via $\{w_{T-F+1}, \ldots, w_T\}$ (as showcased in the bottom panel of Figure 1). In order to construct the next few time steps from the embedding, it must contain the most representative and meaningful aspects from the input time series. This design is inspired from the success of video representation learning using a similar architecture.[14]

After the encoder-decoder is pre-trained, it is treated as an intelligent feature-extraction blackbox. Specifically, the LSTM cell states are extracted as learned fixed-dimensional 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.

After the full model is trained, the inference stage involves only the encoder and the prediction network. The complete inference algorithm is presented in Figure 1, where the prediction uncertainty contains two terms: (i) the inherent noise level, estimated on a held-out validation set, and (ii) the model and misspecification uncertainties, estimated by the sample variance of a number of stochastic feedforward passes where MC dropout is applied to both the encoder and the prediction network. Finally, an approximate α-level prediction interval is constructed by , where is the upper quantile of a standard normal.

Two hyper-parameters need to be specified for inference: the dropout probability, *p*, and the number of iterations, *B*. As for the dropout probability, the uncertainty estimation is relatively stable across a range of *p*, and so we choose the one that achieves the best performance on the validation set. As for the number of iterations, *B*, the standard error of the estimated prediction uncertainty is proportional to . We measure the standard error across different repetitions, and find that a few hundreds of iterations will suffice to achieve a stable estimation.

Next, we showcase our model's performance on a moderately sized data set of daily trips processed by the Uber platform by evaluating the prediction accuracy and the quality of uncertainty estimation during both holidays and non-holidays. We will also illustrate how we apply this model at scale to real-time anomaly detection for millions of metrics at Uber.

BNN application during extreme event forecasting at Uber

For the purpose of this article, we illustrate our BNN model's performance using the daily completed trips over four years across eight representative 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. Our encoder-decoder framework is constructed with two-layer LSTM cells containing 128 and 32 hidden states, respectively, and the prediction network is composed of three fully connected layers with tanh activation, containing 128, 64, and 16 hidden units, respectively.

Our samples are constructed using a sliding window where each sample contains the previous 28 days as input and aims to forecast the upcoming day. The raw data is log-transformed to alleviate exponential effects. Next, within each sliding window, the first day is subtracted from

Engineering

Prediction performance

We compared the prediction accuracy among four different models:

1. **Last-Day Model**: A naive model that uses the last day's completed trips as the prediction for the next day.
2. **Quantile Random Forest (QRF) Model:** Based on the naive last-day prediction, a quantile random forest 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. **Long Short-Term Memory Model**: A vanilla stacked LSTM with a similar size as Uber's prediction 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 the 28 days as input and predicts the next day.
4. **Uber's Model**: Our model with an encoder-decoder framework and a prediction network, as displayed in Figure 1.

Table 1, below, reports the symmetric mean absolute percentage error (SMAPE) of the four models evaluated against the testing set:

Table 1: The SMAPE is compared across four different prediction models and evaluated against the test data.

In the figure above, 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 percent improvement across the eight sampled cities.

As we further incorporate the encoder-decoder framework and introduce external features for holidays to the prediction network, our proposed model achieves another 36 percent improvement in prediction accuracy. Note that when using LSTM and our model, only one generic model is trained and 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.

Engineering

Figure 2: Daily completed trips in San Francisco during eight months of the testing set. True values are represented by the orange solid line, and predictions by the blue dashed line, where the 95 percent prediction band is shown as the grey area. (Note: exact values are anonymized.)

Through our SMAPE tests, we observed that accurate predictions are achieved for both normal days and holidays (e.g., days with high rider traffic).

Uncertainty estimation

Finally, we evaluate the quality of the uncertainty estimation by calibrating the empirical coverage of the predictive intervals. In this calculation, the dropout probability is set to be 5 percent at each layer. The intervals are constructed from the estimated predictive variance assuming Gaussian distribution. Table 2, below, reports the empirical coverage of the 95 percent prediction band under three different scenarios:

1. **Prediction Network (PredNet)**: Uses only model uncertainty estimated from MC dropout in the prediction network with no dropout layers in the encoder.
2. **Encoder + Prediction Network (Enc+Pred)**: Uses MC dropout in both the encoder and the prediction network, but without the inherent noise level.
3. **Encoder + Prediction Network + Inherent Noise Level (End+Pred+Noise)**: Uses the full prediction uncertainty as presented in Algorithm 2, including as in the Encoder + Prediction Network, as well as the inherent noise level, .

Table 2: Coverage of 95 percent predictive intervals in the Encoder + Prediction Network + Inherent Noise Level (Enc+Pred+Noise) scenario is determined using the test data.

addition, by further accounting for the inherent noise level, the empirical coverage of the final uncertainty estimation, Encoder + Prediction Network + Inherent Noise Level, nicely centers around 95 percent as desired.

One important use case of the uncertainty estimation is to provide insight for unusual patterns (e.g., anomalies) in a time series. Figure 3, below, shows the estimated predictive uncertainty on six U.S. holidays during our testing period:

Figure 3: The estimated prediction standard deviations on six U.S. holidays during our testing period suggests that New Year's Eve results in the largest standard deviations across all eight cities. (Note: exact values are anonymized.)

Our research indicates that New Year's Eve has significantly higher uncertainty than all other holidays. This pattern is consistent with our previous neural network forecasts, where New Year's Eve is usually the most difficult day to predict.

In the following section, we further interpret these results.

Embedding features

As previously discussed, the encoder is critical for both improving prediction accuracy as well as for estimating predictive 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, composed of points representing a 28-day time series segment, in the embedding space. We extract the cell states of the two LSTM layers, and project as a 2D space for visualization using Principal Component Analysis (PCA), as displayed in Figure 4, below:

Engineering

Figure 4: The time series training set, visualized in the embedding space, is composed of points that represent a 28-day segment, colored according to the day of week. In this PCA visualization, we evaluate the cell states of the two LSTM layers, where the first layer with dimension 128 is plotted on the left and the second layer with dimension 32 is plotted on the right.

As evidenced by our visualizations, the strongest pattern we observed is day of the week, where weekdays and weekends from different clusters, with Fridays usually sitting in between.

Applications to anomaly detection

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 percent predictive interval. There are two main challenges we need to address in this application, scalability, and performance, detailed below:

- **Scalability:** In order to provide real-time anomaly detection at Uber's scale, each predictive interval must be calculated within a few milliseconds during the inference stage. 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.
- **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.

In Figure 5, below, we illustrate the precision and recall of this framework on an example data set containing 100 metrics randomly selected with manual annotation available, where 17 of them are true anomalies:

Engineering

Figure 5: Four sample metrics (measured in minutes) track rider behavior during a 12-hour span, and anomaly detection is performed on the 30 minutes following this interval. The neural network constructs predictive intervals for the following 30 minutes, visualized by the shaded area in each plot.

Figure 5 depicts four different metrics representative of this framework: (a) a normal metric with large fluctuation, where the observation falls within the predictive interval; (b) a normal metric with small fluctuation following an unusual inflation; (c) an anomalous metric with a single spike that falls outside the predictive interval; and (d) an anomalous metric with two consecutive spikes, also captured by our model. (Note that this 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 percent recall rate and a 80.95 percent precision rate.

When applying this framework to all metrics, we observe a four percent improvement in precision compared to the previous ad-hoc solution, which is substantial at Uber's scale.

Next steps

Using the MC dropout technique and model misspecification distribution, we developed a simple way to provide uncertainty estimation for a BNN forecast at scale while providing 95 percent uncertainty coverage. Significantly, our proposal is applicable to any neural network without modifying the underlying architecture.

For special event uncertainty estimation, we found New Year's Eve to be the most uncertain time. Using uncertainty information, we adjusted the confidence bands of an internal anomaly detection model to improve precision during high uncertainty events, resulting in a four percent accuracy improvement, a large increase given the number of metrics we track at Uber.

This research has been accepted as a publication under the title "Deep and Confident Prediction for Time Series at Uber" and will be presented at the IEEE International Conference on Data Mining (ICDM) in New Orleans on November 18, 2017.

Engineering

at Uber!

Lingxue Zhu is a summer intern scientist at Uber's Intelligent Decision Systems team and is currently pursuing a Ph.D. in Statistics at Carnegie Mellon University.

Nikolay Laptev is a scientist on Uber's Intelligent Decision Systems team and a postdoctoral scholar at Stanford University.

Footnotes

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, 1997.

[2] M. Assaad, R. Bone´, and H. Cardot, "A new boosting algorithm for improved time-series forecasting with recurrent neural networks," *Inf. Fusion*, 2008.

[3] O. P. Ogunmolu, X. Gu, S. B. Jiang, and N. R. Gans, "Nonlinear systems identification using deep dynamic neural networks," *CoRR*, 2016.

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

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

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[7,9,11], & [13] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[8] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems 29*, 2016.

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

[12] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems 29*, 2016.

Comments

Engineering

Nikolay Laptev

Engineering