RDeepSense: Reliable Deep Mobile Computing Models with Uncertainty Estimations

SHUOCHAO YAO, University of Illinois Urbana Champaign
YIRAN ZHAO, University of Illinois Urbana Champaign
HUAJIE SHAO, University of Illinois Urbana Champaign
ASTON ZHANG, Amazon AI
CHAO ZHANG, University of Illinois Urbana Champaign
SHEN LI, IBM Research
TAREK ABDELZAHER, University of Illinois Urbana Champaign

Recent advances in deep learning have led various applications to unprecedented achievements, which could potentially bring higher intelligence to a broad spectrum of mobile and ubiquitous applications. Although existing studies have demonstrated the effectiveness and feasibility of running deep neural network inference operations on mobile and embedded devices, they overlooked the reliability of mobile computing models. Reliability measurements such as predictive uncertainty estimations are key factors for improving the decision accuracy and user experience. In this work, we propose RDeepSense, the first deep learning model that provides well-calibrated uncertainty estimations for resource-constrained mobile and embedded devices. RDeepSense enables the predictive uncertainty by adopting a tunable proper scoring rule as the training criterion and dropout as the implicit Bayesian approximation, which theoretically proves its correctness. To reduce the computational complexity, RDeepSense employs efficient dropout and predictive distribution estimation instead of the model ensemble or sampling-based method for inference operations. We evaluate RDeepSense with four mobile sensing applications using Intel Edison devices. Results show that RDeepSense can reduce around 90% of the energy consumption while producing superior uncertainty estimations and preserving at least the same model accuracy compared with other state-of-the-art methods.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing theory, concepts and paradigms**; • **Computing methodologies → Machine learning**;

Additional Key Words and Phrases: Internet-of-Things, Deep Learning, Mobile Computing, Reliability, Uncertainty Estimation

ACM Reference Format:
Shuochao Yao, Yiran Zhao, Huajie Shao, Aston Zhang, Chao Zhang, Shen Li, and Tarek Abdelzaher. 2017. RDeepSense: Reliable Deep Mobile Computing Models with Uncertainty Estimations. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4, Article 173 (December 2017), 26 pages. https://doi.org/10.1145/3161181

1 INTRODUCTION

Using embedded sensors to infer the surrounding physical states and context is one of the major tasks of mobile and ubiquitous computing. Numerous mobile applications have prospered in a wide range of areas, such as

Authors' addresses: Shuochao Yao, University of Illinois Urbana Champaign; Yiran Zhao, University of Illinois Urbana Champaign; Huajie Shao, University of Illinois Urbana Champaign; Aston Zhang, Amazon AI; Chao Zhang, University of Illinois Urbana Champaign; Shen Li, IBM Research; Tarek Abdelzaher, University of Illinois Urbana Champaign.

2474-9567/2017/12-ART173 $15.00
https://doi.org/10.1145/3161181

health and wellbeing [3, 5, 14, 23, 28, 48, 49, 62], behavior and activity recognition [8, 9, 38, 39, 42, 52, 57], crowd sensing [53, 54, 58, 59, 61, 63], tracking and localization [10, 24, 26, 31, 51, 60]. An important component in these applications is a learning model that outputs target values given sensor inputs.

Rapid advancement in deep learning techniques has tempted researchers to employ deep neural networks as the learning models in the mobile applications. These highly capable models are good at making sophisticated mappings between unstructured data such as sensor inputs and target quantities, which can hardly be achieved by traditional machine learning models. Specific deep learning models have been designed to fuse multiple sensory modalities and extract temporal relationships along sensor inputs. These specifically designed models have shown significant improvements on audio sensing [35], tracking and localization [55], human activity recognition [13, 25, 44, 55], and user identification tasks [55].

However, the inability of treating the deep learning model more than just an incomprehensible black box has become an important factor that hinders researchers from applying the model to mobile applications. The complexity and uninterpretability of such models mainly result from the deep and non-linear structures [37]. Therefore researchers can hardly understand how deep neural networks derive their final predictions. This leads to either the loss of trust in deep learning models or blind faith in deep learning models without being aware of predictive uncertainties and error bound.

In order to explicitly output the reliability measure of deep neural network model, we aim to provide the model with predictive uncertainties during inference. Predictive uncertainty is defined as the probability of occurrence of the target variable conditioned on all available information. One particular approach to express predictive uncertainty is to treat the model predictions as random variables, *i.e.*, in the form of probability distributions instead of point estimations [43]. In this paper, we center our discussion around this specific representation of the uncertainty.

On one hand, although it is hard to directly interpret deep neural networks, predictive uncertainty can provide the quantitative confidence of prediction correctness, which boosts trust and faith in deep learning models. On the other hand, uncertainty estimation itself is crucial for scientific measurements [19, 33]. Extensive investigations show that measurement uncertainties can impact user experiences [30, 36]. In order to monitor the uncertainties of mobile sensing applications, the first important step is to obtain the predictive uncertainties of learning models used in the applications, which, in our case, are deep neural network models.

However, enabling deep neural networks to provide high-quality and well-calibrated uncertainty estimations on mobile and embedded devices poses two major challenges. One challenge is to provide a mathematically grounded uncertainty estimations that require few changes on either the model or the optimization method. Although mathematically grounded methods such as Bayesian approaches serves as powerful tools to estimate predictive uncertainties [11], Bayesian neural networks are computationally expensive to train and inference even for brawny servers, let alone mobile and embedded devices [43]. Therefore a mathematically grounded theory under minimal model modification requirements is a must for reliable uncertainty estimations.

The other challenge is to reduce the computational burden of uncertainty estimations during inference. For mobile and ubiquitous computing applications, although we can train the deep neural networks on brawny servers with powerful GPUs, running inference on mobile and embedded devices is difficult due to limited energy supplies and computational resources on such devices [56]. Illuminating studies from the machine learning community try to provide mathematically grounded uncertainty estimations for deep neural networks, but these methods are based either on the sampling method [17] or the ensemble method [34]. They require either running a single stochastic neural network for multiple times or training and running multiple deterministic neural networks. All these solutions are not resource-friendly to mobile and embedded devices. Therefore, mobile applications call for a novel solution that theoretically guarantees the correctness of predictive uncertainties, and at the same time consumes much less resource.

In this work, we propose RDeepSense that enables predictive uncertainties with theoretically proven correctness for mobile and ubiquitous applications. RDeepSense significantly reduces the computational overhead and preserves at least the same model accuracy. To the best of our knowledge, this is the first deep learning model that provides uncertainty estimations for resource-limited devices. The core of RDeepSense is the integration of the dropout training method that interprets neural networks as Gaussian process (GP) through Bayesian approximation [17, 45, 46] and proper scoring rules as training criterion that measure the quality of predictive uncertainty such as log-likelihood and the Brier score [20]. Their integration can be further interpreted as the mixture distribution of a Gaussian or categorical distribution based on latent deep Gaussian process and a deep Gaussian process through Bayesian approximation. Firstly, RDeepSense uses a tunable proper scoring rule as the training criterion that significantly mitigates the problem of underestimating predictive uncertainties in deep neural networks [17]. Secondly, since dropout training can be interpreted as "geometric averaging" over the ensemble of possible "thinned" subnetworks [2], RDeepSense applies dropout training instead of model ensemble. It greatly reduces the computation complexity of the final neural network compared with model ensemble. Therefore, our integrated method incurs only little computational overhead, which makes it feasible on embedded devices for mobile applications.

Evaluations of RDeepSense use the Intel Edison computing platform [1]. We conduct mobile and ubiquitous tasks that focus on human health and wellbeing, smart city transportation, environment monitoring, and activity recognition. Specifically, our experiments include: 1) monitoring arterial blood pressure through photoplethysmogram (PPG) from fingertip [27], 2) NY city taxi commute time estimation [50], 3) gas mixture concentrations estimation through the chemical sensor array [15], 4) and heterogeneous human activity recognition through motion sensors [47].

We compare RDeepSense with the state-of-the-art Monte Carlo dropout method [17], ensemble method [34], and Gaussian process. The resource consumption of Intel Edison module and final model performance such as the accuracy and the quality of uncertainty estimations are measured for all the algorithms. RDeepSense can reduce more than 90% of inference time and energy consumption, while obtaining the uncertainty estimations with better quality compared with the other algorithms. The well-calibrated uncertainty estimations and resource efficiency make RDeepSense the first choice to obtain uncertainty estimations of deep neural networks in mobile applications.

In summary, we propose a simple yet effective and theoretically-grounded method, RDeepSense, which empowers neural networks with well-calibrated predictive uncertainty estimations. RDeepSense is also a resource-friendly algorithm for mobile and embedded devices that adds almost no computational overhead during model inference.

The rest of paper is organized as follows. Section 2 introduces related works about uncertainty estimations and deep neural networks. We describe the technical details of RDeepSense in Section 3. The evaluation is presented in Section 4. Finally, we discuss the results in Section 5 and conclude in Section 6.

2 RELATED WORK

On one hand, reliability and uncertainty estimation is one important issue of mobile and ubiquitous computing. A lot of works have been proposed to utilize uncertainty estimations for improving the decision accuracy and user experience. Baumann et al. [4] make next-place predictions based on the uncertainty estimation of classifiers. Kay et al. [29] propose a novel discrete representation of uncertainties for visualizing and user interaction. Boukhelifa et al. [7] propose design considerations for uncertainty-aware data analytics. On the other hand, the recent advances in deep learning techniques have motivated people to apply deep neural networks for solving mobile and ubiquitous computing tasks. Lane et al. [35] apply deep neural networks to solve audio sensing tasks. Castro et al. [13] predict daily activities from egocentric images using deep learning. Yao et al. [55] propose a

deep learning structure that fuses multiple sensor inputs and extracts time dependencies. Guan et al. [25] apply ensembles of LSTM for activity recognition. However, uncertainty estimations of deep neural networks for mobile and ubiquitous computing tasks is an important topic that draws less attention.

Table 1. Comparison among deep learning based predictive uncertainty estimation

Algorithm	Dropout Training	Proper Scoring Rules	Ensemble method	Obtain predictive uncertainty with single run
RDeepSense	✓	✓	×	✓
MCDrop	✓	×	×	×
SSP	×	✓	✓	×

Recently there are some illuminating works from the machine learning community that tries to provide deep neural networks with uncertainty estimations. Gal et al. [17] provide the first theoretical proof of the linkage between dropout training with deep Gaussian process called MCDrop. However, the proposed method tends to underestimate the uncertainty due to the nature of variational inference. Lakshminarayanan et al. [34] propose a solution SSP based on proper scoring rules and ensemble methods. However, the proposed method tends to overestimate the uncertainty on real datasets.

Since these previous works do not consider the scenario of mobile and ubiquitous computing, all these proposed methods require the operations with high computational cost during model inference, *i.e.*, sampling methods or ensemble methods. These computationally intensive operations aggravate the time and energy consumption problems in the embedded devices, which is one of the key issues of mobile and ubiquitous computing [22, 40, 41].

To the best of our knowledge, RDeepSense is the first work that provides a simple yet effective solution to estimate the uncertainties of deep neural networks for mobile and ubiquitous computing applications. RDeepSense uses proper scoring rules to mitigate the underestimation effect of MCDrop, and applies dropout training as implicit ensemble to avoid the computationally intensive ensemble method used in SSP.

In order to further illustrate the main difference between RDeepSense and other two deep leaning uncertainty estimation algorithms, MCDrop and SSP, we show the designing components of these three algorithms in Table 1.

3 RDEEPSENSE FRAMEWORK

This section elaborates on the technical details of the RDeepSense framework in three constituents. Section 3.1 introduces a simple yet effective recipe to build a fully-connected neural network with predictive uncertainty estimations. In Section 3.2, we introduce preliminary knowledge and make the theoretical analysis of RDeepSense. We prove that RDeepSense is a mathematically grounded method to obtain predictive uncertainty estimations. In Section 3.3, we introduce an effective and efficient approximation for RDeepSense to obtain predictive uncertainty estimations while running on the resource-constrained embedded devices.

For the rest of this paper, all vectors are denoted by bold lower-case letters (*e.g.*, $\mathbf{x}$ and $\mathbf{y}$), and matrices and tensors are represented by bold upper-case letters (*e.g.*, $\mathbf{X}$ and $\mathbf{Y}$). For a column vector $\mathbf{x}$, the j^{th} element is denoted by $\mathbf{x}_{[j]}$. For a tensor $\mathbf{X}$, the t^{th} matrix along the third axis is denoted by $\mathbf{X}_{..t}$, and the other slicing notations are defined similarly. The superscript l in $\mathbf{x}^{(l)}$ and $\mathbf{X}^{(l)}$ denote the vector and tensor for the l^{th} layer of the neural network. We use calligraphic letters to denote sets (*e.g.*, $\mathcal{X}$ and $\mathcal{Y}$), where $|\mathcal{X}|$ denotes the cardinality of $\mathcal{X}$.

3.1 RDeepSense components

RDeepSense is a simple and effective method that empowers fully-connected neural networks to output predictive uncertainty estimations. There are only two steps to convert an arbitrary fully-connected neural networks into a neural network with uncertainty estimations:

(1) Insert dropout operation to each fully-connected layer.

(2) Adopt a proper scoring rule as the loss function, and emit a distribution estimation instead of a point estimation at the output layer.

The following two subsections describe dropout training and proper scoring rules in detail.

3.1.1 Dropout training. Fully-connected neural networks can be formulated using the following equations:

$$\begin{aligned} \mathbf{y}^{(l)} &= \mathbf{x}^{(l)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}, \\ \mathbf{x}^{(l+1)} &= f^{(l)}\left(\mathbf{y}^{(l)}\right), \end{aligned} \tag{1}$$

where the notation $l = 1, \cdots, L$ is the layer index in the fully-connected neural network. For any layer l, the weight matrix is denoted as $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$; the bias vector is denoted as $\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)}}$; the input is denoted as $\mathbf{x}^{(l)} \in \mathbb{R}^{d^{(l-1)}}$; and $d^{(l)}$ is the dimension of the l^{th} layer. In addition, $f^{(l)}(\cdot)$ is a nonlinear activation function.

However, such formulations could run into feature co-adapting and model overfitting problems. To avoid these problems, researchers introduce the concept of dropout as a regularization method [46]. "Dropout" originally refers to dropping out hidden and visible units in a neural network, which is mathematically equivalent to ignoring rows of the weight matrix $\mathbf{W}^{(l)}$. Therefore, a fully-connected neural network with dropout can be represented as follows:

$$\begin{aligned} \mathbf{z}^{(l)}_{[i]} &\sim \text{Bernoulli}(\mathbf{p}^{(l)}_{[i]}), \\ \tilde{\mathbf{W}}^{(l)} &= \text{diag}\left(\mathbf{z}^{(l)}\right)\mathbf{W}^{(l)}, \\ \mathbf{y}^{(l)} &= \mathbf{x}^{(l)}\tilde{\mathbf{W}}^{(l)} + \mathbf{b}^{(l)}, \\ \mathbf{x}^{(l+1)} &= f^{(l)}\left(\mathbf{y}^{(l)}\right). \end{aligned} \tag{2}$$

As shown in (2), a vector of Bernoulli variables $\mathbf{z}^{(l)} \in \{0, 1\}^{d^{(l-1)}}$ forms a diagonal matrix which acts as a mask to dropout the i^{th} row of $\tilde{\mathbf{W}}^{(l)}$ with probability $\mathbf{p}^{(l)}_{[i]}$. Intuitively, the dropout operations (2) convert a traditional (deterministic) neural network with parameters $\{\mathbf{W}^{(l)}\}$ into a random Bayesian neural network with random variables $\{\tilde{\mathbf{W}}^{(l)}\}$, which equates a neural network with a statistical model without using the Bayesian approach explicitly. This conversion with dropout helps us to obtain predictive uncertainty estimations and avoid the computationally intensive operations used in Bayesian approaches. The detailed analysis about the equivalence will be discussed later.

3.1.2 Proper scoring rules. Optimizing a deep neural network requires minimizing the loss function. Therefore the loss function plays a crucial role in designing an effective neural network. Many commonly used neural network loss functions are proper scoring rules, such as logistic loss and hinge loss.

Scoring rules, also known as score functions, measure the quality of predictive uncertainties [20]. Assume that $p_\theta(y|\mathbf{x})$ is the probabilistic distribution represented by a deep neural network. The scoring rule $S(p_\theta(y|\mathbf{x}), (\mathbf{x}, y))$ assigns a numerical score for the quality of predictive distribution $p_\theta(y|\mathbf{x})$ on event $(\mathbf{x}, y) \sim q(\mathbf{x}, y)$, where $q(\mathbf{x}, y)$ is the true distribution of data samples. The expected scoring rule is formulated as

$$S(p_\theta(y|\mathbf{x}), q(\mathbf{x}, y)) = \int q(\mathbf{x}, y) S(p_\theta(y|\mathbf{x}), (\mathbf{x}, y)) d\mathbf{x} dy. \tag{3}$$

For a proper scoring rule, the equality in $S(p_\theta(y|\mathbf{x}), q(\mathbf{x}, y)) \geq S(q(\mathbf{x}, y)), q(\mathbf{x}, y))$ holds if and only if $p_\theta(y|\mathbf{x}) = q(\mathbf{x}, y)$. Widely-adopted proper scoring rules include Log-likelihood $\log p_\theta(y|\mathbf{x})$ and Brier score $-\sum_{k=1}^{K}(\mathbb{1}_k(y) - p_\theta(y = k|\mathbf{x}))^2$.

RDeepSense employs a tunable function, the weighted sum of negative log-likelihood and mean square error (Brier score for classification problems), which is a proper scoring rule, as the loss functions for both regression

and classification problems. This loss function tries to offset the effect of overestimation and underestimation caused by negative log-likelihood and mean square error respectively, which will be analyzed and evaluated later.

For regression problems, in order to optimize the neural network with negative log-likelihood, we have to emit a distribution estimation instead of a point estimation at the output layer. Therefore, we slightly change the structures of neural networks. The last output layer generates both the predictive mean $\mu(\hat{y})$ and the predictive variance $\sigma^2(\hat{y})$. According to the notation in (2), the output layer is represented by $\mathbf{x}^{L+1} = \left[\mu(\hat{y}),\ \sigma^2(\hat{y})\right]^\intercal = \left[\mathbf{y}^{(L)}_{[0]},\ \text{softplus}(\mathbf{y}^{(L)}_{[1]})\right]^\intercal$, where softplus function is $\log(1 + \exp(\cdot))$ enforcing the positivity constraint on the variance. Predictive mean $\mu(\hat{y})$ and predictive variance $\sigma^2(\hat{y})$ compose a Gaussian distribution $\mathcal{N}(\mu(\hat{y}), \sigma^2(\hat{y}))$ as the output predictive distribution of the neural network.

Then the final loss function of a regression problem, $\mathcal{L}_r$, is the weighted sum of mean square error $\mathcal{L}_{re}$ and negative log-likelihood $\mathcal{L}_{rl}$,

$$\begin{aligned}
\mathcal{L}_{re} &= \sum_{n=1}^{N} \left(y - \mu(\hat{y})\right)^2 + \lambda_e \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_2^2, \\
\mathcal{L}_{rl} &= \sum_{n=1}^{N} \left(\frac{1}{2}\log\sigma^2(\hat{y}) + \frac{1}{2\sigma^2(\hat{y})}\left(y - \mu(\hat{y})\right)^2\right) + \lambda_l \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_2^2, \\
\mathcal{L}_r &= (1-\alpha)\cdot\mathcal{L}_{rl} + \alpha\cdot\mathcal{L}_{re},
\end{aligned} \tag{4}$$

where N is the number of training samples, the second term in the first two equations are the L_2 regularization, and α is a hyper-parameter.

As we will discuss in Section 3.2.3 and evaluate in Section 4.6, a larger α leads neural networks to focus more on estimating an accurate mean value, which may underestimate the true uncertainties, while a smaller α leads neural networks to estimate a larger variance during the optimization process, which may overestimate the true uncertainties. Therefore, α is a hyper-parameter that makes the bias-variance tradeoff and is tuned to generate a well-calibrated predictive uncertainty, *i.e.*, neither underestimation nor overestimation.

For the classification problem, $f^{(L)}(\cdot)$ is the softmax function that generates predictive probabilities for each category. The final loss function of a classification problem, $\mathcal{L}_c$, is the weighted sum of mean square error $\mathcal{L}_{ce}$ and negative log-likelihood $\mathcal{L}_{cl}$,

$$\begin{aligned}
\mathcal{L}_{ce} &= \sum_{n=1}^{N}\sum_{k=1}^{K} (\mathbb{1}_k(y) - p_\theta(y=k|\mathbf{x}))^2 + \lambda_e \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_2^2, \\
\mathcal{L}_{cl} &= \sum_{n=1}^{N} -\log p_{\mathcal{W}}(\hat{y} = y|\mathbf{x}) + \lambda_l \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_2^2, \\
\mathcal{L}_c &= (1-\alpha)\cdot\mathcal{L}_{cl} + \alpha\cdot\mathcal{L}_{ce},
\end{aligned} \tag{5}$$

where N is the number of training samples, K is the number of classes, the second term in the first two equations are the L_2 regularization, and α is a hyper-parameter.

In summary, the whole neural network is optimized through a tunable proper scoring rule that maximizes the quality of predictive uncertainties. The detailed theoretical backup and proof of the equivalence between RDeepSense and a statistical model will be shown in Section 3.2.

3.2 Theoretical analysis: the equivalence between RDeepSense and statistical models

Uncertainty estimations are usually inferred by a statistical model, such as a gaussian process [45] and a graphical model [32]. This section provides the theoretical bases for using RDeepSense to estimate predictive uncertainties

by proving the equivalence between the RDeepSense model and a statistical model. To achieve this goal, we first summarize the preliminary knowledge about the equivalence between dropout training with mean square error and a deep Gaussian process, which is proposed by Gal et al. [17] in Section 3.2.1. Then we prove the equivalence between dropout with the proper scoring rule (log-likelihood) and a Gaussian or categorical distributions based on latent deep Gaussian process in Section 3.2.2. Finally, in Section 3.2.3, we generalize the analysis to another tunable proper scoring rule, weighted sum of log-likelihood and negative mean square error, which provides the theoretical foundation for the RDeepSense.

3.2.1 Preliminary: Dropout with mean square error. Gaussian process is a powerful statistical tool that allows us to model distribution over functions [45]. We show the detailed proof in Appendix A.1 that optimizing a variational approximation of deep Gaussian process is equivalent to optimizing an dropout neural network based on mean square error as the loss function, which is first discussed and proven by Gal et al. [17].

However, mean square error is not a proper scoring rule for regression problems, which cannot generate a well calibrated uncertainty estimations. Besides, due to the mode matching nature of KL divergence, the variational approximating usually generates a highly underestimated predictive uncertainty [6], which is also verified in our experiments in Section 4.4. Therefore we further discuss the case of dropout training with proper scoring rules in Section 3.2.2 and Section 3.2.3, which enables RDeepSense to provide a high quality uncertainty estimation.

3.2.2 Dropout with negative log-likelihood. We have introduced the previous work that treats a neural network with dropout training based on mean square error loss function as a deep Gaussian process with variational approximation. We call this method MCDrop.

However, there are two drawbacks for MCDrop. One is the underestimation of predictive distribution. Variational Bayesian used in MCDrop is known to provide underestimated posterior uncertainty, because optimizing the KL divergence will generate a low-variance estimation to a single mode of true posterior distribution [6]. In addition, the loss function of MCDrop is not a proper scoring rule that can help to mitigate the negative effect of underestimation caused by the variational Bayesian method. Underestimation is not a desirable property for mobile and ubiquitous computing applications, because it means that the deep neural network will always be over-confident about its prediction results.

The other drawback of MCDrop is the high computational burden during uncertainty estimation. Since the output of MCDrop is a stochastic point estimation, Monte Carlo sampling method is required to estimate the predictive mean and variance. Therefore we need to run the whole neural network for multiple times, *i.e.*, running k times for k samples, to generate the predictive uncertainty. Since running time and energy consumption are two crucial problems for mobile and ubiquitous computing applications, MCDrop is not a suitable solution for applications running on embedded devices.

Therefore, we integrate proper scoring rules and dropout training in RDeepSense to solve the aforementioned two drawbacks. The proper scoring rules such as log-likelihood help to reduce or even erase the underestimation effect of MCDrop, because proper scoring rule is a score function that gives higher quality uncertainty estimations more credits. In addition, since a neural network with proper score rule directly generates a predictive distribution estimation instead of a point estimation, we can efficiently obtain an approximated expectation of uncertainty estimation through dropout inference. At the same time, dropout as Bayesian approximation can provide a equivalence between the deep neural network and a statistical model, which guarantees RDeepSense to be a mathematically grounded uncertainty estimation method.

We show the detailed proof in Appendix A.2 that training a fully-connected neural network with dropout and negative log-likelihood loss function is equivalent to a Gaussian or categorical distribution based on the latent deep Gaussian process.

3.2.3 Dropout with weighted sum of negative log-likelihood and mean square error. Training a neural network with a proper scoring rule, log-likelihood loss, should generate predictive uncertainty estimations that faithfully reflect the probability that the prediction will happen. However, training a neural network will log-likelihood

loss solely could converge to a local optima that overestimates the true uncertainty empirically, which will be shown in our evaluation Section 4.4.

The intuitive explanation for this phenomenon is straight-forward. During the early phase of training a neural network with log-likelihood loss, it is relatively hard to generate an accurate estimation of predictive mean. Then increasing the value of variance estimation can consistently decrease the negative log-likelihood loss with a high probability, since there is only a logarithm term that prevents variance from increasing as shown in (4). Therefore, the predictive uncertainty tends to favor an estimation with large variance that overestimates the true uncertainty. As a result, although log-likelihood loss is a proper score rule that assigns more credits to predictive uncertainties with higher quality, it usually fails to achieve a good bias-variance tradeoff during training process in practice.

In order to achieve a well-calibrated uncertainty estimation, *i.e.*, an estimation that neither underestimates nor overestimates, we design a tunable proper scoring rule as the training objective function of RDeepSense. It is a weighted sum of log-likelihood and negative mean square error controlled by a hyper-parameter α,

$$(1-\alpha)\cdot \log p_{\mathcal{W}}(\hat{y}=y|\mathbf{x}) - \alpha\cdot(\hat{y}-y)^2. \tag{6}$$

With the definition in Section 3.1.2, we can easily see that (6) is a proper scoring rule.

According to the analysis in the previous two subsections 3.2.1 and 3.2.2, we can see that RDeepSense, training fully-connected neural network by maximizing the weighted sum of log-likelihood and negative mean square error, is equivalent to the mixture distribution of a Gaussian or categorical distribution based on the latent deep Gaussian process and a deep Gaussian process.

Since training solely with negative mean square error or log-likelihood tends to underestimate or overestimate the predictive uncertainties respectively, it is easy to fine-tune the hyper-parameter α with the validation dataset. When the predictive uncertainty is underestimated, we decrease the value α, and vice versa. The detailed analysis of the effect of hyper-parameter α will be illustrated in Section 4.6.

3.3 RDeepSense uncertainty estimation

The previous sections prove that RDeepSense is a mathematically grounded method to estimate predictive uncertainties for fully-connected neural networks. In this section, we show that RDeepSense can efficiently estimate predictive uncertainties of fully-connected neural networks with only little computational overhead.

According to the analysis in Appendix A.2, the approximated predictive distribution is

$$q(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x},\mathcal{W})q(\mathcal{W})d\mathcal{W} = \mathbb{E}_{q(\mathcal{W})}\Big[p(\mathbf{y}|\mathbf{x},\mathcal{W})\Big], \tag{7}$$

where $\mathcal{W} = \{\tilde{\mathbf{W}}^{(l)}\}$ is the random variables generated by dropout operations at each layer.

$$\begin{aligned} \mathbf{z}_{[i]}^{(l)} &\sim \text{Bernoulli}(\mathbf{p}_{[i]}^{(l)}), \\ \tilde{\mathbf{W}}^{(l)} &= \text{diag}\Big(\mathbf{z}^{(l)}\Big)\mathbf{W}^{(l)}. \end{aligned} \tag{8}$$

Usually Monte Carlo estimation is used to approximate the predictive distribution $q(\mathbf{y}|\mathbf{x})$ through sampling random variables $\mathcal{W}$,

$$q(\mathbf{y}|\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} p(\mathbf{y}|\mathbf{x},\mathcal{W}_m). \tag{9}$$

For classification, (9) is the average of categorical distribution. For regression, (9) is an average of Gaussian distributions. If we assume that M Gaussian distributions are independent, the resulted average distribution can

be approximated by a single Gaussian distribution according to the central limit theorem,

$$\begin{aligned}\frac{1}{M}\sum_{m=1}^{M} p(\mathbf{y}|\mathbf{x}, \mathcal{W}_m) &= \sum_{m=1}^{M} \mathcal{N}(\mu_m(\mathbf{x}), \sigma_m^2(\mathbf{x})) \\ &= \mathcal{N}(\hat{\mu}(\mathbf{x}), \hat{\sigma}^2(\mathbf{x})), \\ \hat{\mu}(\mathbf{x}) &= \frac{1}{M}\sum_{m=1}^{M} \mu_m(\mathbf{x}), \\ \hat{\sigma}^2(\mathbf{x}) &= \frac{1}{M}\sum_{m=1}^{M} \left(\sigma_m^2(\mathbf{x}) + \mu_m^2(\mathbf{x})\right) - \hat{\mu}^2(\mathbf{x}).\end{aligned} \tag{10}$$

The drawback of Monte Carlo estimation for embedded devices is its high energy and time consumptions. We have to run the whole neural network for M times to generate M samples, which is not suitable for embedded devices with limited resources.

Fortunately, there is a simple yet effective recipe proposed by the dropout operation that can effectively approximate the expected output value instead of using Monte Carlo estimation [46]. During test time, the dropout operation is changed from (2) into

$$\begin{aligned}\tilde{\mathbf{W}}^{(l)} &= \text{diag}\left(\mathbf{p}^{(l)}\right)\mathbf{W}^{(l)}, \\ \mathbf{y}^{(l)} &= \mathbf{x}^{(l)}\tilde{\mathbf{W}}^{(l)} + \mathbf{b}^{(l)}, \\ \mathbf{x}^{(l+1)} &= f^{(l)}\left(\mathbf{y}^{(l)}\right).\end{aligned} \tag{11}$$

Although the approximation (11) is not theoretically equivalent to the Monte Carlo estimation (10) by assuming the zero variance of mean estimation, $\sum_{m=1}^{M} \mu_m^2(\mathbf{x}) - (\sum_{m=1}^{M} \mu_m(\mathbf{x}))^2 = 0$, the proposed approximation (11) turns to be an effective and efficient approximation during the evaluation in Section 4. In the evaluation section, we will empirically compare the biased approximation (11) with the unbiased Monte Carlo estimation (10).

Therefore, with the approximation (11), we can directly estimate the expected predictive mean and variance of a Gaussian distribution for regression problems and expected categorical probabilities for classification problems by just running the neural network for a single time. This makes RDeepSense a suitable candidate for deep neural networks with uncertainty estimations used in mobile and ubiquitous computing applications.

4 EVALUATION

In this section, we evaluate RDeepSense on four mobile and ubiquitous computing tasks. We first introduce the experimental setup for each task, including hardware, datasets, and baseline algorithms. We then evaluate the accuracy and the quality of uncertainty estimation. Next we evaluate the inference time and energy consumption of all algorithms on the testing hardware. At last we evaluate and analyze the effect of hyper-parameter α in the training objective function (6) on the model performance such as accuracy and quality of uncertainty estimation.

4.1 Testing hardware

Our testing hardware is based on Intel Edison computing platform [1]. The Intel Edison computing platform is powered by the Intel Atom SoC dual-core CPU at 500 MHz and is equipped with 1GB memory and 4GB flash storage. For fairness, all neural network models are run solely on CPU during evaluation for inference time and energy consumption.

Table 2. Statistical Information of four datasets used in evaluations

Dataset	Training Size	Validating Size	Testing Size	Mean of output	Std of output	Range of output
BPEst	1,281,098	26,689	26,689	88.74	25.01	[50.0, 199.93]
NYCommute	10,287,766	214,328	214,328	15.08	52.79	[0.0, 1439.5]
GasSen	2,839,933	59,166	59,166	94.56	145.16	[0.0, 533.33]
HHAR	28,314	1,686	1,686	N/A	N/A	{0, 1, 2, 3, 4, 5}

4.2 Evaluation tasks

We conduct four experiments related to human health and wellbeing, smart city transportation, environment monitoring, and human activity recognition with RDeepSense and other two state-of-the-art deep learning uncertainty measuring methods as well as a statistical model. The experimental settings of the tasks and datasets are introduced in this subsection.

The detailed statistical information of four datasets is illustrated in Table 2

- *BPEst: Cuffless blood pressure monitoring through photoplethysmogram.* The first task is to monitor cuffless blood pressure through photoplethysmogram from fingertip. The dataset is originally collected by patient monitors at various hospitals between 2001 and 2008. Waveform signals were sampled at the frequency of 125 Hz with at least 8 bit accuracy [21]. The photoplethysmogram from fingertip (PPG) and arterial blood pressure (ABP) signal (mmHg) is extracted by Mohamad et al. for the non-invasive cuffless blood pressure monitoring task [27].[1] The target of BPEst task is to infer the waveform of ABP based on the waveform of PPG collected from fingertips. This is a more challenging task compared with estimating the upper and lower bound of the blood pressure, which requires a more precise estimation of predictive uncertainty. During the experiment, a learning model is trained to estimate a 2-second ABP waveform (250 samples) based on the corresponding 2-second PPG waveform.
- *NYCommute: Commute time estimation of New York City.* Smart transportation is an increasingly important task within the topic of smart city. The second task is to estimate commute time in New York City through the pick-up time and location as well as the drop-off location. We use the yellow and green taxi trip records within January 2017 as the training, validation, and testing dataset.[2] The input of the learning model is a vector with 5 elements, containing the standardized longitude and latitude of pick-up and drop-off location as well as the pick-up time within a day. The output of the learning model is the expected commute time and its corresponding uncertainty estimation.
- *GasSen: Estimate dynamic gas mixtures from chemical sensors.* The third task is related to the environment monitoring. The task is to estimate real concentration of Ethylene and CO gas mixture from an array of low-end chemical sensors. Fonollosa et al. constructed the dataset by the continuous acquisition the signals of a sensor array with 16 chemical sensors for a duration of about 12 hours without interruption with the sampling frequency of 100 Hz [15]. [3] Gas concentrations range from $0 - 600$ parts-per-million (ppm). The learning model is trained and tested to predict the concentration Ethylene and CO gas mixtures through the vector of 16 sensor inputs.
- *HHAR: Heterogeneous human activity recognition..* The previous three tasks are all regression tasks, but this one is a classification task. Heterogeneous means that we are testing on a new user who has not appeared in the training set. This dataset contains readings from two motion sensors (accelerometer and gyroscope). Readings are recorded when users execute activities scripted in no specific order, while carrying

[1] https://archive.ics.uci.edu/ml/datasets/Cuff-Less+Blood+Pressure+Estimation
[2] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
[3] https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+under+dynamic+gas+mixtures

smartwatches and smartphones. The dataset contains 9 users, 6 activities (biking, sitting, standing, walking, climbStairup, and climbStair-down), and 6 types of mobile devices [47] [4]. We segment raw measurements into 5-second samples and take Fourier transform on these samples as the input data. For future extension to RNNs as discussed in Section 5, each sample is further divided into time intervals of length $\tau = 0.25$s. Then we calculate the frequency response of sensors for each time interval. The output of HHAR is one of the 6 activities.

4.3 Baseline algorithms

We compare RDeepSense with other two state-of-the-algorithm deep learning uncertainty estimation algorithms, RDeepSense with Monte Carlo estimation, and Gaussian process. The algorithms with deep neural network, including RDeepSense, use the same neural network architecture. It is a 4-layer fully-connected neural network with 500 hidden dimension.

- *MCDrop:* This algorithm is based on the Monte Carlo dropout as described in Section 3.2.1 [17]. Compared with RDeepSense, the main difference is that MCDrop is not optimized by a proper scoring rule. MCDrop requires running the neural network for multiple times to generate samples during uncertainty estimation. Therefore we use MCDrop-k to represent MCDrop with k samples. Multiple samples, *i.e.*, $k > 1$, are required to generate a predictive uncertainty estimation. During the evaluation, we let k to be 3, 5, 10, and 20 to evaluate the tradeoff between the quality of uncertainty estimation and the resource consumption for MCDrop.
- *SSP:* This algorithm trains the neural network with proper scoring methods and uses the ensemble method [34]. Compared with RDeepSense, the main difference is that SSP uses the ensemble method instead of the dropout operation in each layer. SSP requires training multiple neural networks for ensemble. Therefore we use SSP-k to represent SSP by ensemble k individual neural networks. During the evaluation, we let k to be 1, 3, 5, and 10 to evaluate the tradeoff between the quality of uncertainty estimation and the resource consumption for SSP.
- *RDeepSense-MC:* This algorithm is basically the proposed RDeepSense algorithm. The difference is that, during the inference, RDeepSense-MC uses Monte Carlo estimation (10) instead of the efficient approximation (11) for uncertainty estimation. Therefore we use RDeepSense-MCk to present RDeepSense-MC with k samples. During the evaluation, we let k to be 3, 5, 10, and 20 to evaluate the effectiveness and efficiency of RDeepSense inference approximation (11) compared with the Monte Carlo estimation (10).
- *GP:* Gaussian process (GP) is the baseline algorithm used during the evaluation of accuracy and the quality of uncertainty estimations, but not for the evaluations of running time and energy consumption on Edison. The main reason is that the computation cost during model inference for GP is $O(N^3)$, where N is the number of data instances. This cost can be prohibitive even for moderately sized datasets on embedded devices, such as Intel Edison. In additional, GP requires $O(N^2)$ memory consumption during training. Therefore we train the GP with only a proportion of dataset on a server with 128GB memory. Notice that GP is the baseline used to illustrate the quality of uncertainty estimations generated by a statistical model, so the size of training dataset is not the main concern.

4.4 Accuracy of prediction and quality of uncertainty estimations

In this section, we discuss the accuracy and the uncertainty estimation quality of RDeepSense compared with the other baseline algorithms. RDeepSense is tuned with the validating dataset, and all algorithms in all experiments are tested on the testing dataset.

[4] https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition

For three regression problems, two types of evaluation results will be illustrated and discussed. The first type of evaluation is based on some basic measurements including mean absolute error and negative log-likelihood. The second type of evaluation is based on the calibration curves, also known as reliability diagrams. We compute the $z\%$ confidence interval for each testing data based on predictive mean and variance of each algorithm. Then we measure the fraction of the testing data that falls into this confidence interval. For a well-calibrated uncertainty estimation, the fraction of testing data that falls into the confidence interval should be similar to $z\%$. We compute the calibration curves with $z = [10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 85\%, 95\%, 99\%, 99.5\%, 99.9\%]$ for all three regression problems.

For the classification problem, the calibration curve is not available. Therefore, we evaluate HHAR based on accuracy, F1 Score, negative log-likelihood, and a new measurement called the mean entropy of false predictions. If the entropies of false predictions are higher, the learning algorithms show more uncertainties about the false predictions, which represents a better quality of uncertainty estimations.

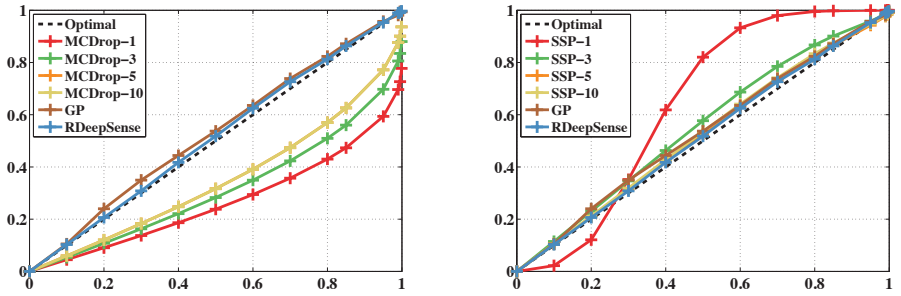
Table 3. Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) for the BPEst task. Except for RDeepSense-MC20, RDeepSense is the best-performing algorithm for NLL and is the second best-performing algorithm for MAE.

	RDeepSense	RDeepSense-MC3	RDeepSense-MC5	RDeepSense-MC10	RDeepSense-MC20
MAE	**14.18**	14.93	14.64	14.44	14.32
NLL	**3.46**	3.49	3.47	3.46	**3.45**
	SSP-1	SSP-3	SSP-5	SSP-10	GP
MAE	15.76	14.68	14.67	14.78	19.15
NLL	4.4	3.69	3.48	3.49	3.59
	MCDrop-3	MCDrop-5	MCDrop-10	MCDrop-20	
MAE	14.80	14.41	**14.09**	**14.09**	
NLL	38.1	5.28	4.00	4.00	

4.4.1 BPEst. We first compare RDeepSense with four baseline algorithms based on mean absolute error (MAE) and negative log-likelihood (NLL), which is illustrated in Table 3, where we highlight the results of RDeepSense and the best-performing one.

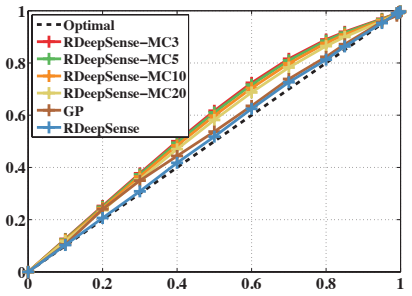
From Table 3, we can see that, except for RDeepSense-MC20, RDeepSense is the best-performing and the second best-performing algorithm for NLL and MAE respectively, which means that RDeepSense can provide accurate estimation with high-quality predictive uncertainty. RDeepSense-MC20 only slightly beats RDeepSense on NLL, however RDeepSense-MC20 consumes around $\times 20$ time and energy compared with RDeepSense. The performance of MCDrop-k increases when k increases. Larger k means that MCDrop algorithm generates more samples during model inference, which can provide higher-quality estimations but more resource consumptions. MCDrop-3 provides a relatively bad result for NLL, which means MCDrop does require a number of samples for uncertainty estimation with reasonable quality. The ensemble method used in SSP increases the prediction performance, but it is not consistent. SSP-10 observes the performance degradation compared with SSP-5. GP obtains a relatively large MAE. This is because GP cannot be scaled to train on the whole dataset.

The calibration curves of BPEst task is illustrated in Figure 1. These three figures show the quality of predictive uncertainty estimations. RDeepSense generates predictive uncertainties with the highest quality. RDeepSense even slightly out-performs the traditional statistical model, GP. As we mentioned in Section 3.2, MCDrop-k tends to underestimate the predictive uncertainty, while SSP-k tends to overestimate the predictive uncertainty. RDeepSense even generates predictive uncertainty with better calibration compared with RDeepSense-MCk, which indicate the effectiveness of approximation during inference. All MCDrop-k, SSP-k, and RDeepSense-MCk improve the quality of uncertainty estimations by increasing the value of k.



(a) The calibration curves of RDeepSense, GP, and MCDrop-k.

(b) The calibration curves of RDeepSense, GP, and SSP-k.



(c) The calibration curves of RDeepSense, GP, and RDeepSense-MCk.

Fig. 1. The calibration curves of BPEst for RDeepSense, GP, MCDrop-k, SSP-k, and RDeepSense-MCk. MCDrop-k underestimates the predictive distribution. SSP-k overestimates the predictive distribution. RDeepSense is the closest curve to the optimal predictive distribution.

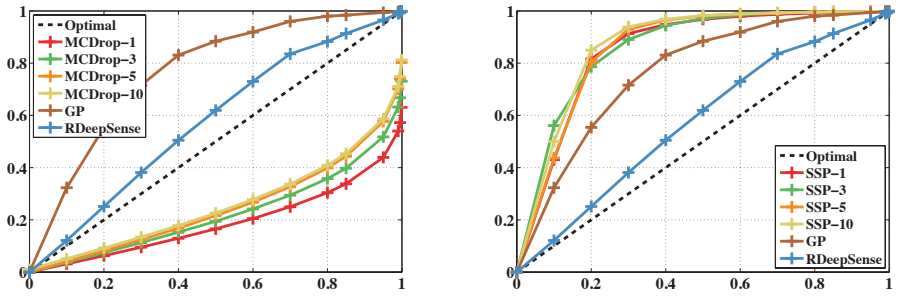
4.4.2 NYCommute. Then we compare RDeepSene with baseline algorithms for NYCommute task. The comparison based on Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) is shown in Table 4.

In this task, RDeepSense tends to find a balance between MAE and NLL measurements. MCDrop-k shows low MAE and high NLL, while SSP-k shows high MAE and low NLL. MCDrop-k tries to minimize the mean square error, while SSP-k tries to minimize the negative log-likelihood. Therefore, MCDrop-k focuses more on the mean of predictive distribution, and SSP-k focuses more on the overall likelihood. RDeepSense combines two objective functions, mean square error and negative log-likelihood, which tries to find a balance point between these two. Still, due to the scalability problem, GP obtains a relatively larger MAE. Compared with RDeepSense-MCk, RDeepSense achieve a good performance on both MAE and NLL. Only RDeepSense-MC20 shows the same performance on the NLL measurement.

The calibration curves of NYCommute task is illustrated in Figure 2. Both MCDrop-k and SSP-k fail to generate high-quality uncertainty estimations by either underestimating or overestimating the predictive uncertainties. However, RDeepSense can still provide uncertainty estimations with good quality, which outperforms GP with a significant margin. Compared with RDeepSense-MCk, RDeepSense shows similar performance on generating well-calibrated predictive uncertainties, which shows that the approximation (11) works well in practice.

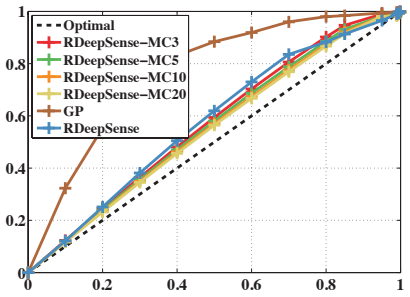
Table 4. Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) for the NYCommute task.

	RDeepSense	RDeepSense-MC3	RDeepSense-MC5	RDeepSense-MC10	RDeepSense-MC20
MAE	**5.64**	6.10	6.04	5.99	5.96
NLL	7.7	7.85	7.81	7.73	7.7
	SSP-1	SSP-3	SSP-5	SSP-10	GP
MAE	8.15	7.90	7.51	7.03	11.84
NLL	4.86	**4.67**	4.84	4.81	7.46
	MCDrop-3	MCDrop-5	MCDrop-10	MCDrop-20	
MAE	5.69	5.64	**5.61**	**5.61**	
NLL	19995.6	1335.73	640.35	640.35	



(a) The calibration curves of RDeepSense, GP, and MCDrop-k.

(b) The calibration curves of RDeepSense, GP, and SSP-k.



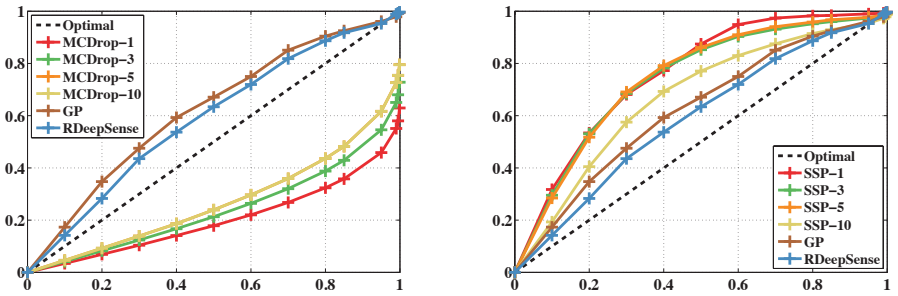
(c) The calibration curves of RDeepSense, GP, and RDeepSense-MCk.

Fig. 2. The calibration curves of NYCommute for RDeepSense, GP, MCDrop-k, SSP-k, and RDeepSense-MCk. MCDrop-k highly underestimates the predictive distribution. SSP-k highly overestimates the predictive distribution. RDeepSense makes a tradeoff between these two and is the closest curve to the optimal predictive distribution.

4.4.3 GasSen. Next we compare RDeepSense with other baseline algorithms for the GasSen task. Table 5 illustrates the performance of all these algorithms based on Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL). Except for RDeepSense-MC20, RDeepSense is the best-performing algorithm according to these

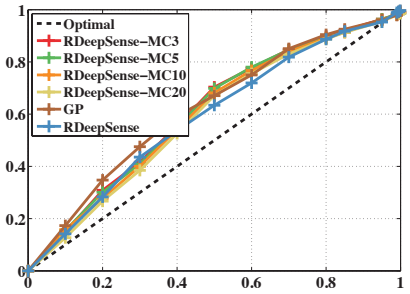
Table 5. Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) for the GasSen task. Except for RDeepSense-MC20, RDeepSense is the best-performing algorithm for both MAE and NLL.

	RDeepSense	RDeepSense-MC3	RDeepSense-MC5	RDeepSense-MC10	RDeepSense-MC20
MAE	**15.25**	17.21	16.44	16.34	15.61
NLL	**3.77**	4.23	4.18	3.88	**3.73**
	SSP-1	SSP-3	SSP-5	SSP-10	GP
MAE	24.40	22.53	20.75	20.68	35.74
NLL	4.76	4.34	3.92	3.81	7.76
	MCDrop-3	MCDrop-5	MCDrop-10	MCDrop-20	
MAE	21.23	20.45	19.79	19.79	
NLL	2201.95	463.94	170.45	170.45	



(a) The calibration curves of RDeepSense, GP, and MCDrop-k.

(b) The calibration curves of RDeepSense, GP, and SSP-k.



(c) The calibration curves of RDeepSense, GP, and RDeepSense-MCk.

Fig. 3. The calibration curves of GasSen for RDeepSense, GP, MCDrop-k, SSP-k, and RDeepSense-MCk. MCDrop-k highly underestimates the predictive distribution. SSP-k highly overestimates the predictive distribution. RDeepSense is the closest curve to the optimal predictive distribution.

two metrics. Similarly, MCDrop-k shows low MAE and NLL, while SSP-k shows high MAE and NLL. This is due to the objective of these two types of algorithms. MCDrop-k minimizes the mean square error, while SSP-k minimizes

the negative log-likelihood. Therefore, MCDrop-k focuses more on the mean of predictive distribution, and SSP-k focuses more on the overall likelihood. RDeepSense combines two objective function. Therefore, RDeepSense is able to achieve the best performance in both cases. The usage of dropout that prevents feature co-adapting is the main reason why RDeepSense achieves better NLL compared with SPP-k. The RDeepSense still achieves good performance compared with its Motel Carlo version. Only RDeepSense-MC20 slightly outperforms RDeepSense under the NLL measurement, which shows the effectiveness of the approximation used in RDeepSense.

The calibration curves of GasSen task is illustrated in Figure 3. The calibration curves of MCDrop-k highly underestimates the predictive distribution as shown in Figure 3a, while the calibration curves of SSP-k highly overestimates the predictive distribution as shown in Figure 3b. Although there exists a bit deviation for RDeepSense compared with the optimal calibration curve, RDeepSense greatly reduces the effect of underestimation and overestimation, and slightly outperforms the traditional statistical model, GP. Compared with unbiased RDeepSense-MCk, RDeepSense shows the similar performance. However, RDeepSense saves save a great amount of energy and time consumption as we will discuss in Section 4.5.

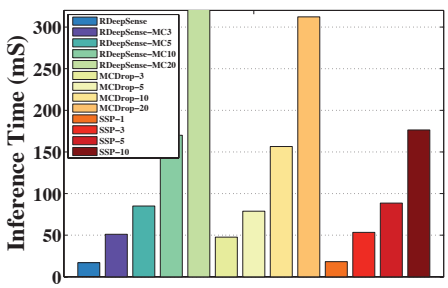
4.4.4 HHAR. Last we compare RDeepSense with the other baseline algorithm for the HHAR task. Table 6 illustrates the performance metrics of all algorithms based on Accuracy (Acc), F1 Score (F1 Score), Negative Log-Likelihood (NLL), and Mean Entropy of False Predictions (MEFP).

Table 6. Accuracy (Acc), Negative Log-Likelihood (NLL), Mean Entropy of False Predictions (MEFP) for the HHAR task. RDeepSense is the best-performing algorithm according to all measures.

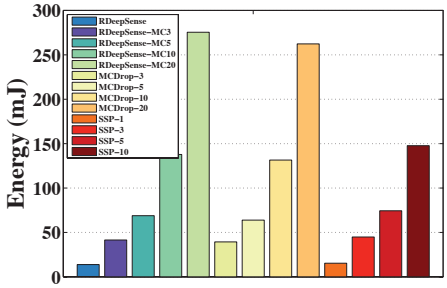
	RDeepSense	RDeepSense-MC3	RDeepSense-MC5	RDeepSense-MC10	RDeepSense-MC20
Acc	**83.98%**	80.66%	83.07%	83.08%	83.85%
F1 Score	**0.670**	0.601	0.638	0.668	**0.671**
NLL	**0.161**	0.193	0.188	0.172	**0.159**
MEFP	**1.715**	1.604	1.621	1.626	1.628
	SSP-1	SSP-3	SSP-5	SSP-10	GP
Acc	77.15%	78.34%	79.30%	80.30%	77.29%
F1 Score	0.650	0.652	0.657	0.661	0.659
NLL	1.138	1.188	1.165	1.214	0.807
MEFP	1.619	1.629	1.672	1.708	1.218
	MCDrop-3	MCDrop-5	MCDrop-10	MCDrop-20	
Acc	79.53%	79.73%	79.73%	80.51%	
F1 Score	0.586	0.589	0.589	0.593	
NLL	0.166	0.163	0.162	**0.161**	
MEFP	0.501	0.548	0.574	0.579	

Except for RDeepSense-MC20, RDeepSense is the best-performing algorithm according to all measures, which means RDeepSense can provide both high prediction accuracy as well as high quality of uncertainty estimations. MCDrop-k algorithms are trained with log-likelihood. Therefore they try to minimize the negative log-likelihood, but they are over-confident about their prediction even when they make some wrong predictions according to the MEFP measure. SSP-k algorithms are trained with Brier score. Therefore they fall short to achieve smaller NLL values. Compared with RDeepSense-MCk algorithms, RDeepSense still provides a good performance in all measurements. Only RDeepSense-MC20 shows a superior performance on F1 Score and NLL measurements.

4.5 Inference time and energy consumption

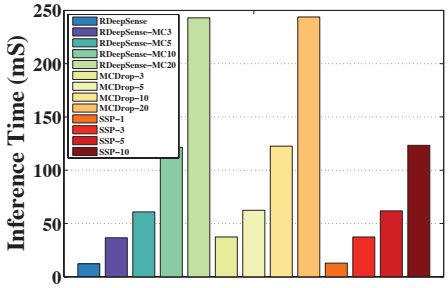


(a) The inference time of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for BPEst.

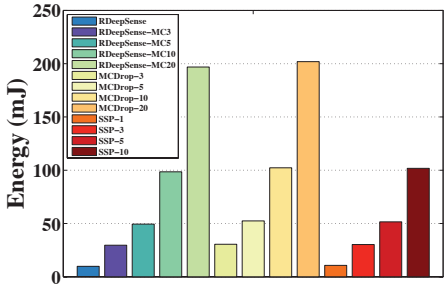


(b) The energy consumption of RDeepSense, RDeepSense-MCk,MCDrop-k, and SSP-k for BPEst.

Fig. 4. The inference time and energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for BPEst.

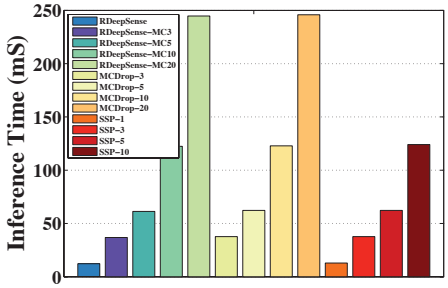


(a) The inference time of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for NYCommute.

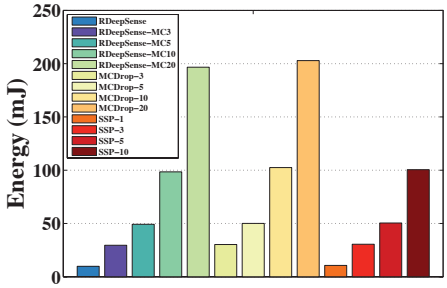


(b) The energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for NYCommute.

Fig. 5. The inference time and energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for NYCommute.

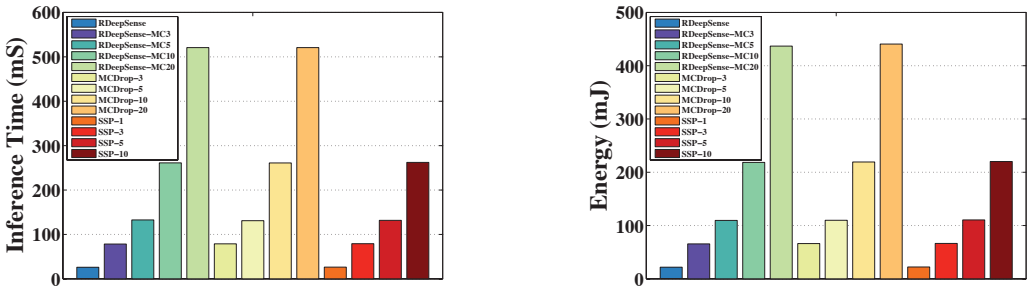


(a) The inference time of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for GasSen.



(b) The energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for GasSen.

Fig. 6. The inference time and energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for GasSen.



(a) The inference time of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for HHAR.

(b) The energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for HHAR.

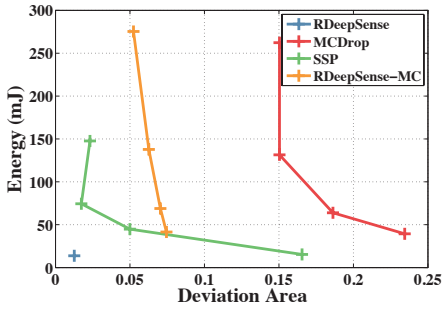
Fig. 7. The inference time and energy consumption of RDeepSense, RDeepSense-MCk, MCDrop-k, and SSP-k for HHAR.

We compared the resource consumption of each algorithm including inference time and energy consumption of one-data-sample execution, which are two key issues for mobile and ubiquitous computing. All the experiments are conducted on Intel Edison with only CPU as the computing unit. No further optimization is made on any algorithms. The inference time and energy consumption of GP are not included. This is because the time complexity of GP is $O(N^3)$, where N is the size of training dataset, which is infeasible for embedded devices such as Intel Edison. The results of four tasks, *i.e.*, BPEst, NYCommute, GasSen, and HHAR, are illustrated in Figures 4, 5, 6, and 7 respectively.

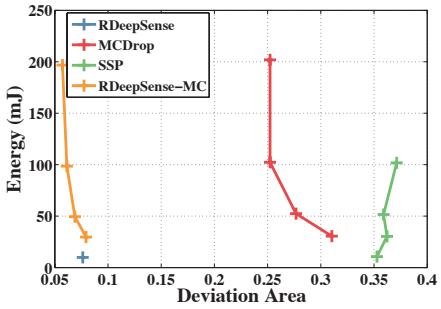
We can clearly see that RDeepSense greatly reduces the inference time and energy consumption compared with the other deep learning uncertainty estimation algorithms. Compared with MCDrop algorithm, RDeepSense is trained according to the proper scoring rule, which can directly output the predictive distribution instead of using sampling methods. Compared with SSP algorithm, RDeepSense uses dropout regularization as an implicit ensemble method, which avoids running multiple deep learning models during model inference on embedded devices. Compared with RDeepSense-MC, RDeepSense use the approximation (11) to replace the computationally intensive Motel Carlo method (10) during the inference.

We further analyze the relationship between energy consumption and the quality of uncertainty estimation for each algorithms. For regression problems, we use the area between the calibration curve of an algorithm and the optimal calibration curve, called deviation area, as the quality measurement of uncertainty. The smaller deviation area is, the better quality of uncertainty the algorithm estimates. When the calibration curve of an algorithm is optimal, the deviation area is 0. For classification problems, we use the negative mean entropy of false predictions as the quality measurement of uncertainty. Smaller negative mean entropy of false predictions means is that the algorithm is more uncertain about their false predictions. The result is shown in Figure 8.

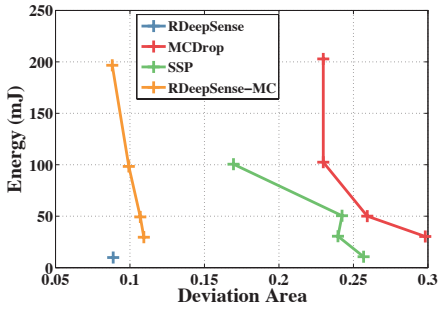
The point or line stay in the bottom-left corner of the graph represents a better tradeoff between energy and uncertainty quality, *i.e.*, using less energy to obtain better uncertainty estimations. Therefore, RDeepSense is the best-performing algorithm that uses the least amount of energy to obtain the best uncertainty estimation quality. RDeepSense-MC can achieve similar uncertainty estimation quality as RDeepSense, however it requires much more energy consumption. The results show that RDeepSense is an effective and efficient uncertainty estimation algorithm (11) compared with its Monte Carlo version (10). Other two baseline algorithms, MCDrop and SSP, usually suffer a large deviation area or become over-confidence about their false predictions while using more energy for computation. Figure 8 shows that RDeepSense is the most suitable algorithm for generate predictive uncertainty estimations for mobile and ubiquitous computing application on embedded devices.



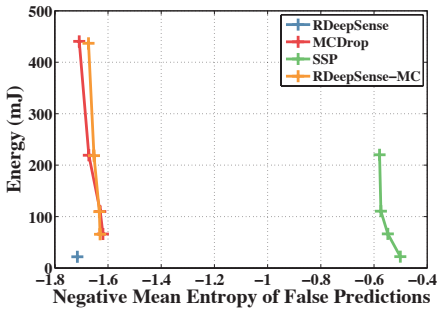
(a) The relationship between deviation area and energy consumption for BPEst.



(b) The relationship between deviation area and energy consumption for NYCommute.



(c) The relationship between deviation area and energy consumption for GasSen.



(d) The relationship between negative mean entropy of false predictions and energy consumption for HHAR.

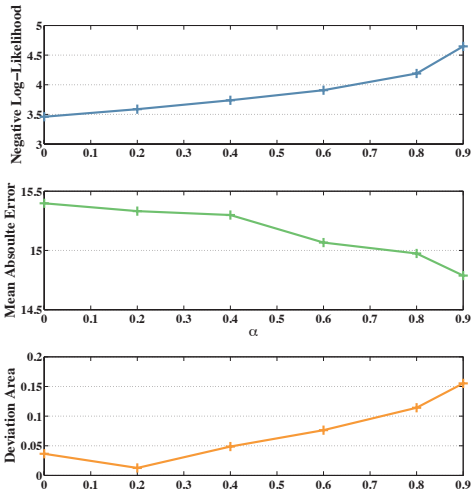
Fig. 8. The relationship between deviation area/negative mean entropy of false predictions and energy consumption of all algorithms. RDeepSense (in the bottom-left corner) is the best-performing algorithm that uses the least energy to achieve the best uncertainty estimation quality

4.6 Effect of hyper-parameter α on model performance

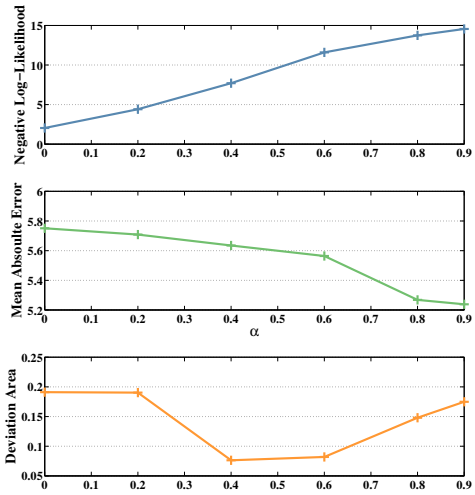
The hyper-parameter α controls the tradeoff between optimization of mean and variance within the training objective function (6) that can help to obtain a well-calibrated uncertainty estimation. In this subsection, we evaluate the functionality of α and also shed light on the way of tuning α.

For each task, we train RDeepSense with $\alpha = [0, 0.2, 0.4, 0.6, 0.8, 0.9]$. When $\alpha = 0.0$, RDeepSense is trained by minimizing the negative log-likelihood. When we increase the value of α, RDeepSense focuses more on the mean value estimation instead of the negative log-likelihood. In order to show the effect of the choice of α on the quality of predictive uncertainty estimation, we show the negative log-likelihood and devision area (the area between the calibration curve of an algorithm and the optimal calibration curve) for regression tasks and show the negative log-likelihood and Negative Mean Entropy (NME) of false predictions for the classification task in Figure 9.

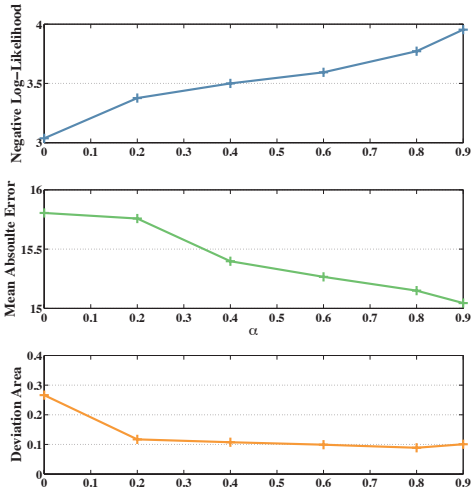
A good uncertainty estimation should faithfully reflect the probability that prediction will happen. Therefore, RDeepSense targets on a well-calibrated uncertainty estimation, such as the prediction with low devision area, in stead of the prediction with low negative log-likelihood. From Figure 9a, 9b, and 9c, we can see that hyper-parameter α controls the tradeoff between optimization mean and variance within the training objective



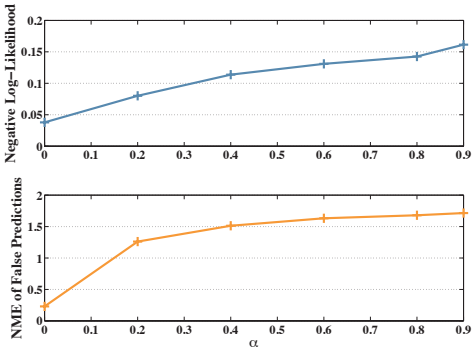
(a) Negative Log-Likelihood, Mean Absolute Error, and Deviation Area with different selections of α for BPEst.



(b) Negative Log-Likelihood, Mean Absolute Error, and Deviation Area with different selections of α for NY-Commute.



(c) Negative Log-Likelihood, Mean Absolute Error, and Deviation Area with different selections of α for GasSen.



(d) Negative Log-Likelihood and Negative Mean Entropy (NME) of false predictions with different selections of α for HHAR.

Fig. 9. Negative Log-Likelihood, Mean Absolute Error, and Deviation Area/Negative Mean Entropy (NME) of false predictions with different selections of α for four tasks.

function (6). Smaller α tends to reduce negative log-likelihood by increasing the predictive variance, which tends to result the overestimation of predictive uncertainties. Larger α tends to reduce negative log-likelihood by predicting a better mean value, which tends to result the underestimation of predictive uncertainties. When tuning the hyper-parameter α, we can easily found a point that achieve the smallest devision area by grid searching α from 0 to 1. At the same time, it is not surprising that increasing α can slightly increase the negative log-likelihood, since $\alpha = 0$ represents regarding negative log-likelihood as the objective function. In addition, Figure 9d shows that increasing α can consistently increase the negative mean entropy of false predictions.

5 DISCUSSION

This paper focuses on empowering neural networks to generate high-quality predictive uncertainty estimations in a theoretically-grounded and energy-efficient manner for mobile and ubiquitous computing tasks. Currently, RDeepSense can only support fully-connected neural networks. It is possible to extend the two-step solution introduced in Section 3 to convolutional and recurrent neural networks by replacing the original dropout operation with convolutional dropout [16] and recurrent dropout [18]. These two dropout operations can convert convolutional neural networks and recurrent neural networks into Bayesian neural networks [16, 18], but additional efforts are needed to 1) theoretically prove that the extended two-step solution can equate an arbitrary neural network with a statistical model, and 2) empirically show that the extended two-step solution can provide high-quality uncertainty estimations on the real datasets.

Another interesting extension could be empowering existing neural networks within mobile and ubiquitous computing applications to generate predictive uncertainty estimations without additional training. A lot of neural networks have already been trained with dropout operations. As shown by Gal et al. [17], although these models tend to underestimate the true uncertainties, they can provide uncertainty estimations during model inference. This can be a good solution for mobile and ubiquitous computing applications that want to obtain an indictor of predictive uncertainty instead of a high-quality predictive uncertainty estimation without retraining their neural networks. However, additional efforts are needed to bypass the Monte Carlo sampling method and provide an energy-efficient method for generating uncertainty estimations on embedded devices.

In addition, for classification problems, although traditional neural networks can also output predictive distribution on each class, which contains predictive uncertainties, RDeepSense provides a high-quality predictive distribution on each class and has been proved to be equivalent to a statistical model.

6 CONCLUSION

We introduced RDeepSense, a simple yet effective solution that empowers fully-connected neural networks to generate well-calibrated predictive uncertainty estimations during model inference. RDeepSense is a computationally efficient algorithm that can provide predictive uncertainty estimations in mobile and ubiquitous computing applications with almost no additional overhead. Theoretical analysis also shows the equivalence between RDeepSense and a statistical model. We evaluated RDeepSense on four mobile and ubiquitous computing tasks, where RDeepSense outperformed the state-of-the-art baselines by significant margins on the quality of uncertainty estimations while still consuming the least amount of energy on embedded devices. In summary, RDeepSense is a simple, effective, and efficient solution for mobile and ubiquitous applications to build reliable neural networks with uncertainty estimations.

ACKNOWLEDGMENTS

Research reported in this paper was sponsored in part by the U.S. Army Research Laboratory and was accomplished under Cooperative Agreements W911NF-17-2-0196 and W911NF-09-2-0053, DARPA contract W911NF-17-C-0099, and NSF grants CNS 13-29886 and CNS 16-18627. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied,

of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

A THEORETICAL ANALYSIS: THE EQUIVALENCE BETWEEN RDEEPSENSE AND STATISTICAL MODELS

A.1 Dropout with mean square error

Assume that we have N pairs of training data, which can be formed into the input matrix $\mathbf{X} \in \mathbb{R}^{N \times d^{(0)}}$ and the corresponding output matrix $\mathbf{Y} \in \mathbb{R}^{N \times d^{(L)}}$. For the regression problem, we place a joint Gaussian distribution over all function values

$$\begin{aligned} p(\mathbf{F}|\mathbf{X}) &\sim \mathcal{N}(0, K(\mathbf{X}, \mathbf{X})), \\ p(\mathbf{Y}|\mathbf{F}) &\sim \mathcal{N}(F, \tau^{-1}\mathbf{I}). \end{aligned} \tag{12}$$

where τ is the precision hyper-parameter and $K(\cdot, \cdot)$ is the covariance function, encoding the prior function distribution of the Gaussian process. With a dataset of N samples, $K(\cdot, \cdot)$ is a $N \times N$ matrix.

To formulate a fully-connected neural network as a Gaussian process, for a single fully-connected layer in a Bayesian neural network, we can define the covariance function as

$$K(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{W}^{(l)}) f^{(l)}(\mathbf{x}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}) f^{(l)}(\mathbf{x}'\mathbf{W}^{(l)} + \mathbf{b}^{(l)}) d\mathbf{W}^{(l)}, \tag{13}$$

where $p(\mathbf{W}^{(l)}) = \mathcal{N}(0, l^{-2}\mathbf{I})$ and $f^{(l)}(\cdot)$ is the nonlinear activation function. For an L-layer fully-connected neural network, we can feed the output of one Gaussian process to the covariance of the next as a deep Gaussian process model [12]. Then our final target, predictive distribution estimation, can be formulated as

$$p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}|\mathbf{x}, \mathcal{W}) p(\mathcal{W}|\mathbf{X}, \mathbf{Y}) d\mathcal{W}, \tag{14}$$

where $p(\mathbf{y}|\mathbf{x}, \mathcal{W})$ is the whole Bayesian neural network with random variables $\mathcal{W} = \{p(\mathbf{W}^{(l)})\}$.

However, calculating the predictive distribution estimation $p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y})$ requires the posterior distribution $p(\mathcal{W}|\mathbf{X}, \mathbf{Y})$, and calculating the posterior distribution $p(\mathcal{W}|\mathbf{X}, \mathbf{Y})$ further requires calculating the inverse of an $N \times N$ matrix, which is infeasible for a large-scale dataset used by a deep neural network. Therefore, a variational distribution $q(\mathcal{W}) = \prod_{l=1}^{L} p(\tilde{\mathbf{W}}^{(l)})$ is proposed to approximate the true posterior distribution, where $\tilde{\mathbf{W}}^{(l)}$ is the random variable used in dropout operations introduced in (2).

Then we minimize the KL divergence between the approximated posterior $q(\mathcal{W})$ and the posterior of the deep Gaussian process over the variational parameters $\{\tilde{\mathbf{W}}^{(l)}\}$. The minimization objective is the negative log evidence lower bound derived from the likelihood,

$$\mathcal{L}_{gp} = -\int q(\mathcal{W}) \log p(\mathbf{Y}|\mathbf{X}, \mathcal{W}) d\mathcal{W} + KL(q(\mathcal{W})||p(\mathcal{W})). \tag{15}$$

We can use Monte Carlo sampling to approximate the first integral in (15), and (15) can be reduced to

$$\mathcal{L}_{gp} = \sum_{n=1}^{N} (\mathbf{y}_n - \hat{\mathbf{y}}_n)^2 + \frac{p_i l^2}{2\tau N} \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_2^2, \tag{16}$$

where p_i is the dropout probability in (2), τ is the hyperparameter in (12), and l is the length-scale used to define the prior distribution $p(\mathbf{W}^{(l)})$.

If we compare (16) with the first equations in (4) and (5), we can find that optimizing a variational approximation of deep Gaussian process is equivalent to optimizing an dropout neural network based on mean square error as the loss function.

A.2 Dropout with negative log-likelihood

We have already shown that the equivalence between dropout training and deep Gaussian process with variational approximation. In order to further formulate a fully-connected neural network with log-likelihood as a statistical model, we adds an additional generative step to deep Gaussian process that converts (12) into a new statistical model,

$$\begin{aligned} p(\mathbf{F}|\mathbf{X}) &\sim \mathcal{N}(0, K(\mathbf{X},\mathbf{X})), \\ p(\mathbf{Z}|\mathbf{F}) &\sim \mathcal{N}(F, \tau^{-1}\mathbf{I}), \\ p(\mathbf{Y}|\mathbf{Z}) &\sim g(\mathbf{Y};\mathbf{Z}), \end{aligned} \tag{17}$$

where $g(\mathbf{Y};\mathbf{Z})$ is a distribution that converts latent Gaussian process into predictive distribution that conforms the proper scoring rule, *i.e.*, log-likelihood.

For regression problems, $p(\mathbf{Y}|\mathbf{Z})$ is the Gaussian distribution,

$$\begin{aligned} \mathbf{Z} &= [\mathbf{Z}_\mu, \mathbf{Z}_{\sigma^2}], \\ p(\mathbf{Y}|\mathbf{Z}) &\sim \mathcal{N}(\mathbf{Z}_\mu, \mathbf{Z}_{\sigma^2}). \end{aligned} \tag{18}$$

For classification problems, $p(\mathbf{Y}|\mathbf{Z})$ is the composition of categorical distribution with softmax function

$$p(\mathbf{Y}_{nk}|\mathbf{Z}_{n\cdot}) \sim \frac{\exp(Z_{nk})}{\sum_{k'}\exp(Z_{nk'})}. \tag{19}$$

Therefore, the final predictive distribution estimation is changed from (14) into

$$p(\mathbf{y}|\mathbf{x},\mathbf{X},\mathbf{Y}) = \int p(\mathbf{y}|\mathbf{z})\Big(\int p(\mathbf{z}|\mathbf{x},\mathcal{W})p(\mathcal{W}|\mathbf{X},\mathbf{Y})d\mathcal{W}\Big)d\mathbf{z}. \tag{20}$$

In order to calculate the predictive probability (20), we still have to propose the same variational distribution $q(\mathcal{W}) = \prod_{l=1}^{L} p(\tilde{\mathbf{W}}^{(l)})$ to approximate the posterior distribution $p(\mathcal{W}|\mathbf{X},\mathbf{Y})$, where $\tilde{\mathbf{W}}^{(l)}$ is the random variable used in dropout operations introduced in (2).

Then, in order to optimize over the variational distribution, the log evidence lower bound for the likelihood can be derived from the likelihood function,

$$\begin{aligned} &\log p(\mathbf{Y}|\mathbf{X}) \\ &= \log \int p(\mathbf{Y}|\mathbf{Z})p(\mathbf{Z}|\mathbf{X},\mathcal{W})p(\mathcal{W})d\mathcal{W}d\mathbf{Z} \\ &= \log \int q(\mathcal{W})p(\mathbf{Y}|\mathbf{Z})p(\mathbf{Z}|\mathbf{X},\mathcal{W})\frac{p(\mathcal{W})}{q(\mathcal{W})}d\mathcal{W}d\mathbf{Z} \\ &\geq \int q(\mathcal{W})p(\mathbf{Z}|\mathbf{X},\mathcal{W})\log\left(p(\mathbf{Y}|\mathbf{Z})\frac{p(\mathcal{W})}{q(\mathcal{W})}\right)d\mathcal{W}d\mathbf{Z} \\ &= \int q(\mathcal{W})p(\mathbf{Z}|\mathbf{X},\mathcal{W})\log p(\mathbf{Y}|\mathbf{Z})d\mathcal{W}d\mathbf{Z} - KL\Big(q(\mathcal{W})||p(\mathcal{W})\Big). \end{aligned} \tag{21}$$

Therefore we minimize the negative log evidence lower bound derived in (21) to optimize the variational parameters $\{\tilde{\mathbf{W}}^{(l)}\}$,

$$\mathcal{L}_{lgp} = -\sum_{n=1}^{N}\int p(\mathbf{z}_n|\mathbf{x}_n,\mathcal{W})\cdot\log p(\mathbf{y}_n|\mathbf{z}_n)d\mathcal{W}d\mathbf{Z} + KL\Big(q(\mathcal{W})||p(\mathcal{W})\Big). \tag{22}$$

The first integral in (22) can be approximated with Monte Carlo integration and the second term can be approximated according to MCDrop [17],

$$\mathcal{L}_{lgp_{mc}} = -\sum_{n=1}^{N} \log p\Big(\mathbf{y}_n|\hat{\mathbf{z}}_n(\mathbf{x}_n, \hat{\mathcal{W}})\Big) + \frac{p_i l^2}{2\tau N} \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_2^2. \tag{23}$$

Then it is trivial to verify that (23) is equivalent to the second equation in (4) and (5) for regression and classification problems respectively by substituting $p(\mathbf{y}_n|\hat{\mathbf{z}}_n(\mathbf{x}_n, \hat{\mathcal{W}}))$ with (18) or (19).

Now, we have shown that training a fully-connected neural network with dropout and negative log-likelihood loss function is equivalent to a Gaussian or categorical distribution based on the latent deep Gaussian process.

REFERENCES

[1] Intel edison compute module. http://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison-module_HG_331189.pdf.

[2] P. Baldi and P. J. Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

[3] J. S. Bauer, S. Consolvo, B. Greenstein, J. Schooler, E. Wu, N. F. Watson, and J. Kientz. Shuteye: encouraging awareness of healthy sleep recommendations with a mobile, peripheral display. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1401–1410. ACM, 2012.

[4] P. Baumann, M. Langheinrich, A. Dey, and S. Santini. Quantifying the uncertainty of next-place predictions. In *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, pages 74–85. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[5] F. R. Bentley, Y.-Y. Chen, and C. Holz. Reducing the stress of coordination: sharing travel time information between contacts on mobile phones. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 967–970. ACM, 2015.

[6] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, (just-accepted), 2017.

[7] N. Boukhelifa, M.-E. Perrin, S. Huron, and J. Eagan. How data workers cope with uncertainty: A task characterisation study. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3645–3656. ACM, 2017.

[8] K.-Y. Chen, D. Ashbrook, M. Goel, S.-H. Lee, and S. Patel. Airlink: sharing files between multiple devices using in-air gestures. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 565–569. ACM, 2014.

[9] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. LeGrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, et al. The mobile sensing platform: An embedded activity recognition system. *IEEE Pervasive Computing*, 7(2), 2008.

[10] J. Chung, M. Donahoe, C. Schmandt, I.-J. Kim, P. Razavai, and M. Wiseman. Indoor location sensing using geo-magnetism. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 141–154. ACM, 2011.

[11] M. Clyde and E. I. George. Model uncertainty. *Statistical science*, pages 81–94, 2004.

[12] A. Damianou and N. Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

[13] V. B. E. T. G. A. H. C. Daniel Castro, Steven Hickson and I. Essa. Predicting daily activities from egocentric images using deep learning. *ISWC*, 2015.

[14] M. Faurholt-Jepsen, M. Vinberg, M. Frost, S. Debel, E. Margrethe Christensen, J. E. Bardram, and L. V. Kessing. Behavioral activities collected through smartphones and the association with illness activity in bipolar disorder. *International journal of methods in psychiatric research*, 25(4):309–323, 2016.

[15] J. Fonollosa, S. Sheik, R. Huerta, and S. Marco. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, 215:618–629, 2015.

[16] Y. Gal and Z. Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

[17] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

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

[19] Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[20] T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[21] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet. *Circulation*, 101(23):e215–e220, 2000.

[22] D. Gordon, J. Czerny, T. Miyaki, and M. Beigl. Energy-efficient activity recognition using prediction. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pages 29–36. IEEE, 2012.

[23] E. Griffiths, T. S. Saponas, and A. Brush. Health chair: implicitly sensing heart and respiratory rate. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 661–671. ACM, 2014.
[24] T. Grosse-Puppendahl, X. Dellangnol, C. Hatzfeld, B. Fu, M. Kupnik, A. Kuijper, M. R. Hastall, J. Scott, and M. Gruteser. Platypus: Indoor localization and identification through sensing of electric potential changes in human bodies. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 17–30. ACM, 2016.
[25] Y. Guan and T. Ploetz. Ensembles of deep lstm learners for activity recognition using wearables. *arXiv preprint arXiv:1703.09370*, 2017.
[26] Y. Jiang, X. Pan, K. Li, Q. Lv, R. P. Dick, M. Hannigan, and L. Shang. Ariel: Automatic wi-fi based room fingerprinting for indoor localization. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 441–450. ACM, 2012.
[27] M. Kachuee, M. M. Kiani, H. Mohammadzade, and M. Shabany. Cuff-less high-accuracy calibration-free blood pressure estimation using pulse transit time. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 1006–1009. IEEE, 2015.
[28] S. Kaiser, A. Parks, P. Leopard, C. Albright, J. Carlson, M. Goel, D. Nassehi, and E. C. Larson. Design and learnability of vortex whistles for managing chronic lung function via smartphones. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 569–580. ACM, 2016.
[29] M. Kay, T. Kola, J. R. Hullman, and S. A. Munson. When (ish) is my bus?: User-centered visualizations of uncertainty in everyday, mobile predictive systems. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5092–5103. ACM, 2016.
[30] M. Kay, S. N. Patel, and J. A. Kientz. How good is 85%?: A survey tool to connect classifier evaluation to acceptability of accuracy. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 347–356. ACM, 2015.
[31] C. Koehler, N. Banovic, I. Oakley, J. Mankoff, and A. K. Dey. Indoor-alps: an adaptive indoor location prediction system. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 171–181. ACM, 2014.
[32] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
[33] M. Krzywinski and N. Altman. Points of significance: importance of being uncertain. *Nature methods*, 10(9):809, 2013.
[34] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.
[35] N. D. Lane, P. Georgiev, and L. Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294. ACM, 2015.
[36] B. Y. Lim and A. K. Dey. Investigating intelligibility for uncertain context-aware applications. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 415–424. ACM, 2011.
[37] Z. C. Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.
[38] A. Mannini, S. S. Intille, M. Rosenberger, A. M. Sabatini, and W. Haskell. Activity recognition using a single accelerometer placed at the wrist or ankle. *Medicine and science in sports and exercise*, 45(11):2193, 2013.
[39] P. Melgarejo, X. Zhang, P. Ramanathan, and D. Chu. Leveraging directional antenna capabilities for fine-grained gesture recognition. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 541–551. ACM, 2014.
[40] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, and J. Song. E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 260–273. ACM, 2011.
[41] G. Pirkl and P. Lukowicz. Robust, low cost indoor positioning using magnetic resonant coupling. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 431–440. ACM, 2012.
[42] Q. Pu, S. Gupta, S. Gollakota, and S. Patel. Whole-home gesture recognition using wireless signals. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 27–38. ACM, 2013.
[43] J. Quinonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Schölkopf. Evaluating predictive uncertainty challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 1–27. Springer, 2006.
[44] V. Radu, N. D. Lane, S. Bhattacharya, C. Mascolo, M. K. Marina, and F. Kawsar. Towards multimodal deep learning for activity recognition on mobile devices. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 185–188. ACM, 2016.
[45] C. E. Rasmussen. Gaussian processes for machine learning. 2006.
[46] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
[47] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 127–140. ACM, 2015.
[48] T. Toscos, K. Connelly, and Y. Rogers. Best intentions: health monitoring technology and children. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1431–1440. ACM, 2012.

[49] E. J. Wang, W. Li, D. Hawkins, T. Gernsheimer, C. Norby-Slycord, and S. N. Patel. Hemaapp: noninvasive blood screening of hemoglobin using smartphone cameras. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 593–604. ACM, 2016.

[50] H. Wang, Y.-H. Kuo, D. Kifer, and Z. Li. A simple baseline for travel time estimation using large-scale trip data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 61. ACM, 2016.

[51] J. Weppner, B. Bischke, and P. Lukowicz. Monitoring crowd condition in public spaces by tracking mobile consumer devices with wifi interface. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1363–1371. ACM, 2016.

[52] L. Yao, F. Nie, Q. Z. Sheng, T. Gu, X. Li, and S. Wang. Learning from less for better: semi-supervised activity recognition via shared structure discovery. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 13–24. ACM, 2016.

[53] S. Yao, M. T. Amin, L. Su, S. Hu, S. Li, S. Wang, Y. Zhao, T. Abdelzaher, L. Kaplan, C. Aggarwal, et al. Recursive ground truth estimator for social data streams. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, pages 1–12. IEEE, 2016.

[54] S. Yao, S. Hu, S. Li, Y. Zhao, L. Su, L. Kaplan, A. Yener, and T. Abdelzaher. On source dependency models for reliable social sensing: Algorithms and fundamental error bounds. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 467–476. IEEE, 2016.

[55] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. Deepsense: a unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017.

[56] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017.

[57] H.-S. Yeo, J. Lee, A. Bianchi, and A. Quigley. Watchmi: pressure touch, twist and pan gesture input on unmodified smartwatches. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 394–399. ACM, 2016.

[58] C. Zhang, L. Liu, D. Lei, Q. Yuan, H. Zhuang, T. Hanratty, and J. Han. Triovecevent: Embedding-based online local event detection in geo-tagged tweet streams. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 595–604. ACM, 2017.

[59] C. Zhang, K. Zhang, Q. Yuan, H. Peng, Y. Zheng, T. Hanratty, S. Wang, and J. Han. Regions, periods, activities: Uncovering urban dynamics via cross-modal representation learning. In *Proceedings of the 26th International Conference on World Wide Web*, pages 361–370. International World Wide Web Conferences Steering Committee, 2017.

[60] Y. Zhang, M. K. Chong, J. Müller, A. Bulling, and H. Gellersen. Eye tracking for public displays in the wild. *Personal and Ubiquitous Computing*, 19(5-6):967–981, 2015.

[61] Y. Zhao, S. Li, S. Hu, L. Su, S. Yao, H. Shao, H. Wang, and T. Abdelzaher. Greendrive: A smartphone-based intelligent speed adaptation system with real-time traffic signal prediction. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pages 229–238. ACM, 2017.

[62] Y. Zhao, S. Yao, S. Li, S. Hu, H. Shao, and T. F. Abdelzaher. Vibebin: A vibration-based waste bin level detection system. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):122, 2017.

[63] Y. Zhao, Y. Zhang, T. Yu, T. Liu, X. Wang, X. Tian, and X. Liu. Citydrive: A map-generating and speed-optimizing driving system. In *INFOCOM, 2014 Proceedings IEEE*, pages 1986–1994. IEEE, 2014.

Received May 2017; revised August 2017; accepted October 2017