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

Shuochao Yao et al

Problem Statement

- Deep learning models have shown significant improvement in the expected accuracy of sensory inference tasks but do not provide uncertainty estimates in their outputs.

- Uncertainty estimates are indispensable for IoT applications. Empirical extensive testing consumes a lot of energy and overhead.

- Authors proposed ApDeepSense, an efficient deep learning uncertainty estimation method for resource-constrained IoT devices. Achieved 88.9% reduction in run time and 90% reduction in energy consumption.

- Approach links Bayesian approximation allowing the output uncertainty to be quantified. Introduced a novel layer-wise approximation which replaces the sampling-based uncertainty estimation methods.

- Designed for neural networks which leverage dropout. Dropout is a patented regularization technique from Google.

ApDeepSense Model features

- Helps pre-trained deep neural networks with dropout to generate output uncertainty estimates in a computationally efficient manner without any re-training.
- Replace resource-hungry sampling method with efficient layer-wise distribution approximations
- Closed-form Gaussian approximation is then optimally fitted to best approximate the true output distribution of each operation by minimizing the Kullback-Liebler (KL) divergence.
- Trick to handle the non-linearity inherent with activation functions by substituting with piece-wise linear functions.

Preliminaries

1. Basics of dropout.

$$\mathbf{y}^{(l)} = \mathbf{x}^{(l)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}$$

$$\mathbf{x}^{(l+1)} = f^{(l)}(\mathbf{y}^{(l)})$$

$\mathbf{W}^{(l)}$, $\mathbf{b}^{(l)}$ parameters of the l^{th} layer of the neural network

- To prevent co-adapting and model overfitting, Srivastava et al. proposed dropout, which drops out hidden and visible units in neural networks.

Preliminaries

- This can be shown to be equivalent to

$$z_{[i]}^{(l)} \sim \text{Bernoulli}(p_{[i]}^{(l)})$$

$$\mathbf{W}^{*(l)} = \text{diag}(z^{(l)})\mathbf{W}^{(l)}$$

$$y^{(l)} = x^{(l)}\mathbf{W}^{*(l)} + \mathbf{b}^{(l)}$$

$$x^{(l+1)} = f^{(l)}(y^{(l)})$$

Here $z^{(l)}$ forms a diagonal matrix which acts as a mask to dropout the i^{th} row of $\mathbf{W}^{*(l)}$

- This makes the neural network stochastic since it's structure is partly described by random variables (Bernoulli variables).
- The variance of the output is a measure of the neural network output uncertainty.

Preliminaries

2. Dropout as Bayesian approximation

- Interested in learning the posterior distribution over weight matrices $\mathbf{W}$

  $p(W|\mathbf{X},\mathbf{Y})$ given the training data $\mathbf{X}$ and labels $\mathbf{Y}$, where $W = \{\mathbf{W}^{(l)}\}$

- Then the posterior can be applied to calculate the output distribution y of testing data x through

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

Preliminaries

- Computing the exact posterior distribution is not tractable in a Bayesian Neural Network. Use variational inference instead to approximate posterior distribution $q(\mathbf{W})$.
- Gal et al. proved that, if we select the approximate posterior distribution to be:

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

- We see striking similarity between dropout and approximate posterior distribution. Further works in the paper shows that their objective functions are equivalent.
- During inference, the output mean and variance can be estimated using samples generated with random dropout. More samples need to be collected which also means running the neural network model again.
- Not feasible for edge and mobile computing applications.

ApDeepSense Model

- Interested in an entire probability distribution of output of the layer rather than the expected value of each output at the layer.

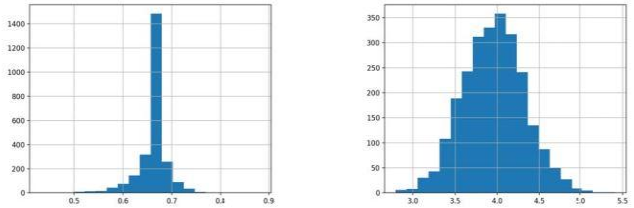
- This is achieved by extending matrix multiplication functions and activation functions

- Select the multivariate Gaussian distribution to approximate the output distribution of each layer.

ApDeepSense Model

1. *The choice of approximation distribution family*

- Feeding the output of one Gaussian process to covariance of the next.
- Train a 20-layer neural network with ReLU and dropout operation to learn the sum of 200 independent Gaussian variables.
- The output distributions of two hidden units in Figure 1 clearly exhibit the shapes of bell curves with different means and variances.



(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.

ApDeepSense Model ctd.

2. *Approximation criteria*
   - Approximate the multivariate Gaussian distribution based on minimizing the Kullbeck-Liebler (KL) divergence between the real an approximate distributions.
   - The main objective function for the approximation is

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

   - The values of μ and σ^2 obtained are $\boldsymbol{\mu = \int p(x)x dx}$ and $\boldsymbol{\sigma^2 = \int p(x)(x-\mu^2) dx}$ which can be viewed as mean and variance matching between $p(x)$ and $q(x)$. These are the required values for the optimal $q(x)$.

ApDeepSense Model ctd.

3. *Approximating matrix multiplication with dropout*
   - The basic matrix multiplication operation with dropout is summarized before as using the Bernoulli variable as a mask for dropout and the Gaussian variables x are independent random variables. We need to find the value the means and variances of output distribution *p(y)*.
   - Mean of output variables is given by $\mathbb{E}[\mathbf{y}_{[j]}] = \mathbb{E}\left[\sum_i \mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}\right] = \sum_i \boldsymbol{\mu}_{[i]}\mathbf{p}_{[i]}\mathbf{W}_{[i,j]}.$ and the variances are given by

$$\mathrm{Var}[\mathbf{y}_{[j]}] = \mathrm{Var}\left[\sum_i \mathbf{x}_{[i]}\mathbf{z}_{[i]}\mathbf{W}_{[i,j]}\right] = \sum_i (\boldsymbol{\mu}_{[i]}^2 + \boldsymbol{\sigma}_{[i]}^2)\mathbf{p}_{[i]}\mathbf{W}_{[i,j]}^2 - \boldsymbol{\mu}_{[i]}^2\mathbf{p}_{[i]}^2\mathbf{W}_{[i,j]}^2.$$

   - These representations can be efficiently computed if they are represented in the form of matrix as shown

$$\begin{aligned} \mathbb{E}[\mathbf{y}] &= (\boldsymbol{\mu} \odot \mathbf{p})\mathbf{W}, \\ \mathrm{Var}[\mathbf{y}] &= ((\boldsymbol{\mu}^2 + \boldsymbol{\sigma}^2) \odot \mathbf{p} - \boldsymbol{\mu}^2 \odot \mathbf{p}^2)\mathbf{W}^2. \end{aligned}$$

ApDeepSense Model ctd.

4. *Approximating activation functions*
   - The non-linear activation functions are approximated to piece-wise linear functions.
   - The linear transformation of Gaussian random variables is well-understood and forms the basis of the proof.
   - The whole axis is divided into parts and there exists a linear activation function on each interval.
   - Using the linear activation functions and slope in each interval, the authors formulate two cases, ($k_p=0$ and $k_p\neq0$)
   - The narrower distributions offer less uncertainty, whereas flatter distributions offer more uncertainty.

Evaluation

- Consider the mean absolute error for accuracy and predictive log-likelihood for correspondence between ground truth and their predicted distribution. Lower values mean higher correspondence.
- Evaluate running time and energy consumption on Intel Edison devices.

a. Testing hardware

- Intel Edison computing platform is powered by Intel Atom SoC dual-core CPU at 500MHz and is equipped with 1GB memory and 4GB flash storage.
- All neural network models run on the CPU during experiments.

Evaluation ctd.

b. Evaluation tasks and datasets

Evaluation is based on 4 tasks which are as follows:

- BPEst : Cuffless BP monitoring
- NYCommute : Commute estimation of NYC
- GasSen : Estimate dynamic gas mixtures from sensors
- HHAR : Heterogeneous human activity recognition

c. Testing models and uncertainty estimation algorithms

- Two pre-trained neural networks with the same structure but different activation functions.
- *DNN-ReLu* and *DNN-Tanh.* As the name suggests these neural networks were trained with ReLu and Tanh activation function respectively.

Evaluation ctd

- Authors compare with two other uncertainty estimation algorithms
- *ApDeepSense* : proposed algorithm
- *MCDrop-k* : sampling-based unbiased uncertainty estimation method for deep neural network with dropout and generated k output samples to use for predicting uncertainties.
- *RDeepSense* : Efficient uncertainty estimation which involves retraining the neural networks. Used as an upper-bound of estimation performance which can be achieved.

d. Model estimation performance

- For each of the earlier mentioned tasks, we discuss model estimation performance
- For regression tasks, MAE and NLL are calculated.
- For classification tasks, ACC and NLL are calculated

Evaluation ctd.

1. BPEst
- The two pre-trained neural networks with ApDeepSense consistently have the lowest NLL values. This shows the approximation method used in ApDeepSense works well in the real dataset.
- ApDeepSense is not the best-performing on the MAE metric. It achieves bias-variance tradeoff by directly approximating the output distribution.

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**

Evaluation ctd.

2. NYCommute
   - Consistent with the trend we see that the ApDeepSense pre-trained neural networks perform way better than the others.
   - *MCDrop-50* involves running the entire neural network model 50 times to obtain 50 samples. This algorithm still ends up having a high NLL value which indicates that it requires even more samples to achieve the same performance as ApDeepSense.

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**

Evaluation ctd.

3. GasSen
   - ApDeepSense still outperforms all the other algoriths for jucertainty estimation with NLL metric.
   - ApDeepSense still achieves a bias-variance trafeoff with better NLL.
   - In the *DNN-Tanh* network, we see that the uncertainty estimation is the clear priority of ApDeepSense.

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**

Evaluation ctd.

4. HHAR
   - This is a classification task.
   - The metrics for measurement are accuracy in percentage(ACC) and negative log likelihood(NLL).
   - The results obtained show that ApDeepSense outperforms the other algorithms in both ACC and NLL metrics.
   - Achieves better classification results and also likelihood estimation.

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**

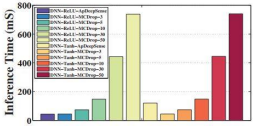
Evaluation ctd.

e. System performance

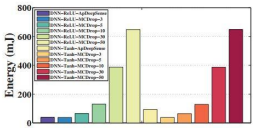
- In this section inference time, energy consumption and model and system performance tradeoff is compared among the various uncertainty estimation algorithms.

- Inference Time and Energy Consumption results
- ApDeepSense saves around 94.1% and 83,6% inference time in average for DNN with ReLu and Tanh activation function respectively.
- ApDeepSense saves around 94.2% and 85.7% energy consumption in average for DNN with ReLu and Tanh activation function respectively.
- On comparison with *MCDrop-50* algorithm we see that ApDeepSense requires just 2 and 7 piecewise linear function to approximate ReLu and Tanh function. This saves the model from running 50 times and saves around 96% and 86% of computations

Results Obtained

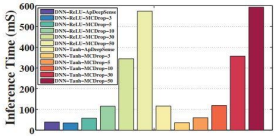


(a) The inference time of the BPEst task.

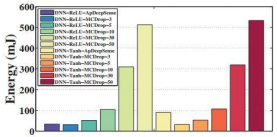


(b) The energy consumption of the BPEst task.

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

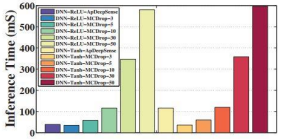


(a) The inference time of the NYCommute task.

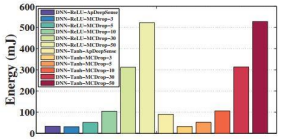


(b) The energy consumption of the NYCommute task.

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

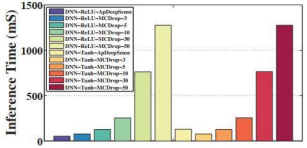


(a) The inference time of the GasSen task.

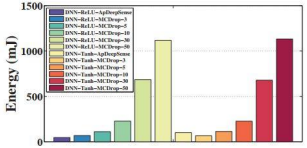


(b) The energy consumption of the GasSen task.

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



(a) The inference time of the HHAR task.



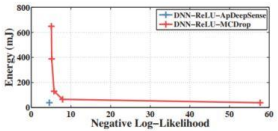
(b) The energy consumption of the HHAR task.

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

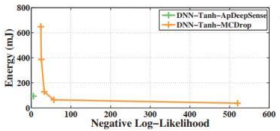
Evaluation ctd.

- Energy Consumption and Quality of Uncertainty Estimation
- We use the NLL metric for representing the quality of the uncertainty estimation.
- Smaller NLL means better uncertainty estimation quality
- Points in the left-bottom corner of these graphs indicate better tradeoffs between energy consumption and uncertainty estimation.
- Indicates using lesser energy to achieve a predictive distribution with lower negative log-likelihood.
- These tradeoffs show that ApDeepSense is in fact an effective and efficient uncertainty estimation algorithm for deep neural networks.
- The authors claim that ApDeepSense is the first energy-efficient test-time uncertainty estimation algorithm for trained deep neural networks deployed on IoT devices.

Results Obtained

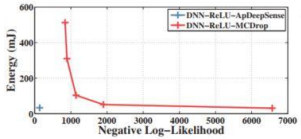


(a) The tradeoff for DNN-ReLU of the BPEst task..

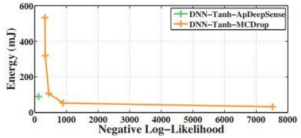


(b) The tradeoff for DNN-Tanh of the BPEst task.

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

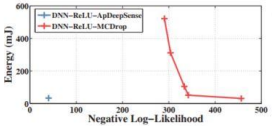


(a) The tradeoff for DNN-ReLU of the NYCommute task..

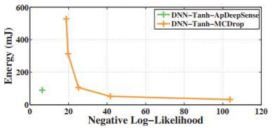


(b) The tradeoff for DNN-Tanh of the NYCommute task.

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

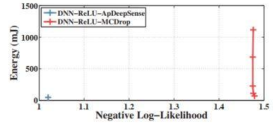


(a) The tradeoff for DNN-ReLU of the GasSen task..

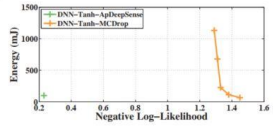


(b) The tradeoff for DNN-Tanh of the GasSen task.

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



(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.

Conclusion and Future Directions

- ApDeepSense does a good job in providing uncertainty estimations for neural networks
- There is no need for structure-changing or re-training of the models.
- Experiments on Intel Edison platform for different tasks showed reduction in inference time and energy consumption to provide uncertainty estimates.
- Supports only fully connected networks, can be extended to support convolutional and recurrent neural networks by replacing the dropout to convolutional or recurrent dropout.
- These dropout operations convert the neural network into a Bayesian neural network.
- Challenges in extending related operations to apply to probabilistic distribution inputs and offer closed-form output distribution using APIs in deep learning libraries.

Q&A Session


Thank
you