Efficient Sparse Processing in Smart Home Applications

Rishikanth Chandrasekaran, Yunhui Guo, Anthony Thomas, Massimiliano Menarini, Michael H Ostertag, Yeseong Kim, Tajana Rosing
University of California San Diego
{r3chandr,yug185,ahthomas,mmenarini,mosterta,yek048,tajana}@ucsd.edu

ABSTRACT

In recent years, smart home technology has become prevalant and important for various applications. A typical smart home system consists of sensing nodes sending raw data to a cloud server which performs inference using a Machine Learning (ML) model trained offline. This approach suffers from high energy and communication costs and raises privacy concerns. To address these issues researchers proposed hierarchy aware models which distributes the inference computations across the sensor network with each node processing a part of the inference. While hierarchical models reduce these overheads significantly they are computationally intensive to run on resource constrained devices which are typical to smart home deployments. In this work we present a novel approach combining Hierarchy aware Neural Networks (HNN) with variational dropout technique to generate sparse models which have low computational overhead allowing them to be run on edge devices with limited resources. We evaluate our approach using an extensive real-world smart home deployment consisting of several edge devices. Measurements across different devices show that without significant loss of accuracy, energy consumption can be reduced by up to 35% over state-of-the-art.

CCS CONCEPTS

• **Computing methodologies** → *Distributed computing methodologies*; • **Computer systems organization** → *Embedded systems*; • **Mathematics of computing** → Information theory.

ACM Reference Format:
Rishikanth Chandrasekaran, Yunhui Guo, Anthony Thomas, Massimiliano Menarini, Michael H Ostertag, Yeseong Kim, Tajana Rosing. 2019. Efficient Sparse Processing in Smart Home Applications. In *1st Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML 2019), November 10, 2019, New York, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3362743.3362963

1 INTRODUCTION

In recent years, smart home technology has evolved from simple functionality such as automated device control to more sophisticated applications like activity recognition, surveillance, smart grid controls and many more. These applications entirely rely on cloud servers to perform Machine Learning(ML) on data generated by smart home sensors. A typical smart home architecture comprises of a hierarchy of sensors, actuators, hubs/gateways where sensors transmit raw data to hubs which aggregates the received data and sends it to the cloud. The cloud performs inference using a pretrained ML model to predict a quantity of interest specific to the application. The Samsung Smart Things ecosystem[11] is one such commercially available example of this architecture.

This monolithic approach has several disadvantages. First, the constant communication of data is energy intensive and reduces the life of battery operated sensor nodes. Second, large volumes of data generated by some sensors eg. cameras, might saturate and slow down the network. This excessive communication could demand provisioning of additional expensive resources. Lastly, smart home sensors record sensitive data capturing various user activities within the home. Sending this raw personal data to a third party vendor raises serious privacy concerns.

To overcome these issues recent work [2] proposed a hierarchical approach which pushes ML inference computations from the cloud to edge devices. In this approach, edge devices run partial ML inference on the sensor values, transforming them into a lower dimensional representation. These partial values are aggregated by the cloud(at the top of the hierarchy) where the final inference result is computed. The authors demonstrate that this approach reduces energy consumption by 63% and network latency by 68%. However, they consider relatively high powered edge devices (RaspberryPi 3[4]) and do not address memory or computation costs for running on resource constrained devices. In order to leverage the advantages of this approach, the computations need to be optimized for resource constrained edge devices typical to smart home applications. This is challenging as the current dominant approach is to use Neural Networks (NN) which are memory and computation heavy. State of the art neural networks for image classification often have millions of parameters and require 100-500MB of RAM for storing the weights alone and more to perform inference computations. However, a micro-controller on a typical edge device often has 128KB of RAM or less.

In this paper, we present a novel implementation combining hierarchy aware inference with variational dropout which can be run on resource constrained IoT edge devices for smart home applications. We use Human Activity Recognition (HAR) as a use case to evaluate our approach. The rest of the paper will discuss our approach in the context of HAR. While our evaluation is specific to HAR, the principles of the approach can be applied to any such application. We evaluate our implementation using a comprehensive real-world smart home deployment. We show that our approach reduces energy by 35% in addition to the reductions achieved by current hierarchical methods[2].

SenSys-ML 2019, November 10, 2019, New York, NY, USA

ACM ISBN 978-1-4503-7011-0/19/11…$15.00
https://doi.org/10.1145/3362743.3362963

2 RELATED WORK

Human Activity Recognition(HAR) in smart homes has been well studied in recent years by the community [12], [14], [10]. However, most works focus only on improving accuracy of activity recognition and use the traditional monolithic approach which as explained earlier has disadvantages. Our work addresses all these issues and we conduct experiments using an extensive real-world deployment with different subjects for evaluation.

Prior work has also explored hierarchy aware learning by pushing down inference computations across the IoT network.[2], [13] implement hierarchy aware machine learning for distributing inference computations on edge devices. However they evaluate their systems on relatively high powered devices which aren't representative of typical edge devices which have fewer resources.

Methods to compress or reduce models have also been explored by the community. The "variational dropout" method proposed in [7] has been shown theoretically to enforce better regularization, and empirically to lead to substantial reductions in the number of non-zero model parameters - up to 280x as reported in [7]. Thus, because variational dropout is able to simultaneously regularize and induce sparsity, we consider this a promising method for implementing efficient deep learning models on resource constrained edge devices.

3 HIERARCHICAL TRAINING METHODOLOGY

We now describe our hierarchy aware methodology in detail. We consider a Smart Home deployment consisting of s sensing devices gathering features $\mathbf{x} = (x_1, ..., x_s)$. For simplicity of notation, we assume each sensor gathers a single feature, thus there is a 1:1 correspondence between sensors s_i and features x_i. Each sensor communicates with one or more of m different aggregator devices E_i. Note that each aggregator device necessarily receives input from multiple sensors. We denote by $\mathbf{x}_i \subset \mathbf{x}$ the subset of features which are locally available on aggregator E_i. Each aggregator computes a function of its input data and emits a lower dimensional representation: $\mathbf{z}_i = g_i(\mathbf{x}_i)$. The resulting compression factor is $|\mathbf{x}_i|/|\mathbf{z}_i|$. In our case, as proposed in [2], $g_i(\mathbf{x}_i)$ is an MLP defined over the locally available data. Here we consider a three tier hierarchy. The output of aggregator devices at the first tier (i.e. the sensors) is used as input to MLPs running in the aggregator devices at the second tier. More formally, an aggregator E_i^2 in the second tier which takes input from $E_j^1, ..., E_k^1$ in the first tier computes:

$$\mathbf{z}_i^2 = g_i^2(\mathbf{z}_j^1, ..., \mathbf{z}_k^1) \quad (1)$$

$$= g_i^2(g_j^1(\mathbf{x}_j, ..., g_k^1(\mathbf{x}_k) \quad (2)$$

This process is repeated until the final layer in the cloud. As discussed in the introduction, we train the model offline using standard SGD and then copy the weights corresponding to each local MLP to the corresponding device during inference.

As discussed above, a potential issue is that the embedded devices acting as aggregators are typically resource constrained and have low power processors which may not be able to deliver satisfactory performance even when using smaller local MLPs. We apply the variational dropout method of [7] to induce sparsity in the local network weights. The method of [7] only *allows* for a sparse solution, it does not guarantee the degree of sparsity. Their method includes a threshold parameter which can be used to truncate small weights to zero and encourage a sparse solution. Section 5 compares the tradeoff between accuracy and sparsity along with resulting savings in energy whilst varying this parameter.

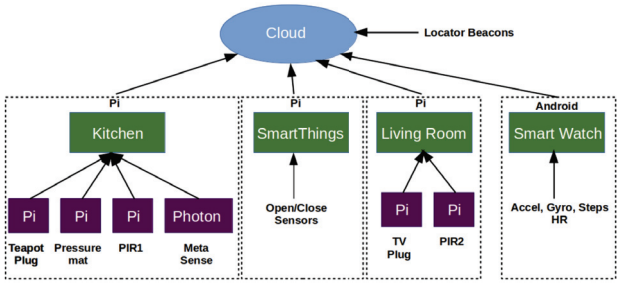

Figure 1: Diagram of connectivity between sensors. Green and purple correspond to local aggregation units

Test Subject	1	2	3	4
Count	28, 840	28, 787	29, 758	28, 599

Table 1: Summary of Analysis Datasets

4 SMART HOME DEPLOYMENT

We conducted a large scale deployment in the home of one of our group members. We deployed a total of twenty-one sensors around the home. Figure 2 presents a floor plan of the deployment environment and the location of sensors therein.

4.1 Application Network Architecture

This section details the hierarchical structure of our deployment and the connectivity between each layer in the hierarchy. Figure 1 presents a high level overview of our hierarchical architecture. During the model training phase, all sensors send their data to the cloud. Once data has been collected, a hierarchy aware model is trained offline and the resulting local partial inference models are deployed to each of the local aggregators in the sensing hierarchy. All communication between devices uses the MQTT protocol [1].

We introduce a "room level aggregator" running on a Raspberry Pi. Each room has several sensors deployed in various locations measuring a particular phenomena of interest. Each sensor connects to a processing node - a Photon, or a Raspberry Pi Zero, depending on the type of interface required by the sensor. This processing node collects data from the sensors and performs partial inference during the testing phase.

Most sensors, such as the smart-plugs, come with on-board micro-controllers that interface with the actual sensor and send the raw readings out to a hub or some other interface. Ideally, our hierarchical models would run on these native computing platforms. However, since many sensors are off the shelf products, we don't usually have access to modify the firmware and hence emulate such local processing by connecting to a Photon or a Raspberry Pi Zero. We use an Intel NUC computer[6] as in-house "cloud server." We

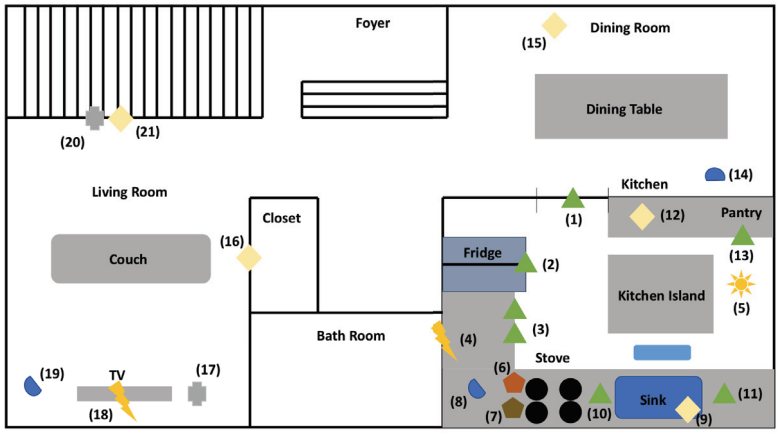

Figure 2: Sensor deployment in smart home. (1) kitchen door contact (2) fridge door contact (3) kitchen drawer contacts 1&2 (4) teapot smart-plug (5) kitchen smart bulb (6) metasense (7) airbeam (8) kitchen angular motion (9) kitchen locator beacon 1 (10) kitchen cabinet contact 1 (11) kitchen cabinet contact 2 (12) kitchen locator beacon 2 (13) kitchen pantry contact (14) dining room multi-sensor (15) dining room locator beacon (16) living room locator beacon 1 (17) living room motion 1 (18) TV smart plug (19) living room angular motion (20) living room motion 2 (21) living room locator beacon 2

Clearing table	Drinking tea	Eating	Making Tea	Prepping Food	Setting Table	Watching TV
805	174	1042	540	416	287	1373

Table 2: Number of instances for each activity

implemented a shared MQTT message bus, which all devices use to send data. Each device in the hierarchy sends its partial output to a specific topic. The corresponding data aggregator in the next level of the hierarchy subscribes to this topic, to obtain the partial inference output.

While we have connected all devices running machine learning computations to a standard WiFi network in our experiments, a real world deployment would support different communication protocols. For example, for the Zigbee and Z-Wave sensors we use a SmartThings Hub to forward their events to the local network and then the MQTT broker to forward the events via WiFi to a Raspberry PI where we execute the classifier. A more efficient approach would be to add a Zigbee and Z-Wave radio to the PI and run the computation where the data is received, this however would deprive us of the useful SmartThings tools to configure and monitor the sensor network. An alternative would have been to run the classifier on the SmartThings Hub. We did not choose this second option because, while it is possible to write a classifier on SmartThings, it would currently run on the SmartThings Cloud, therefore we would not be able to measure power and performance during our experiments. This illustrates an important point: even though local processing may be possible *in principle* software limitations imposed by device manufacturers may preclude it.

4.2 Data Collection

Three rounds of data collection were performed at the test home using a different test subject each time. The test subject performed a series of eight activities: preparing food, cooking food, setting the table, eating food, clearing the table, making tea, drinking tea, and watching TV. While preparing food, the test subject collected ingredients from a kitchen pantry and refrigerator. While cooking, the subject prepared pasta and tomato sauce using the food prepared previously. The subject then set the dining room table and ate the prepared meal at the table. The subject then cleared the table and boiled water to prepare tea using an electric tea kettle. Finally, the subject drank tea and watched TV in the living room. Ground truth labels were gathered manually, recording the timestamp at which the activity changed, and by recording video and audio. Table 2 shows the number of instances of each activity recorded, where each instance of an activity corresponds to multiple windows of sensor values. Since a wide variety of binary and continuous sensors were used, the readings were synced to the highest sampling device. Training and prediction happen over these windows. Table 1 shows the number of individual datapoints collected for each test subject (note that this is different from the number of activity instances recorded.

5 EXPERIMENTAL EVALUATION

In this section, we discuss various experiments we performed for real-time evaluation. We evaluate the system on two key metrics: inference accuracy, energy usage. We compare the performance of traditional Monolithic Neural Network (MNN) with our proposed sparse Hierarchical Neural Network (HNN). The energy required for communication overhead is not factored in the results. This is due to the reasoning that changing the sparsity of the model only affects the computational complexity and does not change the data

rate. Hence any savings in energy due to communication will be the same as seen in [2]. The sparse models provide additional savings over it.

5.1 Inference Accuracy: Monolithic vs Hierarchical

In this section we present an analysis of our proposed methods on the data collected in our home-deployment. Our goal is to illustrate that our proposed hierarchy aware methods can deliver comparable (or superior) accuracy to methods which allow for full communication. We emphasize that the purpose of this work is *not* to improve on state-of-the-art accuracy or methods for activity detection in smart homes. Rather, we merely seek to show that existing techniques can be implemented in a communication efficient manner by making them hierarchy aware.

Figure 3 presents a comparison of accuracy for the monolithic and hierarchy-aware approaches on test subject 3 - for which we were able to obtain the most accurate results. The models were trained on subjects 1 and 2 and subject 3 was used as the held out test set. The X-axis represents different values of $\log(\alpha)$ parameter which controls sparsity in the resulting network. This threshold parameter does not directly control sparsity and is only used to truncate weights which are *likely* to be zero. Increasing it does not necessary result in an actual increase in the number of weights with zero values. The numbers on top of each bar is the sparsity of the resulting network. The number of output units in each "local" MLP running on a distributed device is 2 for devices in the first tier of hierarchy, and 4 in the second tier. Each bar in Figure 3 presents results using the indicated number of hidden units *per hidden layer* in each local network. For example, the blue bar indicates each local network has 16 hidden units per hidden layer, meaning local networks in the first tier of the hierarchy have the architecture $(K, 16, 2)$, where K is the number of input features. In presenting these results, our goal is to better understand the tradeoffs between accuracy, complexity and sparsity in the local MLP model.

5.1.1 Results. Overall, we see that the hierarchy aware approach yields comparable accuracy to the monolithic approach which allows for full communication between the devices. These results validate our central claim that communication efficiency can be improved by training hierarchy aware model without sacrificing a substantial degree of accuracy. We also observe that we are able to sparsify networks substantially while still attaining high accuracy. For example, in the hierarchy aware case for $\alpha = 1$ and with 256 hidden units, we are able to omit almost 87% of the weights in the network. This corresponds to a dense network with only 32 hidden units in the first layer. Unsurprisingly, a greater fraction of weights can be zeroed out for larger networks. This is valuable as it enables us to use larger networks, which learn more complex patterns in data, without dramatically increasing storage requirements. While we again stress that our goal is not to compare the accuracy of our approach with prior work, we remark that our accuracy results are comparable to 73% - 94% achieved by state-of-the-art as in [8].

Reassuringly, we find that increasing the number of hidden units per hidden layer has little (and sometimes negative) effect on classification accuracy in the hierarchy aware case. This means that we can obtain computational efficiency gains by reducing the size of local networks without incurring meaningful losses in accuracy. In section 5.3 we show this reduction in network complexity translates into energy savings. While these results are optimistic for smaller local networks, unfortunately, there are no known robust techniques for selecting model architecture apriori of training. Overall, we take these experiments as evidence that our proposed methods can be deployed on practical applications without substantially losing accuracy relative to the full communication setting.

Table 3: Accuracy VS Sparsity on subject 1 between HNN and MNN.

Model	Threshold	7	6	5	4	3
HNN	Sparsity	0.3469	0.4685	0.4845	0.5305	0.5501
	Accuracy	0.6154	0.5828	0.6384	0.5850	0.6078
MNN	Sparsity	0.2711	0.3624	0.4691	0.4829	0.5947
	Accuracy	0.6703	0.6777	0.6776	0.7398	0.5272

Table 4: Accuracy VS Sparsity on subject 2 between HNN and MNN.

	Threshold	7	6	5	4	3
HNN	Sparsity	0.3523	0.4127	0.4772	0.5669	0.5545
	Accuracy	0.7146	0.4589	0.5775	0.4584	0.3879
MNN	Sparsity	0.2231	0.2972	0.4223	0.4929	0.5521
	Accuracy	0.4242	0.4124	0.4344	0.4348	0.4239

Table 5: Accuracy VS Sparsity on subject 3 between HNN and MNN.

	Threshold	7	6	5	4	3
HNN	Sparsity	0.2814	0.3996	0.4611	0.4933	0.5839
	Accuracy	0.8402	0.7919	0.8270	0.7402	0.7099
MNN	Sparsity	0.1899	0.2969	0.3228	0.4628	0.4458
	Accuracy	0.8126	0.7974	0.7736	0.8240	0.7993

5.2 Activity Recognition Accuracy with Sparse Neural Networks

In this section we evaluate the accuracy of the model for detecting the activities of daily living. In addition to presenting the accuracy of activity recognition, we also aim to show that our sparse models have little to no loss in accuracy compared to monolithic models - with and without sparsity. Tables 3, 4 and 5 show the results of overall accuracy of activity recognition across different levels of sparsity for both Hierarchical and Monolithic models. From empirical analysis we found that the best neural network architecture was 2 hidden layers with 64 hidden units each. All results below are for this configuration of neural network.

For subject 1, highest accuracy the monolithic model achieved was 66%. From Table 3 we can see that sparse MNN seems to perform slightly better overall with highest accuracy of 67%. The sparse HNN achieves a maximum accuracy of 63% at 48% sparsity. For subject 2 the best non-sparse monolithic model achieves an accuracy of 86%. Table 4 shows that HNN performs better than MNN with a highest accuracy of 71% for 35% sparsity. We can also see that the sparse

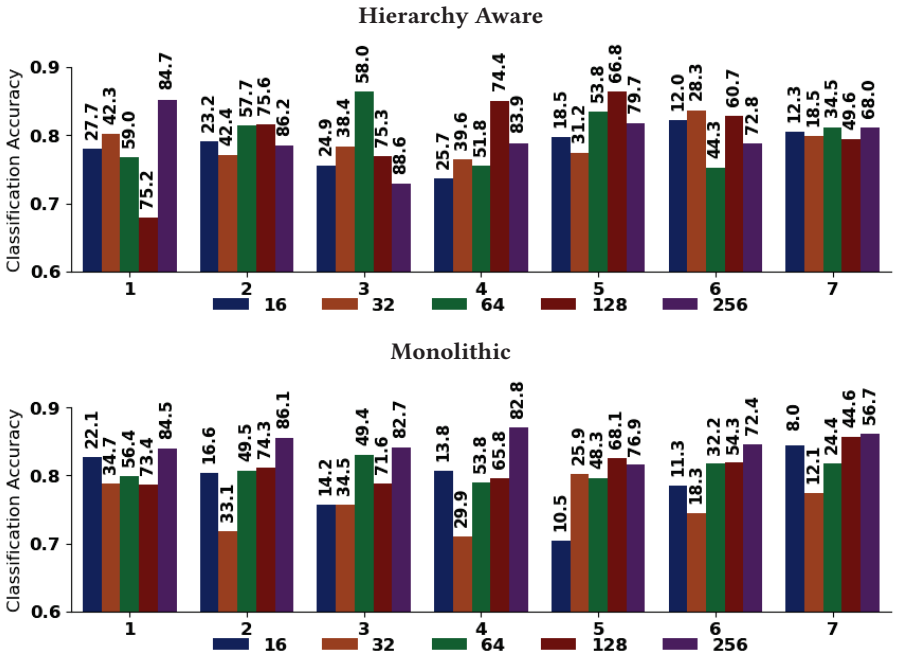

Figure 3: Sparse Monolithic and Sparse Hierarchy-Aware Accuracy for different no of hidden units per hidden layer.

MNN performs significantly worse compared to the monolithic approach with 42% accuracy.

For Subject 6, the baseline accuracy achieved by the non-sparse monolithic model is 82%. Table 5 shows that sparse HNN has better or comparable accuracy to sparse MNN. The sparse HNN has a highest accuracy of 84% for 28% sparsity. However for 46% sparsity it achieves an accuracy of 82% which gives better overall trade-off between accuracy, energy and communication costs.

5.2.1 Discussion. Overall we observe that sparse HNN performed better for subjects 2 and 6, while for subject 1 sparse MNN performed slightly better. Figure 4 shows the energy reduction achieved for different sparsity for the best architecture (64 hidden units per layer) chosen from training. The lowest and highest sparsity that yielded the best accuracy for any subject is marked by vertical lines. We can observe that for 35-48% sparsity we get 25-35% energy savings with comparable accuracy. While the analysis was done for activity recognition in a smart home, the principles are applicable for any IoT wireless sensor network. This validates that sparse hierarchical neural networks could reduce energy usage.

5.3 Energy Usage

In this section we analyze the trade-offs between sparsity and energy usage. We test our implementation on two embedded devices which are representative of typical IoT devices you might see in use: Raspberry Pi Zero[5], Photon[9]. The Raspberry Pi Zero has a 1GHz single-core CPU with 512MB RAM and the Photon features a 120MHz ARM Cortex M3 with 128KB RAM. In addition, we try to implement our sparse model on an Arduino UNO[3] which has extremely small computational power. The UNO uses an ATmega328P

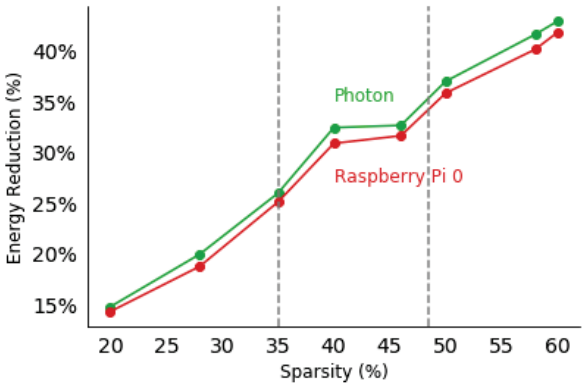

Figure 4: Monolithic and Hierarchy-Aware Accuracy

microcontroller which runs at 16MHz and has 2KB of RAM. The energy used for an inference is directly proportional to the number of computations performed. This is influenced by number of hidden units and sparsity of the weights. Below, we analyze the trade-offs between energy, sparsity and number of hidden units. We use a neural network with 2 hidden layers, for evaluation. The input and output layers have 4 units each.

5.3.1 Effect of size of neural network. For this analysis we fix sparsity at 45% and vary the number of hidden units per layer. The sparsity was chosen to be 45 as it strikes an ideal balance between accuracy and reducing energy. Figure 5 shows the amount of energy saved in comparison to a hierarchical model without sparsity

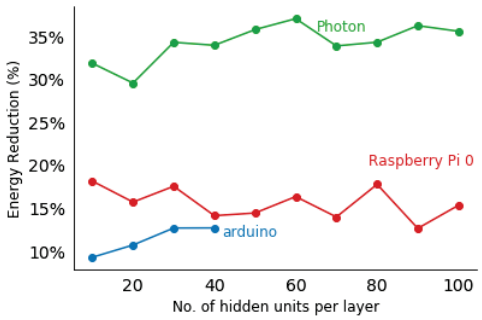

Figure 5: Energy Reduction vs Number of hidden units

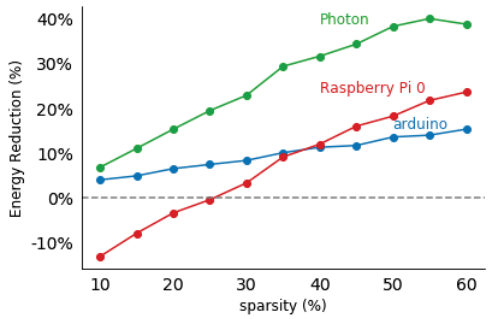

Figure 6: Energy reduction (%) for different levels of sparsity

for performing 120000 inference computations. The Arduino UNO runs out of memory for more than 40 hidden units.

Although the results don't show an overall trend, there's a drop in energy savings across both the Pi Zero and the Photon towards the end. This is explained by the fact that when sparsity is constant and the number of hidden units is increased, the number of non-zero weights increases. For 45% sparsity beyond 50 hidden units per layer, the effect of the increased number of hidden units dominates over sparsity. It is important to note that despite this, the approach produces 30-35% (Photon) and 12-15% (Pi Zero) energy reductions.

5.3.2 Effect of sparsity. We fix number of hidden units per layer at 30 and vary sparsity. The energy used for performing 120000 inference computations is measured. We observe from Figure 6 that energy savings increases with sparsity across all three devices. For a fixed number of hidden units, when sparsity is increased, more weights become zero and hence computations per inference reduces. The figure shows that for Pi Zero the sparse model is more expensive when sparsity is below 25%. This is because, the CSR based implementation is more expensive and inefficient for dense matrices. The traditional approach takes advantage of cache locality which reduces memory accesses required. After 25% sparsity, the CSR based approach is clearly more efficient as it only performs computations for non zero weights. The photon which doesn't have an internal cache doesn't show this trend as all memory accesses have the same cost and CSR implementation reduces the number of accesses. This validates our expectations that using a sparse hierarchical model will produce additional energy savings.

5.4 Drawbacks

Our work addresses the use case where the model uses heterogeneous input consisting of different kinds of sensor data having different characteristics. This is in contrast to federated learning which deals with the case where each node process the same kind of data(eg. camera images). A potential pitfall in our approach is that, an error or malfunction of a single sensor/node could adversely affect the final prediction. We seek to explore in the future, methods to make the model more robust to such failures. A potential approach would be to propagate noise/simulated error during training.

6 CONCLUSION

With rising importance of smart home applications, it is important to efficiently perform inference on edge devices with limited resources. In this work we proposed a new method which combines hierarchy aware networks with variational dropout technique to produce sparse hierarchical neural networks. These sparse models have very low overhead and can be run on resource constrained edge devices. We evaluated our approach on a real-world smart home deployment consisting of various sensors. We analyzed the various trade-offs between energy consumption, network size and level of sparsity across three different low resource devices. Our measurements on three different edge devices show that we can reduce energy consumption by up to 35% with little loss in accuracy.

7 ACKNOWLEDGEMENTS

This work was partially funded by grants from Samsung, SRC, NSF and KACST.

REFERENCES

[1] The MQTT messaging protocol. http://mqtt.org/
[2] Anthony Thomas, Y.G., Yeseong Kim, B.A., Arun Kumar, T.S.R.: Hierarchical and distributed machine learning inference beyond the edge. IEEE International Conference on Networking Systems and Control (ICNSC) (2019)
[3] Arduino: Arduino uno specifications. https://store.arduino.cc/usa/arduino-uno-rev3
[4] Foundation, R.: Raspberry pi 3 specifications. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[5] Foundation, R.P.: Raspberry pi zero w specifications. https://www.raspberrypi.org/products/raspberry-pi-zero-w/
[6] Intel: Intel nuc specifications. https://www.intel.com/content/www/us/en/support/articles/000005545/mini-pcs.html
[7] Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2498–2507. JMLR. org (2017)
[8] Ordonez, F.J., Roggen, D.: Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. Sensors **16**(1) (2016). https://doi.org/10.3390/s16010115, http://www.mdpi.com/1424-8220/16/1/115
[9] Particle: Particle photon specifications. https://docs.particle.io/datasheets/wi-fi/photon-datasheet/
[10] Roy, N., Misra, A., Cook, D.: Ambient and smartphone sensor assisted adl recognition in multi-inhabitant smart environments. Journal of ambient intelligence and humanized computing **7**(1), 1–19 (2016)
[11] Samsung: Samsung smartthings ecosystem. https://www.smartthings.com/
[12] Tapia, E.M., Intille, S.S., Larson, K.: Activity recognition in the home using simple and ubiquitous sensors. In: Ferscha, A., Mattern, F. (eds.) Pervasive Computing. pp. 158–175. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
[13] Teerapittayanon, S., McDanel, B., Kung, H.: Distributed deep neural networks over the cloud, the edge and end devices. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 328–339. IEEE (2017)
[14] Wang, J., Chen, Y., Hao, S., Peng, X., Hu, L.: Deep learning for sensor-based activity recognition: A survey. Pattern Recognition Letters (2019), deep Learning for Pattern Recognition