2018 IEEE 38th International Conference on Distributed Computing Systems

ApDeepSense: Deep Learning Uncertainty Estimation Without the Pain for IoT Applications

Shuochao Yao*, Yiran Zhao*, Huajie Shao*, Chao Zhang*, Aston Zhang†,
Dongxin Liu*, Shengzhong Liu*, Lu Su‡, Tarek F. Abdelzaher*
*University of Illinois at Urbana-Champaign, †Amazon AI, ‡State University of New York at Buffalo
Email: {syao9, zhao97, hshao5, czhang82, lzhang74, dongxin3, sl29}@illinois.edu, lusu@buffalo.edu, zaher@illinois.edu

***Abstract*—Recent advances in deep-learning-based applications have attracted a growing attention from the IoT community. These highly capable learning models have shown significant improvements in expected accuracy of various sensory inference tasks. One important and yet overlooked direction remains to provide *uncertainty estimates* in deep learning outputs. Since robustness and reliability of sensory inference results are critical to IoT systems, uncertainty estimates are indispensable for IoT applications. To address this challenge, we develop ApDeepSense, an effective and efficient deep learning uncertainty estimation method for resource-constrained IoT devices. ApDeepSense leverages an implicit Bayesian approximation that links neural networks to deep Gaussian processes, allowing output uncertainty to be quantified. Our approach is shown to significantly reduce the execution time and energy consumption of uncertainty estimation thanks to a novel layer-wise approximation that replaces the traditional computationally intensive sampling-based uncertainty estimation methods. ApDeepSense is designed for neural networks trained using *dropout*; one of the most widely used regularization methods in deep learning. No additional training is needed for uncertainty estimation purposes. We evaluate ApDeepSense using four IoT applications on Intel Edison devices. Results show that ApDeepSense can reduce around 88.9% of the execution time and 90.0% of the energy consumption, while producing more accurate uncertainty estimates compared with state-of-the-art methods.**

***Index Terms*—Deep learning; Internet of Things; Uncertainty estimation; Mobile computing;**

I. Introduction

Sensory measurements on IoT devices have been widely explored for diverse physical context inference and decision-making tasks. A broad variety of applications have been proposed in the areas of health and well-being [1]–[3], tracking and localization [4]–[7], physical state monitoring [8]–[11], and crowd sensing [12]–[14]. An important part of these successful applications is usually a learning model that predicts target quantities or states based on sensory inputs.

With recent breakthroughs in deep learning, attempts to apply deep neural networks as learning models within IoT systems have shown impressive results in a large number of applications [15], including audio sensing [16], tracking and localization [17], human activity recognition [17], [18], psychological state prediction [19], and user identification [17]. Compared with traditional machine learning models, these highly capable deep neural networks are better at making sophisticated mapping between input sensory measurements and the predicted quantities or states of interest.

Despite the fact that deep learning research has made such significant improvements in various estimation, classification, and prediction tasks [16], [17], concerns remain that hinder practical deployment of deep neural networks in the context of IoT applications. One important concern is the absence of efficient and accurate solutions for quality assessment of deep learning outputs. Such assessments are indispensable for applications that interact with the physical world, where errors may have adverse economic, mission, or safety consequences. In a deep neural network, the stacked non-linear structures cause tremendous difficulties analyzing and interpreting their behaviors [20]. Therefore, practitioners either mistrust neural networks altogether or blindly trust them without a principled understanding of the underlying uncertainties.

In this paper, we address the aforementioned problem by developing an approach for estimating uncertainty in neural network outputs (i.e., *output uncertainty*).

Estimating output uncertainty of neural networks running on IoT devices is a challenging task. In principle, one can empirically estimate uncertainty through extensive testing. However, this would take a lot of energy and overhead. Illuminating studies from the machine learning community [21] recently provided exciting theoretical foundations for output uncertainty estimation by proving an equivalence between deep neural networks (with dropout regularization) and variational Gaussian processes. Yet, their proposed solution is not resource-friendly, because it is a sampling-based solution. In order to generate enough output samples for estimating output mean and variance, the solution has to run a stochastic neural network multiple times [21]. As such, it is unsuitable for low-end IoT devices. A recent study proposes an uncertainty-aware deep learning model for IoT applications that aims to reduce energy and time consumption [22]. The proposed solution, however, requires re-training the neural network to generate output uncertainty estimates, which is inefficient for IoT applications with pre-trained neural networks[1].

To this end, we propose ApDeepSense that enables pre-trained deep neural networks with dropout regularization to generate output uncertainty estimates in a computationally efficient manner without any re-training. To the best of our knowledge, this is the first paper that directly enables existing

[1]https://github.com/tensorflow/models/tree/master/research/slim#Pretrained

2575-8411/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDCS.2018.00041

dropout trained neural networks to efficiently generate output uncertainty estimates on resource-limited devices.

The basic idea of ApDeepSense is to replace the resource-hungry sampling approach with efficient layerwise distribution approximations amenable to closed-form representations. ApDeepSense extends the original operations performed on signals inside a neural network, such as matrix multiplication and activation functions, to apply to inputs described by probabilistic distributions. A closed-form Gaussian approximation is then optimally fitted to best approximate the true output distribution of each operation by minimizing the Kullback-Leibler (KL) divergence. In order to find closed-form solutions for distribution approximations, one challenge is to handle the non-linearity inherent within activation functions. ApDeepSense solves this problem by further approximating the non-linear activation functions with piece-wise linear functions. Accordingly, we can compute approximate output distributions and quantify uncertainty without any additional training steps (if the neural networks have been already trained with the dropout method).

We evaluate ApDeepSense on the Intel Edison platform [23]. We conduct four different IoT tasks focusing on health and well-being, smart city transportation, environment monitoring, and activity recognition. We compare ApDeepSense with the state-of-the-art test-time deep learning output uncertainty estimation algorithm: MCDrop [21]. Model performance, such as accuracy and output log-likelihood, as well as system performance, such as inference time and energy consumption, are all estimated on the Intel Edison platform. Experimental results show that, compared with the unbiased estimator, MCDrop, ApDeepSense can reduce around 88.9% of execution time and around 90.0% of energy consumption on average while offering better uncertainty estimates.

The rest of paper is organized as follows. Section II introduces preliminary knowledge about dropout training and its relationship with bayesian approximation. We describe the technical details of ApDeepSense in Section III. The evaluation part is presented in Section IV. Related work is introduced in Section V. Finally, we conclude the paper in Section VI.

II. Preliminaries

We begin by introducing the basics of dropout training [24] and the equivalence between neural networks with dropout training and deep Gaussian processes [21].

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 $\mathcal{X}$.

A. Dropout training

For a fully-connected neural network, the layer-wise operations can be formulated as:

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

where the notation $l = 1, \cdots, L$ denotes the index of the layer. For the l^{th} layer, 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 vector is denoted as $\mathbf{x}^{(l)} \in \mathbb{R}^{d^{(l-1)}}$; and $d^{(l)}$ denotes the dimension of the l^{th} layer. In addition, $f^{(l)}(\cdot)$ denotes the nonlinear activation function of the l^{th} layer.

In order to prevent feature co-adapting and model overfitting problems, Srivastava *et al.* proposed a regularization method called dropout [24], which drops out hidden and visible units in neural networks. It is mathematically equivalent to zeroing the rows of weight matrix $\mathbf{W}^{(l)}$. Therefore, we can represent the fully-connected neural networks with dropout operations as:

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

As shown in (2), a vector of Bernoulli random 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 $\mathbf{p}^{(l)}_{[i]}$.

Note that, the above is a stochastic representation. In other words, the neural network no longer has a deterministic structure, since its structure is described in part by random variables (namely, the Bernoulli variables mentioned above). When using such a neural network to generate an output, the expected value of the output for a given input has to be computed over the distributions of the Bernoulli variables. The variance of the output is a measure of neural network output uncertainty. The challenge we address in this paper is to estimate this uncertainty given the distribution of the Bernoulli variables for the dropout probabilities.

B. Dropout as a Bayesian approximation

Bayesian models are a powerful tool to model output uncertainty [25]. However, training a Bayesian neural network is a computationally intensive task [26]. Gal *et al.* proved the equivalence between dropout training and approximate inference in a deep Gaussian process [21]. In this subsection, we provide the necessary background for treating dropout as a Bayesian approximation.

For the Bayesian approach, we are interested in learning the posterior distribution over weight matrices $p(\mathcal{W}|\mathbf{X}, \mathbf{Y})$ given training data $\mathbf{X}$ and labels $\mathbf{Y}$, where $\mathcal{W} = \{\mathbf{W}^{(l)}\}$. Then, the posterior can be applied to calculate the output distribution $\mathbf{y}$ of a testing data $\mathbf{x}$ through

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

However, computing the exact posterior distribution is not tractable in a Bayesian neural network. Instead, we can use the variational inference to approximate the posterior distribution [27]. Variational inference finds the best approximate posterior distribution $q(\mathcal{W})$ within a family of simplified distributions. Gal *et al.* proved that, if we select the approximate posterior distribution to be:

$$\begin{aligned} \mathbf{z}_{[i]}^{(l)} &\sim \text{Bernoulli}(\mathbf{p}_{[i]}^{(l)}), \\ q\big(\mathbf{W}^{(l)}\big) &= \text{diag}\big(\mathbf{z}^{(l)}\big)\mathbf{W}^{(l)}. \end{aligned} \tag{3}$$

then there is an equivalence between the deep Gaussian process and the fully-connected neural network with dropout operations trained with mean square error or a cross-entropy loss function. Since the approximate posterior distribution $q(\mathcal{W})$ chosen in (3) is similar to the dropout operation shown in (2), it has been shown that their training objective functions are equivalent [21].

Thus, during the inference, we can estimate the output mean and variance using samples generated with random dropout masks $\mathbf{z}_{[i]}^{(l)} \sim \text{Bernoulli}(\mathbf{p}_{[i]}^{(l)})$. In order to obtain an accurate output mean and variance estimate, more samples are required, which also means running the whole neural network more times.

This approach is not practical for mobile computing applications, although it is mathematically grounded. In the following section, we will introduce ApDeepSense that replaces the computationally intensive sampling process with a resource-friendly distribution approximation method.

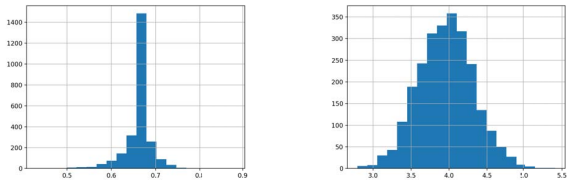
III. ApDeepSense Model

We introduce the technical details of ApDeepSense. We are interested in computing the output distribution that results from the input data and the stochastic (trained) neural network structure. The idea is to approximate the output distribution at each neural network layer by extending matrix multiplication and activation functions used within the later to apply to probabilistic distribution inputs. Such extended operations will now yield not only the expected value of each output of the layer (as is commonly done in deep learning systems), but also the entire probability distribution of that output. We select the multivariate Gaussian distribution to approximate the output distribution of each layer. Below, we first explain the rationale, then derive a closed-form approximation of the output distribution of each operation.

A. The choice of approximation distribution family

In this subsection, we will show evidence justifying the choice of the multivariate Gaussian distribution as the layer-wise distribution approximation family.

First, we observe that the theoretical proof of equivalence between dropout and deep Gaussian processes starts from the case of Gaussian processes and two-layer neural networks. This case is then extended to deep Gaussian processes and deep neural networks by feeding the output of one Gaussian process to the covariance of the next [21]. Therefore, the internal-layer representations of deep neural networks can be represented by the internal representations of deep Gaussian processes, which are multivariate Gaussian distributions.



(a) The output distribution of a hidden unit in the 12th layer.

(b) The output distribution of a hidden unit in the 18th layer.

Fig. 1: The output distributions of hidden units in a neural network.

Next, we empirically show the distribution of units in the hidden layers with a toy experiment. We train a 20-layer fully-connected neural network with dropout operation and Rectified Linear Unit (ReLU) activation function (except for the output layer) to learn the sum of 200 independent Gaussian variables. We randomly pick one hidden unit from the second and the third layer respectively. Then, we run the whole neural network for $25,000$ times with random dropout masks and record generated samples from these two hidden units. The generated distributions are shown in Figure 1.

The output distributions of two hidden units in Figure 1 clearly exhibit the shapes of bell curves with different means and variances. This toy example shows that the distributions of internal hidden units of a neural network can be approximated by the multivariate Gaussian distribution empirically.

In this section, we argued, both theoretically and empirically, for our choice of using the multivariate Gaussian distributions to approximate the output distributions of layers in the neural network. In the following subsections, we will show the technical details of approximating output distributions of basic operations by the multivariate Gaussian distribution with a diagonal covariance matrix.

B. Approximation criteria

The core contribution of ApDeepSense is to extend the basic operations in the stochastic neural networks (generated by dropout) to output not only an expected value but also a probability distribution of the output random variable. Since computing the exact output distribution is not tractable, approximation is needed. In ApDeepSense, we approximate the output distribution with the multivariate Gaussian distribution based on minimizing the Kullback-Leibler (KL) divergence between the real and approximate distributions.

Since we discuss the multivariate Gaussian distribution with a diagonal covariance matrix, without loss of generality, we mainly focus on the analysis of the output distribution with a single element in the rest of paper.

We begin by introducing Lemma 1 as follows.

Lemma 1. *Assume that the exact output distribution is $p(x)$ and the approximate output distribution is $q(x) \sim \mathcal{N}(\mu, \sigma^2)$. Finding the best fitting $q(x)$ according to KL divergence is equivalent to moments matching between $p(x)$ and $q(x)$.*

Proof. The main objective function for the approximation is

$$\begin{aligned}
&\min_q KL(p(x)||q(x)),\\
=&\min_q \int p(x)\log\left(\frac{p(x)}{q(x)}\right)dx\\
=&\min_{\mu_x,\sigma^2} -\int p(x)\log\mathcal{N}(\mu,\sigma^2)dx\\
=&\min_{\mu_x,\sigma^2} \frac{\log(\sigma^2)}{2}+\frac{\int_x p(x)(x-\mu)^2}{2\sigma^2}.
\end{aligned}$$

In order to obtain the optimal $q(x)$, we take derivative over μ and σ^2 and make them equal to 0. Then we can obtain the solution

$$\mu = \int p(x)xdx, \tag{4}$$

$$\sigma^2 = \int p(x)(x-\mu)^2dx. \tag{5}$$

□

Therefore, according to (4) and (5), the approximation process for each operation can be viewed as mean and variance matching between the exact output distribution $p(x)$ and the approximating distribution $q(x)$.

C. Approximating matrix multiplication with dropout

We start with the basic matrix multiplication operation with dropout. The operation can be formulated as:

$$\begin{aligned}
\mathbf{z}_{[i]} &\sim \text{Bernoulli}(\mathbf{p}_{[i]}),\\
\mathbf{x}_{[i]} &\sim \mathcal{N}\big(\boldsymbol{\mu}_{[i]},\boldsymbol{\sigma}^2_{[i]}\big),\\
\tilde{\mathbf{W}} &= \text{diag}\big(\mathbf{z}\big)\mathbf{W},\\
\mathbf{y} &= \mathbf{x}\tilde{\mathbf{W}}+\mathbf{b}.
\end{aligned} \tag{6}$$

As shown in (6), elements in the vector of Bernoulli variables $\mathbf{z}$ and the vector of Gaussian variables $\mathbf{x}$ are independent random variables. According to Lemma 1, we need to calculate the means and variances of the output distribution $p(\mathbf{y})$.

First, we calculate the mean of output variables,

$$\begin{aligned}
\mathbb{E}[\mathbf{y}_{[j]}] =&\mathbb{E}\Big[\sum_i \mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}\Big]\\
=&\sum_i \boldsymbol{\mu}_{[i]}\mathbf{p}_{[i]}\mathbf{W}_{[i,j]}.
\end{aligned} \tag{7}$$

Then, we calculate the variance of output variables. Since the output variable $\mathbf{y}_{[j]}$ is the sum of independent variable $\mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}$, according to Bienaymé formula [28], the variance of the sum of independent random variables is the sum of their variances,

$$\begin{aligned}
\text{Var}[\mathbf{y}_{[j]}] =&\text{Var}\Big[\sum_i \mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}\Big]\\
=&\sum_i \text{Var}\big[\mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}\big]\\
=&\sum_i \mathbb{E}\big[(\mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]})^2\big]-\mathbb{E}\big[\mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}\big]^2\\
=&\sum_i \mathbb{E}\big[\mathbf{x}_{[i]}{}^2\big]\mathbb{E}\big[\mathbf{z}_{[i]}{}^2\big]\mathbf{W}_{[i,j]}{}^2-\boldsymbol{\mu}_{[i]}{}^2\mathbf{p}_{[i]}{}^2\mathbf{W}_{[i,j]}{}^2\\
=&\sum_i \big(\boldsymbol{\mu}_{[i]}{}^2+\boldsymbol{\sigma}_{[i]}{}^2\big)\mathbf{p}_{[i]}\mathbf{W}_{[i,j]}{}^2-\boldsymbol{\mu}_{[i]}{}^2\mathbf{p}_{[i]}{}^2\mathbf{W}_{[i,j]}{}^2.
\end{aligned} \tag{8}$$

However, (7) and (8) are not represented in matrix forms that can be efficiently computed with standard deep learning libraries such as TensorFlow, MXNet, and Theano. If we denote element-wise multiplication as notation $\odot$ and denote $\mathbf{X}^2 \stackrel{\text{def}}{=} \mathbf{X}\odot\mathbf{X}$, we can represent the matrix form of (7) and (8) as:

$$\mathbb{E}[\mathbf{y}] = (\boldsymbol{\mu}\odot\mathbf{p})\mathbf{W}, \tag{9}$$

$$\text{Var}[\mathbf{y}] = \big((\boldsymbol{\mu}^2+\boldsymbol{\sigma}^2)\odot\mathbf{p}-\boldsymbol{\mu}^2\odot\mathbf{p}^2\big)\mathbf{W}^2. \tag{10}$$

Therefore, we can efficiently compute the output distribution of matrix multiplication with dropout, $p(\mathbf{y})\sim\mathcal{N}(\mathbb{E}[\mathbf{y}],\text{Var}[\mathbf{y}])$ through (9) and (10).

D. Approximating activation functions

Next, we describe how to compute the output distribution of activation functions. All commonly used activation functions are element-wise operations. Hence, without loss of generality, we focus on the operation applied on each input random variable. We assume that the activation function with random variable is formulated as:

$$\begin{aligned}
\mathbf{x}_{[i]} &\sim \mathcal{N}\big(\boldsymbol{\mu}_{[i]},\boldsymbol{\sigma}^2_{[i]}\big),\\
\mathbf{y}_{[i]} &= f(\mathbf{x}_{[i]}),
\end{aligned} \tag{11}$$

where $f(\cdot)$ is the element-wise activation function.

According to (11), an activation function on random variables can be regarded as the composition of input random variable $\mathbf{x}_{[i]}$ with function $f(\cdot)$. However, the challenge is that computing the mean and variance of composed random variables with an arbitrary non-linear function is usually not amenable to a closed-form solution. ApDeepSense tackles this challenge by further approximating non-linear functions with piece-wise linear functions. Since the linear transformation of Gaussian random variables is well-understood, we can calculate the mean and variance of an output Gaussian random variable with a piece-wise linear transformation.

We assume that the whole axis $(-\infty,+\infty)$ is divided into P parts, namely (a_p,b_p) for $p=1,\cdots,P$, where $b_p=a_{p+1}$, $a_1=-\infty$, and $b_P=+\infty$. The pre-defined piece-wise linear activation function $\bar{f}(\cdot)$ is a linear function $y_p=k_p\cdot x+c_p$ on each interval (a_p,b_p). One example of approximating sigmoid function with piece-wise linear function can be found in [29]. The input random variable follows a Gaussian distribution: $x\sim\mathcal{N}(\mu,\sigma^2)$.

Then, we calculate the mean and variance of the output distribution $y=\bar{f}(x)$ respectively.

$$\mu_y = \mathbb{E}[y]=\sum_{p=1}^{P}\mathbb{E}_p[y] \tag{12}$$

$$= \sum_{p=1}^{P}\int_{a_p}^{b_p}(k_px+c_p)\cdot\mathcal{N}(\mu,\sigma^2)dx. \tag{13}$$

$$\sigma_y{}^2 = \mathbb{E}[(y-\mu_y)^2] = \sum_{p=1}^{P} \mathbb{E}_p[(y-\mu_y)^2] \quad (14)$$

$$= \sum_{p=1}^{P} \int_{a_p}^{b_p} (k_p x + c_p - \mu_y)^2 \cdot \mathcal{N}(\mu, \sigma^2) dx. \quad (15)$$

We calculate the mean and variance by considering the case where $k_p \neq 0$ and $k_p = 0$.

1) When $k_p \neq 0$: Within the interval (a_p, b_p), $y = \bar{f}(x)$ follows a Gaussian distribution $\mathcal{N}(\mu_p, \sigma_p{}^2)$, where $\mu_p = k_p\mu + c_p$ and $\sigma_p = |k_p|\sigma$. Then $\mathbb{E}_p[y]$ (13) can be reformulated as:

$$\mathbb{E}_p[y] = \int_{\bar{a}_p}^{\bar{b}_p} y \cdot \mathcal{N}(\mu_p, \sigma_p{}^2) dy,$$

where $\bar{a}_p = k_p a_p + c_p$ and $\bar{b}_p = k_p b_p + c_p$.

We assume that

$$M_p = \int_{\bar{a}_p}^{\bar{b}_p} (y-\mu_p) \cdot \mathcal{N}(\mu_p, \sigma_p{}^2) dy, \quad (16)$$

$$D_p = \int_{a}^{b} \mathcal{N}(\mu, \sigma^2) dx. \quad (17)$$

Then we can obtain the representation of $\mathbb{E}_p[y]$ as:

$$\mathbb{E}_p[y] = \mu_p \cdot D_p + M_p. \quad (18)$$

Similarly, $\mathbb{E}_p[(y-\mu_y)^2]$ (15) can be reformulated as:

$$\mathbb{E}_p[(y-\mu_y)^2] = \int_{\bar{a}_p}^{\bar{b}_p} (y-\mu_y)^2 \cdot \mathcal{N}(\mu_p, \sigma_p{}^2) dy.$$

We further assume that

$$V_p = \int_{\bar{a}_p}^{\bar{b}_p} (y-\mu_p)^2 \cdot \mathcal{N}(\mu_p, \sigma_p{}^2) dy. \quad (19)$$

Then we can obtain the representation of $\mathbb{E}_p[(y-\mu_y)^2]$ as:

$$\mathbb{E}_p[(y-\mu_y)^2] = V_p + 2(\mu_p - \mu_y) \cdot M_p + (\mu_p - \mu_y)^2 \cdot D_p. \quad (20)$$

2) When $k_p = 0$: In this case, $\bar{f}(\cdot)$ is a constant function, which usually exists when $p = 1$ or $p = P$. Then we can easily obtain that

$$\mathbb{E}_p[y] = c_p \cdot D_p, \quad (21)$$

$$\mathbb{E}_p[(y-\mu_y)^2] = (c_p - \mu_y)^2 \cdot D_p. \quad (22)$$

3) Summary: Once we can calculate M_p, D_p, V_p, we can then obtain the approximate output distribution $y \sim \mathcal{N}(\mu_y, \sigma_y{}^2)$ generated by mean (12) and variance (14) with corresponding P components in (18), (20), (21), and (22).

Clearly, (16), (17), and (19) can be calculated efficiently in closed form with basic math operations and the error function erf$(\cdot)$, which all have corresponding APIs in standard deep learning libraries such as TensorFlow and Theano.

$$D_p = \frac{1}{2} \cdot \Big(\text{erf}\Big(\frac{b_p - \mu}{\sigma\sqrt{2}}\Big) - \text{erf}\Big(\frac{a_p - \mu}{\sigma\sqrt{2}}\Big) \Big), \quad (23)$$

$$M_p = \sqrt{\frac{\sigma_p{}^2}{2\pi}} \cdot \Big(\exp\Big(-\frac{(\bar{a}_p - \mu_p)^2}{2\sigma_p{}^2}\Big) - \exp\Big(-\frac{(\bar{b}_p - \mu_p)^2}{2\sigma_p{}^2}\Big) \Big), \quad (24)$$

$$V_p = \sqrt{\frac{\sigma_p{}^2}{2\pi}} \cdot \Big((\bar{a}_p - \mu_p) \exp\Big(-\frac{(\bar{a}_p - \mu_p)^2}{2\sigma_p{}^2}\Big) - (\bar{b}_p - \mu_p) \exp\Big(-\frac{(\bar{b}_p - \mu_p)^2}{2\sigma_p{}^2}\Big) \Big) + \frac{\sigma_p{}^2}{2} \cdot \Big(\text{erf}\Big(\frac{\bar{b}_p - \mu_p}{\sigma_p\sqrt{2}}\Big) - \text{erf}\Big(\frac{\bar{a}_p - \mu_p}{\sigma_p\sqrt{2}}\Big) \Big). \quad (25)$$

Then, $\mathbb{E}_p[y]$ and $\mathbb{E}_p[(y-\mu_y)^2]$ can be calculated with (23), (24), and (25). Therefore, the final output distribution of the activation function $f(\cdot)$ is

$$\begin{aligned} y &\sim \mathcal{N}(\mu_y, \sigma_y{}^2), \\ \mu_y &= \sum_{p=1}^{P} \mathbb{E}_p[y], \\ \sigma_y{}^2 &= \sum_{p=1}^{P} \mathbb{E}_p[(y-\mu_y)^2]. \end{aligned} \quad (26)$$

Above, we have described the main technical details of ApDeepSense. The basic operations within a fully-connected neural network, such as matrix multiplication with dropout and activation function, now support taking probabilistic distributions as inputs and generate distributions as outputs. Hence, the distribution of the final output can be computed. Narrower distributions offer less uncertainty, whereas flatter distributions offer more uncertainty.

IV. Evaluation

In this section, we evaluate the quality of the computed output probability distributions by the new neural network models. Specifically we consider the mean absolute error and predictive log-likelihood. The former is a metric of accuracy of the output distribution mean. The latter (predictive log likelihood) measures the correspondence between ground truth values and their predicted distribution. Lower numbers mean higher correspondence. We then evaluate system performance, such as running time and energy consumption on the Intel Edison platform. In the following subsections, first, we introduce the evaluation tasks, corresponding datasets, baseline algorithms, and testing neural network details. Next, we evaluate the model performance including mean absolute error or accuracy and negative log-likelihood for all algorithms. At last, we show the system performance including inference time and energy consumption on the Intel Edison platform.

A. Testing hardware

Our hardware is based on the Intel Edison computing platform [23]. 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 experiments.

B. Evaluation tasks and datasets

We evaluate ApDeepSense on four different IoT tasks. Here, we introduce the details of these four tasks and their corresponding datasets.

- *BPEst: Cuffless blood pressure monitoring through photoplethysmogram.* The first task monitors cuffless blood pressure through a photoplethysmogram from a fingertip. The photoplethysmogram from fingertip (PPG) and arterial blood pressure (ABP) signal (mmHg) is extracted for the non-invasive cuffless blood pressure monitoring task.[2] The target of BPEst task is to infer the waveform of ABP based on the waveform of PPG collected from fingertips, estimating a 2-second ABP waveform (250 samples) based on the corresponding 2-second PPG waveform.
- *NYCommute: Commute time estimation of New York City.* This second task estimates commute time in New York City through the pick-up time and location as well as the drop-off location, which is an important component in smart transportation. We use (parts of) the yellow and green taxi trip records within January 2016 as the training, validation, and testing datasets.[3] The input of the learning model is a vector with 5 elements, containing the standardized longitude and latitude of the pick-up and drop-off location as well as the pick-up time within a day. The output of the learning model is the commute time.
- *GasSen: Estimate dynamic gas mixtures from chemical sensors.* The third task estimates real concentration of Ethylene and CO gas mixture from an array of 16 low-end chemical sensors. [4] 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). The dataset contains 9 users, 6 activities (biking, sitting, standing, walking, climbStair-up, and climbStair-down), and 6 types of mobile devices [5].

C. Testing models and uncertainty estimation algorithms

For each experiment, we use two pre-trained neural networks with the same structure but different activation functions.

- *DNN-ReLU:* A 5-layer fully-connected neural network with 512 hidden dimension and ReLU activation function. Since ReLU is already a piece-wise linear function, no activation function approximation is needed.
- *DNN-Tanh:* A 5-layer fully-connected neural network with 512 hidden dimension and Tanh activation function. ApDeepSense uses a piece-wise linear function with 7 pieces to approximate Tanh in all experiments.

During all experiments, we compare two output uncertainty estimation algorithms.

- *ApDeepSense:* The algorithm we propose in this paper.
- *MCDrop-k:* A sampling-based unbiased uncertainty estimation method for deep neural network with dropout. This algorithm generates k output samples with random dropout masks, and then estimates predictive uncertainties based on these k output samples [21]. During all experiments, we choose $k = [3, 4, 10, 30, 50]$.
- *RDeepSense:* An efficient uncertainty estimation method for deep neural network that requires retraining [22]. We introduce this method to illustrate the upper bound of model estimation performance can be achieved with retraining.

D. Model estimation performance

In this subsection, model estimation performance is discussed for four tasks. For each task, two measures are used to show the bias-variance tradeoff among different output uncertainty estimation algorithms. For regression tasks, mean absolute error (MAE) and negative log-likelihood (NLL) are measured. For classification tasks, accuracy (ACC) and negative log-likelihood (NLL) are measured. Negative log likelihood depends on the output uncertainty. It is a popular metric for evaluating output uncertainty [30].

1) BPEst: We compare all the algorithms for two pre-trained neural networks based on mean absolute error (MAE) and negative log-likelihood (NLL), which is illustrated in Table I.

TABLE I: Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) for the BPEst task.

	MAE	NLL
DNN-ReLU-ApDeepSense	**13.41**	**4.56**
DNN-ReLU-MCDrop-3	13.91	57.72
DNN-ReLU-MCDrop-5	13.68	7.89
DNN-ReLU-MCDrop-10	13.50	5.74
DNN-ReLU-MCDrop-30	13.38	5.14
DNN-ReLU-MCDrop-50	**13.35**	5.06
DNN-ReLU-RDeepSense	14.18	**3.46**
DNN-Tanh-ApDeepSense	**19.38**	**5.39**
DNN-Tanh-MCDrop-3	19.61	520.30
DNN-Tanh-MCDrop-5	19.51	56.74
DNN-Tanh-MCDrop-10	19.39	32.68
DNN-Tanh-MCDrop-30	19.32	25.19
DNN-Tanh-MCDrop-50	**19.30**	23.99
DNN-Tanh-RDeepSense	19.38	**4.53**

For both pre-trained neural networks, DNN-ReLU and DNN-Tanh, ApDeepSense is consistently the best predictive uncertainty estimator with the smallest NLL. The result shows that approximation method used in ApDeepSense works well in the real dataset. The experiments with DNN-Tanh demonstrate the effectiveness of approximating non-linear activation functions with piece-wise linear functions. Notice

[2]https://archive.ics.uci.edu/ml/datasets/Cuff-Less+Blood+Pressure+Estimation

[3]http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[4]https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+under+dynamic+gas+mixtures

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

that ApDeepSense is not the best-performing algorithm with respect to MAE. ApDeepSense achieves a better bias-variance tradeoff by directly approximating the output distribution.

2) NYCommute: We compare all the algorithms for two pre-trained neural networks based on mean absolute error (MAE) and negative log-likelihood (NLL), which is illustrated in Table II.

TABLE II: Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) for the NYCommute task.

	MAE	NLL
DNN-ReLU-ApDeepSense	**5.44**	**135.19**
DNN-ReLU-MCDrop-3	5.54	6569.04
DNN-ReLU-MCDrop-5	5.50	1898.79
DNN-ReLU-MCDrop-10	5.47	1140.90
DNN-ReLU-MCDrop-30	5.45	889.60
DNN-ReLU-MCDrop-50	5.44	838.94
DNN-ReLU-RDeepSense	5.64	**7.7**
DNN-Tanh-ApDeepSense	**6.41**	**123.75**
DNN-Tanh-MCDrop-3	6.59	7517.95
DNN-Tanh-MCDrop-5	6.54	892.34
DNN-Tanh-MCDrop-10	6.51	443.04
DNN-Tanh-MCDrop-30	6.48	332.42
DNN-Tanh-MCDrop-50	6.47	321.73
DNN-Tanh-RDeepSense	6.59	**14.11**

ApDeepSense is consistently the best-performing algorithm for both the MAE and NLL measures. The sampling-based algorithm MCDrop shows inferior performance even with 50 samples, *i.e.*, running the whole neural network for 50 times. The result implies that even more samples are required for MCDrop to compete with ApDeepSense.

3) GasSen: We still compare all the algorithms for two pre-trained neural networks based on mean absolute error (MAE) and negative log-likelihood (NLL), which is illustrated in Table III.

TABLE III: Mean Absolute Error (MAE) and Negative Log-Likelihood (NLL) for the GasSen task.

	MAE	NLL
DNN-ReLU-ApDeepSense	**19.42**	**40.21**
DNN-ReLU-MCDrop-3	21.17	456.59
DNN-ReLU-MCDrop-5	20.36	342.13
DNN-ReLU-MCDrop-10	19.66	333.52
DNN-ReLU-MCDrop-30	19.27	303.66
DNN-ReLU-MCDrop-50	19.15	290.51
DNN-ReLU-RDeepSense	**15.25**	**3.77**
DNN-Tanh-ApDeepSense	**39.20**	**6.32**
DNN-Tanh-MCDrop-3	35.74	103.73
DNN-Tanh-MCDrop-5	32.76	41.67
DNN-Tanh-MCDrop-10	32.30	25.13
DNN-Tanh-MCDrop-30	31.71	19.74
DNN-Tanh-MCDrop-50	31.57	18.81
DNN-Tanh-RDeepSense	**19.36**	**4.23**

ApDeepSense still outperforms all the algorithms for uncertainty estimation with NLL metric. In this dataset, ApDeepSense still achieves a bias-variance tradeoff with better NLL. Specially in the DNN-Tanh network, we can clearly see that ApDeepSense sets the uncertainty estimation as its first priority. As we mentioned in the experiment on BPEst task, it is due to the nature of distribution approximation within the ApDeepSense.

TABLE IV: Accuracy (ACC) and Negative Log-Likelihood (NLL) for the HHAR task.

	ACC	NLL
DNN-ReLU-ApDeepSense	**79.12%**	**1.02**
DNN-ReLU-MCDrop-3	73.79%	1.479
DNN-ReLU-MCDrop-5	75.34%	1.476
DNN-ReLU-MCDrop-10	76.38%	1.475
DNN-ReLU-MCDrop-30	76.24%	1.475
DNN-ReLU-MCDrop-50	76.72%	1.476
DNN-ReLU-RDeepSense	**83.98%**	**0.16**
DNN-Tanh-ApDeepSense	**73.57%**	**0.23**
DNN-Tanh-MCDrop-3	70.43%	1.45
DNN-Tanh-MCDrop-5	71.07%	1.38
DNN-Tanh-MCDrop-10	71.68%	1.33
DNN-Tanh-MCDrop-30	72.81%	1.31
DNN-Tanh-MCDrop-50	73.29%	1.29
DNN-Tanh-RDeepSense	**86.78%**	**0.21**

4) HHAR: Since HHAR is a classification task, we compare all the algorithms for two pre-trained neural networks based on accuracy in percentage (ACC) and negative log-likelihood (NLL), which is illustrated in Table IV.

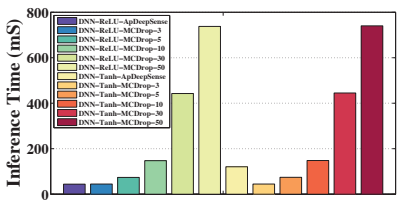
ApDeepSense outperforms the other algorithms according to ACC and NLL metrics. ApDeepSense achieves both better classification results and likelihood estimation at the same time.

E. System performance

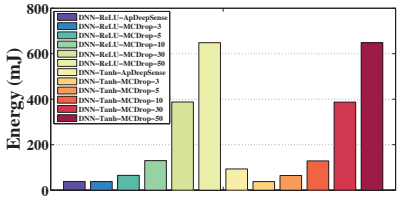
In this subsection, we compare the system performances for all uncertainty estimation algorithms, including inference times, energy consumption, and model and system performance tradeoff. All the experiments are conducted on Intel Edison platform with the CPU unit. All the experiments are conducted for 5000 times and the mean values of such experimental results are reported.

First, we illustrate the inference time and energy consumption of all algorithms for all BPEst, NYCommute, GasSen, and HHAR tasks in Figures 2, 3, 4, and 5 respectively. We can see that ApDeepSense saves around 94.1% and 83.6% inference time in average for DNN with ReLU and Tanh activation function respectively. At the same time, ApDeepSense saves around 94.2% and 85.7% energy consumption in average for DNN with ReLU and Tanh activation function respectively. Here is the main reason for this improvement. MCDrop-50 runs the original neural network for 50 times. ApDeepSense uses a piece-wise linear function with 2 pieces to approximate ReLU function and a piece-wise linear function with 7 pieces to approximate Tanh function. Therefore, ApDeepSense theoretically can save around $1-2/50 = 96\%$ and $1-7/50 = 86\%$ computations for DNN-ReLU and DNN-Tanh respectively, which agrees with our empirical results.

Then we analyze the relationship between the energy consumption and quality of uncertainty estimation for all algo-

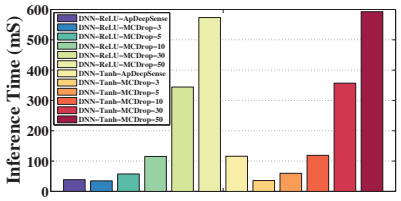


(a) The inference time of the BPEst task.

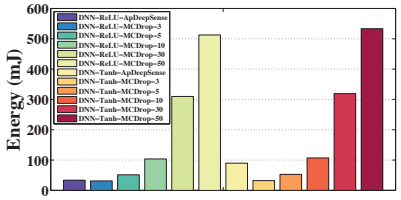


(b) The energy consumption of the BPEst task.

Fig. 2: The inference time and energy consumption of the BPEst task.

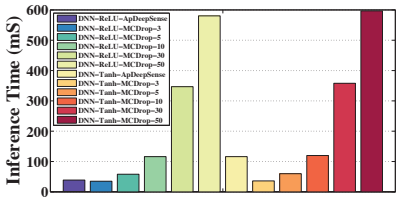


(a) The inference time of the NYCommute task.

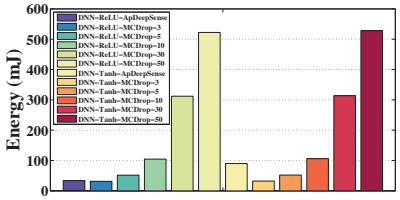


(b) The energy consumption of the NYCommute task.

Fig. 3: The inference time and energy consumption of the NYCommute task.

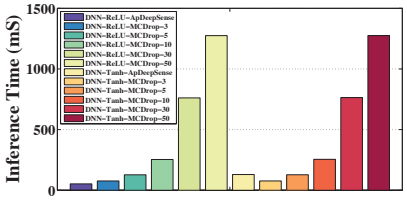


(a) The inference time of the GasSen task.

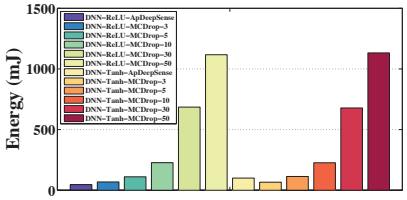


(b) The energy consumption of the GasSen task.

Fig. 4: The inference time and energy consumption of the GasSen task.



(a) The inference time of the HHAR task.



(b) The energy consumption of the HHAR task.

Fig. 5: The inference time and energy consumption of the HHAR task.

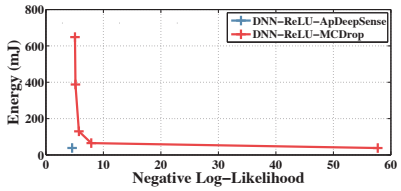
rithms. In this experiment, we use the negative log-likelihood (NLL) to represent the quality of uncertainty estimation. Smaller NLL means better uncertainty estimation quality. The experimental results are illustrated in Fig. 6, 7, 8, and 9.

The points in the left-bottom corner of these graphs represent better tradeoffs between energy consumption and uncertainty estimation (*i.e.*, , consuming less energy to achieve a predictive distribution with lower negative log-likelihood). Therefore, these tradeoffs shows that ApDeepSense is a more effective and efficient uncertainty estimation algorithm for deep neural networks.

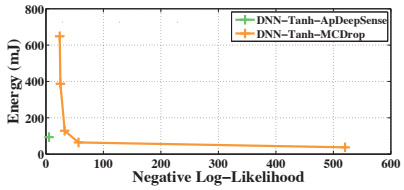
V. Related Work

Reliability and uncertainty estimation is an important topic in the area of Internet of Things. Zhou *et al.* [31] modelled ego-motion uncertainty on IoT platform. Van *et al.* [32] applied sensing uncertainty on mobile robots controlling with collision avoidance. Wang *et al.* [33] used uncertainty estimation to design the optimal sensor sampling policy. Jin *et al.* [34] proposed a holistic framework that optimizes both reliability and temporality for multiple coexisting networks.

Recently, deep learning models were considered in IoT-related systems. Lane *et al.* [16] applied deep learning to solve audio sensing tasks. Yao *et al.* [17] proposed a unified deep learning model that fuses multiple sensor inputs and extracts time dependencies from sensing data. Yao *et al.* [35] proposed a structure compression algorithm for neural networks, improving the system efficiency on low-end IoT devices. However, less attention has been paid to estimating deep learning uncertainty in an IoT system. Recent studies provide a theoretically justified output uncertainty estimation method for neural networks [21]. However, it relies on

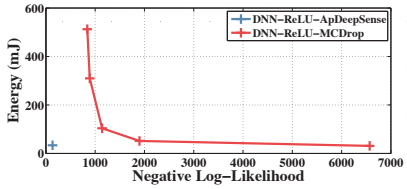


(a) The tradeoff for DNN-ReLU of the BPEst task..

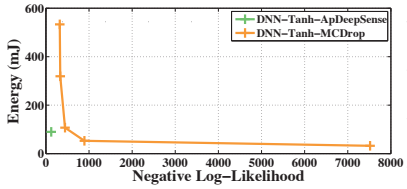


(b) The tradeoff for DNN-Tanh of the BPEst task.

Fig. 6: The tradeoff between energy consumption and NLL of the BPEst task.

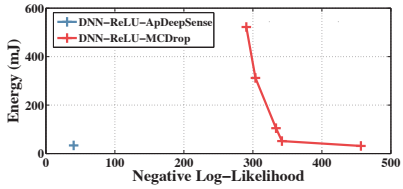


(a) The tradeoff for DNN-ReLU of the NYCommute task..

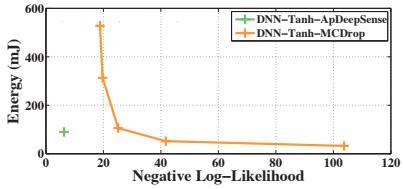


(b) The tradeoff for DNN-Tanh of the NYCommute task.

Fig. 7: The tradeoff between energy consumption and NLL of the NYCommute task.



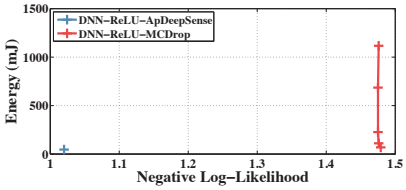
(a) The tradeoff for DNN-ReLU of the GasSen task..



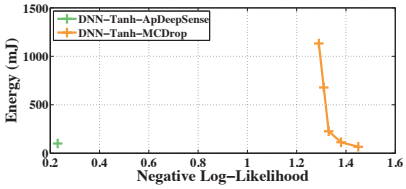
(b) The tradeoff for DNN-Tanh of the GasSen task.

Fig. 8: The tradeoff between energy consumption and NLL of the GasSen task.

computationally intensive operations that are not practical for mobile computing applications. Recent work propose an



(a) The tradeoff for DNN-ReLU of the HHAR task..



(b) The tradeoff for DNN-Tanh of the HHAR task.

Fig. 9: The tradeoff between energy consumption and NLL of the HHAR task.

energy-efficient uncertainty estimation algorithm for neural networks [22]. However, users have to retrain the whole neural network according to the proposed design specifications.

To the best of our knowledge, ApDeepSense is the first energy-efficient test-time uncertainty estimation algorithm for trained deep neural networks deployed on IoT devices.

VI. Conclusion

We described ApDeepSense, an effective and efficient uncertainty estimation algorithm for fully-connected neural networks. ApDeepSense solves the problem of providing uncertainty estimations for neural networks without further structure-changing or re-training by adopting a novel layer-wise distribution approximation method. In addition, piece-wise linear functions are used to approximate nonlinear activation functions for deriving an efficient closed-form solution. Experiments on the Intel Edison platform with four mobile computing tasks show that ApDeepSense saves around 90% inference time and energy consumption to provide uncertainty estimations with smaller negative log-likelihood measures compared with the state-of-the-art baseline algorithm.

The paper leaves exciting oppoertunities for future work. For example, currently, ApDeepSense can only support fully-connected neural networks. It is possible to extend the solution proposed in Section III to convolutional and recurrent neural networks by replacing the original dropout operation with convolutional dropout [36] and recurrent dropout [37]. These two dropout operations can convert convolutional neural networks and recurrent neural networks into Bayesian neural networks. However, challenges still exist in extending related operations, such as convolution, to apply to probabilistic distribution inputs and offer closed-form output distribution using APIs in standard deep learning libraries. Such extensions and approximations are a topic of future work of the authors.

VII. Acknowledgements

We sincerely thank Yuan He for shepherding the final version of this paper, and the anonymous reviewers for their

invaluable comments. Research reported in this paper was sponsored in part by NSF under grants CNS 16-18627 and CNS 13-20209 and in part by the Army Research Laboratory under Cooperative Agreements W911NF-09-2-0053 and W911NF-17-2-0196. 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, NSF, 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.

REFERENCES

[1] J. Ko, C. Lu, M. B. Srivastava, J. A. Stankovic, A. Terzis, and M. Welsh, "Wireless sensor networks for healthcare," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1947–1960, 2010.

[2] C. A. Boano, M. Lasagni, and K. Römer, "Non-invasive measurement of core body temperature in marathon runners," in *Proceedings of the* 10^{th} *IEEE International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. IEEE, 2013.

[3] J. H. Lim, A. Zhan, E. Goldschmidt, J. Ko, M. Chang, and A. Terzis, "Healthos: a platform for pervasive health applications," in *Proceedings of the Second ACM Workshop on Mobile Systems, Applications, and Services for HealthCare*. ACM, 2012, p. 4.

[4] M. Bocca, O. Kaltiokallio, N. Patwari, and S. Venkatasubramanian, "Multiple target tracking with rf sensor networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 8, pp. 1787–1800, 2014.

[5] W. Xi, Y. He, Y. Liu, J. Zhao, L. Mo, Z. Yang, J. Wang, and X. Li, "Locating sensors in the wild: pursuit of ranging quality," in *Proceedings of the 8th ACM conference on Embedded Networked Sensor Systems*. ACM, 2010, pp. 295–308.

[6] X. Jiang, C.-J. M. Liang, K. Chen, B. Zhang, J. Hsu, J. Liu, B. Cao, and F. Zhao, "Design and evaluation of a wireless magnetic-based proximity detection platform for indoor applications," in *Proceedings of the 11th international conference on Information Processing in Sensor Networks*. ACM, 2012, pp. 221–232.

[7] R. Jurdak, P. Sommer, B. Kusy, N. Kottege, C. Crossman, A. Mckeown, and D. Westcott, "Camazotz: multimodal activity-based gps sampling," in *Proceedings of the 12th international conference on Information processing in sensor networks*. ACM, 2013, pp. 67–78.

[8] W. Du, Z. Xing, M. Li, B. He, L. H. C. Chua, and H. Miao, "Optimal sensor placement and measurement of wind for water quality studies in urban reservoirs," in *Information Processing in Sensor Networks, IPSN-14 Proceedings of the 13th International Symposium on*. IEEE, 2014.

[9] W. Hu, N. Bulusu, C. T. Chou, S. Jha, A. Taylor, and V. N. Tran, "Design and evaluation of a hybrid sensor network for cane toad monitoring," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 1, p. 4, 2009.

[10] X. Zheng, C. Julien, R. Podorozhny, F. Cassez, and T. Rakotoarivelo, "Efficient and scalable runtime monitoring for cyber–physical system," *IEEE Systems Journal*, 2016.

[11] Ş. Gună, L. Mottola, and G. P. Picco, "Dice: Monitoring global invariants with wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 10, no. 4, p. 54, 2014.

[12] 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 *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 14.

[13] 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*. International World Wide Web Conferences Steering Committee, 2017, pp. 361–370.

[14] 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*. ACM, 2017, pp. 595–604.

[15] S. Yao, Y. Zhao, A. Zhang, S. Hu, H. Shao, C. Zhang, S. Lu, and T. Abdelzaher, "Deep learning for the internet of things," *Computer*, vol. 51, 2018.

[16] 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*. ACM, 2015, pp. 283–294.

[17] 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, pp. 351–360.

[18] 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 *Ubicomp: Adjunct*. ACM, 2016.

[19] G. Mikelsons, M. Smith, A. Mehrotra, and M. Musolesi, "Towards deep learning models for psychological state prediction using smartphone data: Challenges and opportunities," *arXiv preprint arXiv:1711.06350*, 2017.

[20] Z. C. Lipton, "The mythos of model interpretability," *arXiv preprint arXiv:1606.03490*, 2016.

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

[22] S. Yao, Y. Zhao, H. Shao, A. Zhang, C. Zhang, S. Li, and T. Abdelzaher, "Rdeepsense: Reliable deep mobile computing models with uncertainty estimations," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 4, p. 26, 2017.

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

[24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *JMLR*, 2014.

[25] M. E. Borsuk, C. A. Stow, and K. H. Reckhow, "A bayesian network of eutrophication models for synthesis, prediction, and uncertainty analysis," *Ecological Modelling*, vol. 173, no. 2, pp. 219–239, 2004.

[26] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[27] M. J. Wainwright, M. I. Jordan *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[28] H. Läuter, "Loéve, m.: Probability theory i and ii, (graduate texts in mathematics 45 u. 46) springer-verlag, berlin-heidelberg-new york 1977/78, 441, 429 pp," 1982.

[29] H. Amin, K. M. Curtis, and B. R. Hayes-Gill, "Piecewise linear approximation applied to nonlinear function of a neural network," *IEE Proceedings-Circuits, Devices and Systems*, 1997.

[30] J. Quinonero-Candela, C. E. Rasmussen, F. Sinz, O. Bousquet, and B. Scholkopf, "Evaluating predictive uncertainty challenge," *Lecture Notes in Computer Science*, vol. 3944, pp. 1–27, 2006.

[31] D. Zhou, V. Fremont, B. Quost, and B. Wang, "On modeling ego-motion uncertainty for moving object detection from a mobile platform," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014.

[32] J. Van Den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, "Lqg-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty," in *ICRA*. IEEE, 2012.

[33] Y. Wang, B. Krishnamachari, Q. Zhao, and M. Annavaram, "Markov-optimal sensing policy for user state estimation in mobile devices," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 2010.

[34] X. Jin, F. Kong, L. Kong, W. Liu, and P. Zeng, "Reliability and temporality optimization for multiple coexisting wirelesshart networks in industrial environments," *IEEE Transactions on Industrial Electronics*, 2017.

[35] 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.

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

[37] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 1019–1027.