Online Human Activity Recognition using Low-Power Wearable Devices

Ganapati Bhat[1], Ranadeep Deb[1], Vatika Vardhan Chaurasia[1], Holly Shill[2], Umit Y. Ogras[1]
[1]School of Electrical Computer and Energy Engineering, Arizona State University, Tempe, AZ
[2]Lonnie and Muhammad Ali Movement Disorder Center, Phoenix, AZ

ABSTRACT

Human activity recognition (HAR) has attracted significant research interest due to its applications in health monitoring and patient rehabilitation. Recent research on HAR focuses on using smartphones due to their widespread use. However, this leads to inconvenient use, limited choice of sensors and inefficient use of resources, since smartphones are not designed for HAR. This paper presents the *first HAR framework that can perform both online training and inference.* The proposed framework starts with a novel technique that generates features using the fast Fourier and discrete wavelet transforms of a textile-based stretch sensor and accelerometer data. Using these features, we design a neural network classifier which is trained online using the policy gradient algorithm. Experiments on a low power IoT device (TI-CC2650 MCU) with nine users show 97.7% accuracy in identifying six activities and their transitions with less than 12.5 mW power consumption.

1 INTRODUCTION

Advances in wearable electronics has potential to disrupt a wide range of health applications [11, 23]. For example, diagnosis and follow-up for many health problems, such as motion disorders, depend currently on the behavior observed in a clinical environment. Specialists analyze gait and motor functions of patients in a clinic, and prescribe a therapy accordingly. As soon as the patient leaves the clinic, there is no way to continuously monitor the patient and report potential problems [13, 27]. Another high-impact application area is obesity related diseases, which claim about 2.8 million lives every year [2, 4]. Automated tracking of physical activities of overweight patients, such as walking, offers tremendous value to health specialists, since self recording is inconvenient and unreliable.

There has been growing interest in human activity recognition with the prevalence of low cost motion sensors and smartphones. For example, accelerometers in smartphones are used to recognize activities such as stand, sit, lay down, walking, and jogging [3, 16, 20]. This information is used for rehabilitation instruction, fall detection of elderly, and reminding users to be active [18, 35]. Furthermore, activity tracking also facilitates physical activity, which improves the wellness and health of its users [8, 9, 19]. HAR techniques can be broadly classified based on when training and inference take place. Early work collects the sensor data before processing. Then, both classifier design and inference are performed offline [5]. Hence, they have limited applicability. Most recent work trains a classifier offline, but processes the sensor data online to infer the activity [3, 31]. However, to date, there is no technique that can perform *both online training and inference.* Online training is crucial, since it needs to adapt to new, and potentially large number of, users who are not involved in the training process. To this end, *this paper presents the first HAR technique that continues to train online to adapt to its user.*

The vast majority, if not all, of recent HAR techniques employ smartphones. Major motivations behind this choice are their widespread use and easy access to integrated accelerometer and gyroscope sensors [35]. We argue that smartphones are not suitable for HAR for three reasons. First, patients cannot always carry a phone as prescribed by the doctor. Even when they have the phone, it is not always in the same position (e.g., at hand or in pocket), which is typically required in these studies [10, 31]. Second, mobile operating systems are not designed for meeting real-time constraints. For example, the Parkinson's Disease Dream Challenge [1] organizers shared raw motion data collected using iPhones in more than 30K experiments. According to the official spec, the sampling frequency is 100 Hz. However, the actual sampling rate varies from 89 Hz to 100 Hz, since the phones continue to perform many unintended tasks during the experiments. Due to the same reason, the power consumption is in the order of watts (more than 100× of our result). Finally, researchers are limited to sensors integrated in the phones, which are not specifically designed for human activity recognition.

This paper presents an *online* human activity recognition framework using the wearable system setup shown in Figure 1. The proposed solution is the first to perform online training and leverage textile-based stretch sensors in addition to commonly used accelerometers. Using the stretch sensor is notable, since it provides low-noise motion data that enables us to segment the raw

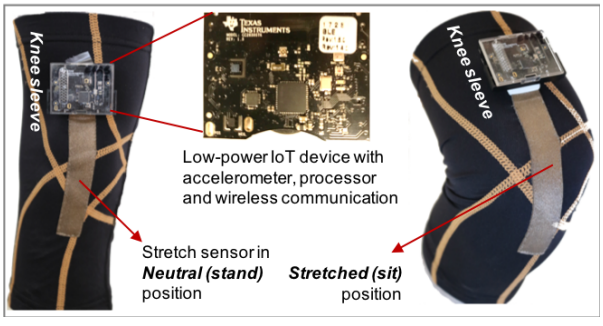

Figure 1: Wearable system setup, sensors and the low-power IoT device [34]. We knitted the textile-based stretch sensor to a knee sleeve to accurately capture the leg movements

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICCAD '18, November 5–8, 2018, San Diego, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240833

data in non-uniform windows ranging from one to three seconds. In contrast, prior studies are forced to divide the sensor data into fixed windows [4, 20] or smoothen noisy accelerometer data over long durations [10] (detailed in Section 2). After segmenting the stretch and accelerometer data, we generate features that enable classifying the user activity into walking, sitting, standing, driving, lying down, jumping, as well as transitions between them. Since the stretch sensor accurately captures the periodicity in the motion, its fast Fourier transform (FFT) reveals invaluable information about the human activity in different frequency bands. Therefore, we judiciously use the leading coefficients as features in our classification algorithm. Unlike the stretch sensor, the accelerometer data is notoriously known to be noisy. Hence, we employ the approximation coefficients of its discrete wavelet transform (DWT) to capture the behavior as a function of time. We evaluate the performance of these features for HAR using commonly used classifiers including neural networks, random forest, and k-nearest neighbor (k-NN). Among these, we focus on neural networks, since it enables online reinforcement learning (RL) using policy gradient [33] with low implementation cost. Finally, this work is the first to provide a detailed power consumption and performance break-down of sensing, processing and communication tasks. We implement the proposed framework on the TI-CC2650 MCU [34], and present an extensive experimental evaluation using data from nine users and a total of 2614 activity windows. Our approach provides 97.7% overall recognition accuracy with 27.60 ms processing time, 1.13 mW sensing and 11.24 mW computation power consumption.

The major contributions of this work are as follows:

- A novel technique to segment the sensor data non-uniformly as a function of the user motion,
- Online inference and training using a neural network, and reinforcement learning based on policy gradient,
- A low power implementation on a wearable device and extensive experimental evaluation of accuracy, performance and power consumption using nine users.

The rest of the paper is organized as follows. We review the related work in Section 2. Then, we present the feature generation and classifier design techniques in Section 3. Online learning using policy gradient algorithm is detailed in Section 4. Finally, the experimental results are presented in Section 5, and our conclusions are summarized in Section 6.

2 RELATED WORK AND NOVELTY

Human activity recognition has been an active area of research due to its applications in health monitoring, patient rehabilitation and in promoting physical activity among the general population [4, 7, 8]. Advances in sensor technology have enabled activity recognition to be performed using body mounted sensors [29]. Typical steps for activity recognition using sensors include data collection, segmentation, feature extraction and classification.

HAR studies typically use a fixed window length to infer the activity of a person [4, 20]. For instance, the studies in [4, 20] use 10 second windows to perform activity recognition. Increasing the window duration improves accuracy [7], since it provides richer data about the underlying activity. However, transitions between different activities cannot be captured with long windows. Moreover, fixed window lengths rarely capture the beginning and end of an activity. This leads to inaccurate classification as the window can have features of two different activities [7]. A recent work proposes action segmentation using step detection algorithm on the accelerometer data [10]. Since the accelerometer data is noisy, they need to smoothen the data using a one-second sliding window with 0.5 second overlap. Hence, this approach is not practical for low-cost devices with limited memory capacity. Furthermore, the authors state that there is a strong need for better segmentation techniques to improve the accuracy of HAR [10]. To this end, we present a robust segmentation technique which produces windows whose sizes vary as a function of the underlying activity.

Most existing studies employ statistical features such as mean, median, minimum, maximum, and kurtosis to perform HAR [4, 20, 28]. These features provide useful insight, but there is no guarantee that they are representative of all activities. Therefore, a number of studies use all the features or choose a subset of them through feature selection [28]. Fast Fourier transform and more recently discrete wavelet transform have been employed on accelerometer data. For example, the work in [10] computes the 5th order DWT of the accelerometer data. Eventually, it uses only a few of the coefficients to calculate the wavelet energy in the 0.625 - 2.5 Hz band. In contrast, we use only the approximation coefficients of a single level DWT with $O(N/2)$ complexity. Unlike prior work, we do not use the FFT of the accelerometer data, since it entails significant high frequency components without clear implications. In contrast, we employ leading FFT coefficients of the stretch sensor data, since it gives a very good indication of the underlying activity.

Early work on HAR used wearable sensors to perform data collection while performing various activities [5]. This data is then processed offline to design the classifier and perform the inference. However, offline inference has limited applicability since users do not get any real time feedback. Therefore, recent work on HAR has focused on implementation on smartphones [3, 8, 17, 31]. Compared to wearable HAR devices, smartphones have limited choice of sensors and high power consumption. In addition, results on smartphones are harder to reproduce due to the variability in different phones, operating systems and usage patterns [9, 31].

Finally, existing studies on HAR approaches employ commonly used classifiers, such as k-NN [14], support vector machines [14], decision trees [30], and random forest [14], which are *trained offline*. In strong contrast to these methods, the proposed framework is the first to enable *online training*.

3 FEATURE SET AND CLASSIFIER DESIGN

3.1 Goals and Problem Statement

The goal of the proposed HAR framework is to recognize the six common daily activities listed in Table 1 and the transitions between them *in real-time* with *more than 90% accuracy under mW power range*. These goals are set to make the proposed system practical for daily use. The power consumption target enables day-long operation using ultrathin lithium polymer cells [12].

Table 1: List of activities used in the HAR framework

• Drive (D)	• Jump (J)	• Lie Down (L)
• Sit (S)	• Stand (Sd)	• Walk (W)
• Transition (T) between the activities		

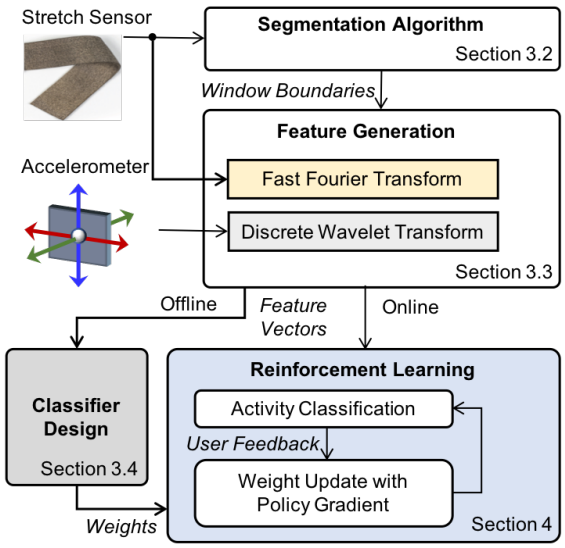

Figure 2: Overview of the proposed HAR framework

The stretch sensor is knitted to a knee sleeve, and the IoT device with a built-in accelerometer is attached to it, as shown in Figure 1. All the processing outlined in Figure 2 is performed locally on the IoT device. More specifically, the streaming stretch sensor data is processed to generate segments ranging from one to three seconds (Section 3.2). Then, the raw accelerometer and stretch data in each window are processed to produce the features used by the classifier (Section 3.3). Finally, these features are used for both online inference (Section 3.4) and reinforcement learning using policy gradient (Section 4). Since communication energy is significant, *only the recognized activity and time stamps* are transmitted to a gateway, such as a phone or PC, using Bluetooth whenever they are nearby (within 10m). The following sections provide a theoretical description of the proposed framework without tying them to specific parameters values. These parameters are chosen to enable a low-overhead implementation using streaming data. The actual values used in our experiments are summarized in Section 5.1 while describing the experimental setup.

3.2 Sensor Data Segmentation

Activity windows should be sufficiently short to catch transitions and fast movements, such as fall and jump. However, short windows can also waste computation time and power for idle periods, such as sitting. Furthermore, a fixed window may contain portions of two different activities, since perfect alignment is not possible. Hence, activity-based segmentation is necessary to maintain a high accuracy with minimum processing time and power consumption.

To illustrate the proposed segmentation algorithm, we start with the snapshot in Figure 3 from our user studies. Both the 3-axis accelerometer and stretch sensor data are preprocessed using a moving average filter similar to prior studies. The unit of acceleration is already normalized to gravitational acceleration. The stretch sensor outputs a capacitance value which changes as a function of its state. This value ranges from around 390 pF (neutral) to close to 500 pF when it is stretched [24]. Therefore, we normalize the stretch sensor output by subtracting its neutral value and scaling by a constant: $s(t) = [s_{raw}(t) - min(s_{raw})]/S_{const}$. We

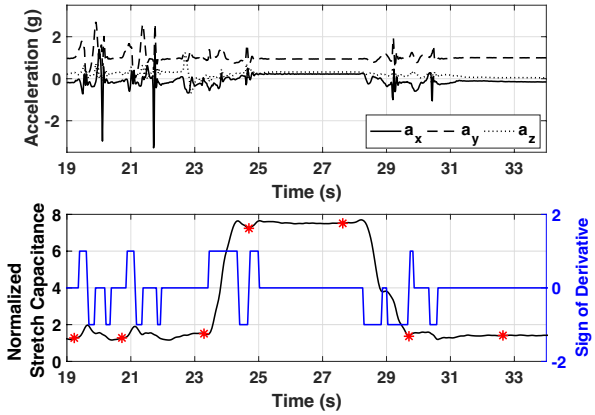

Figure 3: Illustration of the segmentation algorithm

adopted $S_{const} = 8$ to obtain a comparable range to accelerometer readings. First, we note that the 3-axis accelerometer data exhibits significantly larger variations compared to the normalized stretch capacitance. Therefore, decisions based on accelerations are prone to false hits [10]. In contrast, we propose a robust solution which generates the segments specified with red $*$ markers in Figure 3.

The boundaries between different activities can be identified by detecting the deviation of the stretch sensor from its neutral value. For example, the first segment in Figure 3 corresponds to a step during walk. The sensor value starts increasing from a local minimum to a peak at the beginning of the step. The beginning of the second segment ($t \approx 21$ s) exhibits similar behavior, since it is another step. Although the second step is followed by a longer neutral period (the user stops and sits to a chair at $t \approx 23$ s), the beginning of the next segment is still marked by a rise from a local minimum. In general, we can observe a distinct minimum (fall followed by rise as in walk) or a flat period followed by rise (as in walk to sit) at the boundaries of different activity windows. Therefore, the proposed segmentation algorithm monitors the derivative of the stretch sensor to detect the activity boundaries.

We employ the 5-point derivative formula given below to track the trend of the sensor value:

$$s'(t) = \frac{s(t-2) - 8s(t-1) + 8s(t+1) - s(t+2)}{12} \quad (1)$$

where $s(t)$ and $s'(t)$ are the stretch sensor value and its derivative time step t, respectively. When the derivative is positive, we know that the stretch value is *increasing*. Similarly, a negative value means a decrease, and $s'(t) = 0$ implies a flat region. Looking at a single data point can catch sudden peaks and lead to false alarms. To improve the robustness, one can look at multiple consecutive data points before determining the trend. In our implementation, we conclude that the *trend* changes only if the last three derivatives consistently signal the new trend. For example, if the current trend is flat, we require that the derivative is positive for three consecutive data points to filter glitches in the data point. Whenever we detect that the trend changes from flat or decreasing to positive, we produce a new segment. Finally, we bound the window size from below and above to prevent excessively short or long windows. We start looking for a new segment, only if a minimum duration (one second in this work) passes after starting a new window. Besides

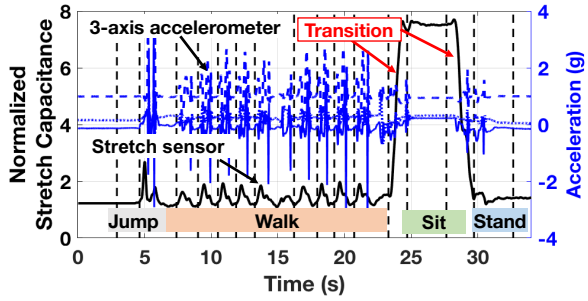

Figure 4: Illustration of the sensor data segmentation

preventing unnecessarily small segments, this approach saves computation time. Similarly, a new segment is generated automatically after exceeding an upper threshold. This choice improves robustness in case a local minimum is missed. We use $t_{max} = 3$ s as the upper bound, since it is long enough to cover all transitions.

Figure 4 shows the segmented data for the complete duration of the illustrative example given in Figure 3. The proposed approach is able to clearly segment each step of walk. Moreover, it is able to capture the transitions from walking to sitting and sitting to standing very well. This segmentation allows us to extract meaningful features from the sensor data, as described in the next section.

3.3 Feature Generation

To achieve a high classification accuracy, we need to choose representative features that capture the underlying movements. We note that human movements typically do not exceed 10-Hz. Since statistical features, such as mean and variance, are not necessarily representative, we focus on FFT and DWT coefficients, which have clear frequency interpretations. Prior studies typically choose the largest transform coefficients [31] to preserve the maximum signal power as in compression algorithms. However, sorting loses the frequency connotation, besides using valuable computational resources. Instead, we focus on the coefficients in the frequency bins of interest by preserving the number of data samples in each segment, as described next.

Stretch sensor features: The stretch sensor shows a periodic pattern for walking, and remains mostly constant during sitting and standing, as shown in Figure 4. As the level of activity changes, the segment duration varies in the (1,3] second interval. We can preserve 10 Hz sampling rate for the longest duration (3 s during low activity), if we maintain $2^5 = 32$ data samples per segment. As the level of activity intensifies, the sampling rate grows to 32 Hz, which is sufficient to capture human movements. We choose a power of 2, since it enables efficient FFT computation in real-time. When the segment has more than 32 samples due to larger sensor sampling rate, we first sub-sample and smooth the input data as follows:

$$s_s[k] = \frac{1}{2S_R} \sum_{i=-S_R}^{S_R} s(tS_R + i), \qquad 0 \le k < 32 \tag{2}$$

where $S_R = \lfloor N/32 \rfloor$ is the subsampling rate, and $s_s[k]$ is the subsampled and smoothed data point. When there are less than 32 samples, we simply pad the segment with zeros.

After standardizing the size, we take the FFT of the current window and the previous window. We use two windows as it allows us to capture any repetitive patterns in the data. With 32 Hz sampling rate during high activity regions, we cover $F_s/2$ =16 Hz activity per Nyquist theorem. We observe that the leading 16 FFT coefficients, which cover the [0-8] Hz frequency range, carry most of the signal power in our experimental data. Therefore, they are used as features in our classifiers. The level of the stretch sensor also gives useful information. For instance, it can reliably differentiate sit from stand. Hence, we also add the minimum and maximum value of the stretch sensor to the feature set.

Accelerometer features: Acceleration data contains faster changes compared to the stretch data, even though the underlying human motion is slow. Therefore, we sub-sample and smoothen the acceleration to $2^6 = 64$ points following the same procedure given in Equation 2. Three axis accelerometers provide acceleration a_x, a_y and a_z along $x-$, $y-$ and $z-$axes, respectively. In addition, we compute the body acceleration excluding the effect of gravity g as $b_{acc} = \sqrt{a_x^2 + a_y^2 + a_z^2} - g$, since it carries useful information.

Discrete wavelet transform is an effective method to recursively divide the input signal to approximation A_i and detail D_i coefficients. One can decompose the input signal to $\log_2 N$ samples where N is the number of data points. After one level of decomposition, A_1 coefficients in our data correspond to 0-32 Hz, while D_1 coefficients cover 32-64 Hz band. Since the former is more than sufficient to capture acceleration due to human activity, we only compute and preserve A_1 coefficients with $O(N/2)$ complexity. The number of features could be further reduced by computing the lower level coefficients and preserving largest ones. As shown in the performance breakdown in Table 5, using the features in the neural network computations takes less time than computing the DWT coefficients. Moreover, keeping more coefficients and preserving the order maintains the shape of the underlying data.

Feature Overview: In summary, we use the following features:
Stretch sensor: We use 16 FFT coefficients, the minimum and maximum values in each segment. This results in 18 features.
Accelerometer: We use 32 DWT coefficients for a_x, a_z and b_{acc}. In our experiments, we use only the mean value of a_y, since no activity is expected in the lateral direction, and b_{acc} already captures its effect given the other two directions. This results in 97 features.
General features: The length of the segment also carries important information, since the number of data points in each segment is normalized. Similarly, the activity in the previous window is useful to detect transitions. Therefore, we also add these two features to obtain a total of 117 features.

3.4 Supervised Learning for State Classification

In the offline phase of our framework, the feature set is assigned a label corresponding to the user activity. Then, a supervised learning technique takes the labeled data to train a classifier which is used at runtime. Since one of our major goals is online training using reinforcement learning, we employ a cost-optimized neural network (NN). *We also compare our solution to most commonly used classifiers by prior work, and provide brief explanations.*

Support Vector Machine (SVM): SVM [14] finds a hyperplane that can separate the feature vectors of two output classes. If a separating hyperplane does not exist, SVM maps the data into higher dimensions until a separating hyperplane is found. Since SVM is a two-class classifier, multiple classifiers need to be trained for recognizing more than two output classes. Due to this, SVM is

not suitable for reinforcement learning with multiple classes [21], which is the case in our HAR framework.

Random Forests and Decision Trees: Random forests [14] use an ensemble of tree-structured classifiers, where each tree independently predicts the output class as a function of the feature vector. Then, the class which is predicted most often is selected as the final output class. C4.5 decision tree [30] is another commonly used classifier for HAR. Instead of multiple trees, C4.5 uses a single tree. Reinforcement learning using random forests has been recently investigated in [26]. As part of the reinforcement learning process, additional trees are constructed and then a subset of trees is chosen to form the new random forest. This adds additional processing and memory requirements on the system, making it unsuitable for implementation on a wearable system with limited memory.

k-Nearest Neighbors (k-NN): k-Nearest Neighbors [14] is one of the most popular techniques used by many previous HAR studies. k-NN evaluates the output class by first calculating k nearest neighbors in the training dataset. Then, it chooses the class that is most common among the k neighbors and assigns it as the output class. This requires storing all the training data locally. Since storing the training data on a wearable device with limited memory is not feasible, k-NN is not suitable for online training.

Proposed NN Classifier: We use the neural network shown in Figure 5 as our classifier. The input layer processes the features denoted by $\mathbf{X}$, and relay to the hidden layer with the ReLU activation. It is important to choose an appropriate number of neurons (N_h) in the hidden layer to have a good accuracy, while keeping the computational complexity low. To obtain the best trade-off, we evaluate the recognition accuracy and memory requirements as a function of neurons, as detailed in Section 5.2.

The output layer includes a neuron for each activity $a_i \in \mathbf{A} = \{D, J, L, S, Sd, W, T\}, 1 \leq i \leq N_A$, where N_A is the number of activities in set $\mathbf{A}$, which are listed in Table 1. Output neuron for activity a_i computes $O_{a_i}(\mathbf{X}, \theta_{in}, \theta)$ as a function of the input features $\mathbf{X}$, and the neural network weights $\{\theta_{in}, \theta\}$. To facilitate the policy gradient approach described in Section 4, we express the output O_{a_i} in terms of the hidden layer outputs as:

$$O_{a_i}(\mathbf{X}, \theta_{in}, \theta) = O_{a_i}(\mathbf{h}, \theta) = \sum_{j=1}^{N_h+1} h_j \theta_{j,i}, \ 1 \leq i \leq N_A \qquad (3)$$

where h_j is the output of the j^{th} neuron in the hidden layer, and $\theta_{j,i}$ is the weight from j^{th} neuron to output activity a_i. Note that h_j is a function of $\mathbf{X}$ and θ_{in}. The summation goes to $N_h + 1$, since there are N_h neurons and one bias term in the hidden layer.

After computing the output functions, we use the softmax activation function to obtain the probability of each activity:

$$\pi(a_i|\,\mathbf{h}, \theta) = \frac{e^{O_{a_i}(\mathbf{h},\theta)}}{\sum_{j=1}^{N_A} e^{O_{a_j}(\mathbf{h},\theta)}}, \ 1 \leq i \leq N_A \qquad (4)$$

We express $\pi(a_i|\,\mathbf{h}, \theta)$ as a function of the hidden layer outputs $\mathbf{h}$ instead of the input features, since our reinforcement learning algorithm will leverage it. Finally, the activity which has the maximum probability is chosen as the output.

Implementation cost: Our optimized classifier requires 264 multiplications for the FFT of stretch data, $118N_h + (N_h + 1)N_A$ multiplications for the NN and uses only 2 kB memory.

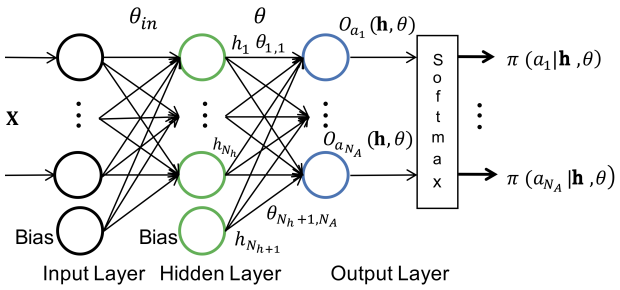

Figure 5: The NN used for activity classification and RL.

4 ONLINE LEARNING with POLICY GRADIENT

The trained NN classifier is implemented on the IoT device to recognize the human activities in real-time. In addition to online activity recognition, we employ policy gradient based reinforcement learning (RL) to continue training the classifier in the field. Online training improves the recognition accuracy for new users by as much as 33%, as demonstrated in our user studies. We use the following definitions for the state, action, policy, and the reward.

State: Stretch sensor and accelerometer readings within a segment are used as the continuous state space. We process them as described in Section 3.3 to generate the input feature vector $\mathbf{X}$ (Figure 5).

Policy: The NN processes input features as shown in Figure 5 to generate the hidden layer outputs $\mathbf{h} = \{h_j, 1 \leq j \leq N_h + 1\}$ and the activity probabilities $\pi(a_i|\mathbf{h}, \theta)$, i.e., the policy given in Equation 4.

Action: The activity performed in each sensor data segment is interpreted as the action in our RL framework. It is given by $argmax\ \pi(a_i|\mathbf{h}, \theta)$, i.e., the activity with maximum probability.

Reward: Online training requires user feedback, which is defined as the reward function. When no feedback is provided by the user, the weights of the network remain the same. The user can give feedback upon completion of an activity, such as walking, which contains multiple segments (i.e., non-uniform action windows). If the classification in this period is correct, a positive reward (in our implementation +1) is given. Otherwise, the reward is negative (−1). We define the sequence of segments for which a reward is given as an *epoch*. The set of epochs in a given training session is called an *episode* following the RL terminology [33].

Objective: The value function for a state is defined as the total reward that can be earned starting from that state and following the given policy until the end of an episode. Our objective is to maximize the total reward $J(\theta)$ as a function of the classifier weights.

Proposed Policy Gradient Update: In general, all the weights in the policy network can be updated after an epoch [33]. This is useful when we start with an untrained network with random weights. When a policy network is trained offline as in our example, its first few layers generate broadly applicable intermediate features [22]. Consequently, we can update only the weights of the output layer to take advantage of offline training and minimize the computation cost. More precisely, we update the weights denoted by θ in Figure 5 to tune our optimized NN to individual users.

Since we use the value function as the objective, the gradient of $J(\theta)$ is proportional to the gradient of the policy [33]. Using this result, the update equation for θ is given as:

$$\theta_{t+1} \doteq \theta_t + \alpha r_t \frac{\nabla_\theta \pi(a_t|\,\mathbf{h}, \theta_t)}{\pi(a_t|\,\mathbf{h}, \theta_t)}, \quad \alpha : \text{Learning rate} \qquad (5)$$

where θ_t and θ_{t+1} are the current and updated weight matrices, respectively. Similarly, a_t is the current action at time t, r_t is the corresponding reward, and $\mathbf{h}$ denotes the hidden layer outputs. Hence, we need to compute the gradient of the policy to update the weights. To facilitate this computation and partial update, we partition the weights into two disjoint sets as $\mathcal{S}_t$ and $\overline{\mathcal{S}_t}$. The weights that connect to the output O_{a_t} corresponding to the current action are in $\mathcal{S}_t$. The rest of the weights belong to the complementary set $\overline{\mathcal{S}_t}$. With this definition, we summarize the weight update rule in a theorem in order not to disrupt the flow of the paper with derivations. Interested readers can go through the proof.

Weight Update Theorem: Given the current policy, reward and the learning rate α, the weights in the output layer of the NN given in Figure 5 are updated online as follows:

$$\theta_{t+1,j,i} \doteq \begin{cases} \theta_{t,j,i} + \alpha r_t (1 - \pi(a_t|\,\mathbf{h}, \theta_t)) \cdot h_j & \theta_{t,j,i} \in \mathcal{S}_t \\ \theta_{t,j,i} - \alpha r_t \pi(a_i|\,\mathbf{h}, \theta_t)) \cdot h_j & \theta_{t,j,i} \in \overline{\mathcal{S}_t} \end{cases} \quad (6)$$

Proof: The partial derivative of the policy $\pi(a_t|\,\mathbf{h}, \theta)$ with respect to the weights $\theta_{j,i}$ can be expressed using the chain rule as:

$$\frac{\partial \pi(a_t|\,\mathbf{h}, \theta)}{\partial \theta_{j,i}} = \frac{\partial \pi(a_t|\,\mathbf{h}, \theta)}{\partial O_{a_i}(\mathbf{h}, \theta)} \frac{\partial O_{a_i}(\mathbf{h}, \theta)}{\partial \theta_{j,i}} \quad (7)$$

where $1 \leq j \leq N_h + 1$ and $1 \leq i \leq N_A$. When $\theta_{t,j,i} \in \mathcal{S}_t$, action a_t corresponds to output $O_{a_t}(\mathbf{h}, \theta)$. Hence, we can express the first partial derivative using Equation 4 as follows:

$$\begin{aligned} \frac{\partial \pi(a_t|\,\mathbf{h}, \theta)}{\partial O_{a_t}(\mathbf{h}, \theta)} &= \frac{e^{O_{a_t}(\mathbf{h}, \theta)}}{\sum_{j=1}^{N_a} e^{O_{a_j}(\mathbf{h}, \theta)}} - \frac{\left(e^{O_{a_t}(\mathbf{h}, \theta)}\right)^2}{\left(\sum_{j=1}^{N_a} e^{O_{a_j}(\mathbf{h}, \theta)}\right)^2} \\ &= \pi(a_t|\,\mathbf{h}, \theta)\Big(1 - \pi(a_t|\,\mathbf{h}, \theta)\Big) \end{aligned} \quad (8)$$

Otherwise, i.e., $\theta_{t,j,i} \in \overline{\mathcal{S}_t}$, the derivative is taken with respect to another output. Hence, we can find the partial derivative as:

$$\frac{\partial \pi(a_t|\mathbf{h}, \theta)}{\partial O_{a_i}(\mathbf{h}, \theta)} = -\frac{e^{O_{a_t}(\mathbf{h}, \theta)} e^{O_{a_i}(\mathbf{h}, \theta)}}{\left(\sum_{j=1}^{N_A} e^{O_{a_j}(\mathbf{h}, \theta)}\right)^2} = -\pi(a_t|\mathbf{h}, \theta)\pi(a_i|\mathbf{h}, \theta) \quad (9)$$

The second partial derivative in Equation 7, $\partial O_{a_i}(\mathbf{h}, \theta)/\partial \theta_{j,i}$, can be easily computed as h_j using Equation 3. The weight update is the product of learning rate α, reward r_t, h_j and the partial derivative of the policy with respect to the output functions. For the weights $\theta_{t,j,i} \in \mathcal{S}_t$, we use the partial derivative in Equation 8. For the remaining weights, we use Equation 9. Hence, we obtain the first and second lines in Equation 6, respectively. **Q.E.D** □

In summary, the weights of the output layer are updated online using Equation 6 after a user feedback. Detailed results for the improvement in accuracy using RL are presented in Section 5.3.

5 EXPERIMENTAL EVALUATION

5.1 Experimental Setup

Wearable System Setup: The proposed HAR framework is implemented on the TI-CC2650 [34] IoT device, which includes a motion processing unit. It also integrates a radio that runs Bluetooth Low Energy (BLE) protocol. This device is placed on the ankle, since this allows for a maximum swing in the accelerometer [16][1]. The users wear the flexible stretch sensor on the right knee to capture the knee movements of the user. In our current implementation, the stretch sensor transmits its output to the IoT device over BLE to provide flexibility in placement. To synchronize the sensors, we record the wall clock time of each sensor at the beginning of the experiment. Then, we compute the offset between the sensors, and use this offset to align the sensor readings, as proposed in [32]. After completing the processing on the IoT device, the recognized activities and their time durations are transmitted to a host, such as a smartphone, for debugging and offline analysis.

Parameter Selection: We use the default sampling frequencies: 100 Hz for the stretch sensor and 250 Hz for the accelerometer. Lower sampling frequencies did not produce any significant power savings. The raw sensor readings are preprocessed using a moving average filter with a window of nine samples.

User Studies: We evaluate the accuracy of the proposed approach using data from nine users, as summarized in Table 2. The users consist of eight males and one female, with ages 20–40 years and heights 160–180 cm. Data from only five of them are employed during the training phase. This data is divided into 80% training/validation and 20% test following the common practice. The rest of the user data is saved for evaluating only the online reinforcement learning framework. Each user performs the activities listed in Table 1 while wearing the sensors. For example, the illustration in Figure 4 is from an experiment where the user jumps, takes 10 steps, sits on a chair, and finally stands up. The experiments vary from 21 seconds to 6 minutes in length, and have different composition of activities. We report results from 58 different experiments with a 100 minutes total duration, as summarized in Table 2. After each experiment, the segmentation algorithm presented in Section 3.2 is used to identify non-uniform activity windows. This results in 2614 unique different segments in our experimental data. Then, each window is labeled manually through visual inspection by four human experts. Finally, the labeled data is used for offline training. Comparing specific HAR approaches is challenging, since data is collected using different platforms, sensors and settings. Therefore, we compare our results with all commonly used classifiers in the next section. We also release the labeled experimental data to the public on the eLab web page[2] to enable other researchers to make comparisons using a common data set.

Table 2: Summary of user studies

Users	Unique Experiments	No. of Segments	Duration (min)
9	58	2614	100

5.2 Training by Supervised Learning

We use a neural network to perform online activity recognition and training. The NN has to be implemented on the wearable device with a limited memory (in our case 20kB). Therefore, it should have a small memory footprint, i.e., number of weights, while giving a high recognition accuracy. To achieve robust online weight updates during reinforcement learning, we first fix the number of hidden layers to one. Then, we vary the number of neurons in the hidden

[1] We plan to integrate the stretch sensor and the TI-CC2650 into single flexible hybrid electronics device [15], as shown in Figure 1, in our future work.
[2] http://elab.engineering.asu.edu/public-release/

layer to study the effect on the accuracy and memory requirements. Specifically, we vary the number of hidden layer neurons from one to seven. Note that the number of neurons in the output layer remains constant as we do not change the number of activities being recognized. Figure 6 shows the recognition accuracy (left axis) and memory requirements (right axis) of the network as a function of number of neurons in the hidden layer. We observe that the accuracy is only about 80%, when a single neuron is used in the hidden layer. As we increase the number of neurons, both the memory requirements and accuracy increase. The accuracy starts saturating after the third neuron, while the number of weights and memory requirements increase. In fact, the increase in memory requirement is linear, with an increase of around 500 bytes with every additional neuron in the hidden layer. Thus, there is a trade-off between the memory requirements and accuracy. In our HAR framework, we choose an NN with four neurons in the hidden layer as it gives an overall accuracy of about 97.7% and has a memory requirement of 2 kB, leaving the rest of the memory for operating system and other tasks.

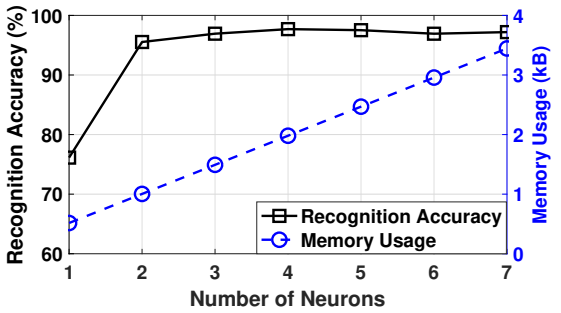

Figure 6: Comparison of accuracy with number of neurons

5.2.1 Confusion Matrix

We analyze the accuracy of recognizing each activity in our experiment in Table 3. There is one column and one row corresponding to the activities of interest. The numbers on the diagonal show the recognition accuracy for each activity. For example, the first row in the first column shows that driving is recognized with 99.4% accuracy. According to the first row, only 0.6% of the driving activity windows are classified falsely as "Transition". To provide also the absolute numbers, the number in parenthesis at the beginning of each row shows the total number of activity windows with the corresponding label. For instance, a total of 155 windows were labeled "Drive" according to row 1.

We achieve an accuracy greater than 97% for five of the seven activities. The accuracy is slightly lower for jump because it is more dynamic than all the other activities. Moreover, there is a higher variability in the jump patterns for each user, leading to slightly lower accuracy. It is also harder to recognize transitions due to the fact that each transition segment contains features of two activities. This can lead to a higher confusion for the NN, but we still achieve more than 90% accuracy. We also note that the loss in accuracy is acceptable for transitions, since we can indirectly infer a transition by looking at the segments before and after the transition.

Table 3: Confusion matrix for 5 training users

	Drive	Jump	Lie Down	Sit	Stand	Walk	Transition
D (155)	**99.4%**	0.00	0.00	0.00	0.00	0.00	0.6%
J (181)	0.00	**93.4%**	0.00	0.00	1.1%	3.9%	1.6%
L (204)	0.00	0.00	**100%**	0.00	0.00	0.00	0.00
S (394)	0.25%	0.25%	0.00	**97.7%**	0.76%	0.00	1.0%
Sd (350)	0.00	0.29%	0.00	0.00	**98.6%**	1.1%	0.00
W (806)	0.00	0.50%	0.00	0.00	0.62%	**98.5%**	0.37%
T (127)	0.00	3.1%	0.79%	2.4%	0.79%	2.4%	**90.5%**

5.2.2 Comparison with other classifiers

It is not possible to do a one to one comparison with existing approaches because they use different devices, data sets and activities. Therefore, we use our data set with the commonly used classifiers described in Section 3.4. The results are summarized in Table 4. Although we use only a single hidden layer and minimize the number of neurons, our implementation achieves competitive test and overall accuracy compared to the other classifiers. We also emphasize that our NN is used for both online classification and training on the IoT device.

Table 4: Comparison of accuracy for different classifiers

Classifier	Train Acc. (%)	Test Acc. (%)	Overall Acc. (%)
Random Forest	100.00	94.58	98.92
C4.5	99.09	93.90	98.05
k-NN	100.00	94.80	98.96
SVM	97.68	95.03	97.15
Our NN	98.53	94.36	97.70

5.3 Reinforcement Learning with new users

The NN classifier is used to recognize the activities of four new users that are previously unseen by the network. This capability provides a real world evaluation of the approach, since a device cannot be trained for all possible users. Due to variations in usage patterns, it is possible that the initial accuracy for a new user is low. Indeed, the initial accuracy for users 6 and 9 is only about 60–70%. Therefore, we use reinforcement learning using policy gradients to continuously adapt the HAR system to each user. Figure 7 shows the improvement achieved using reinforcement learning for four users. Each episode in the x-axis corresponds to an iteration of RL using the data set for new users. The weights of the NN are updated after each segment as a function of the user feedback for a total of 100 episodes. Moreover, we run 5 independent runs, each consisting of 100 epochs, to obtain an average accuracy of the NN at each episode. We observe consistent improvement in accuracy for all the four users. The accuracy for users 6 and 9 starts low and increases to about 93% after about 20 episodes. User 8 starts with a higher accuracy of about 85%. The accuracy increases quickly to about 98% after 10 episodes. In summary, reinforcement learning improves the accuracy for users not previously seen by the network. This ensures that the device can adapt to new users very easily.

5.4 Power, Performance and Energy Evaluation

To fully assess the cost of the proposed HAR framework, we present a detailed breakdown of execution time, power consumption and energy consumption for each step. The first part of HAR involves data acquisition from the sensors and segmentation. Since the segmentation algorithm run continuously while the data is acquired, its energy consumption is included in the sensing block. Table 5

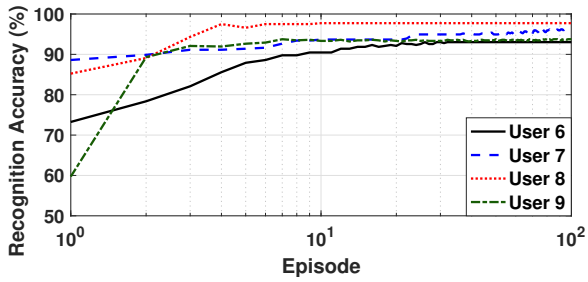

Figure 7: Reinforcement learning results for four new users

shows the power and energy consumption for a typical segment of 1.5 s. The average power consumption for the data acquisition is 1.13 mW, leading to a total energy consumption of 1695 μJ. If the segments are of a longer duration, the energy consumption for data sensing increases linearly. Following the data segmentation, we extract the features and run the classifier. The execution time, power and energy for these blocks are shown in the "Compute" rows in Table 5. As expected, the FFT block has the largest execution time and energy consumption. However, it is still two orders of magnitude lower than the duration of a typical segment. Finally, the energy consumption of the BLE communication block is given in the last row of Table 5. Since we transmit the inferred activity, the energy consumed by the BLE communication is only about 43 μJ. With less than 12.5 mW average power consumption, our approach enables close to 60-hour uninterrupted operation using a 200 mAh @ 3.7 V battery [12]. Hence, it can enable self-powered wearable devices [6] that can harvest their own energy [25].

Table 5: Execution time, power and energy consumption

	Block	Exe. Time (ms)	Average Power (mW)	Energy (μJ)
Sense	Read/Segment	1500.00	1.13	1695.00
Compute	DWT	7.90	9.50	75.05
	FFT	17.20	11.80	202.96
	NN	2.50	12.90	32.25
	Overall	27.60	11.24	310.26
Comm.	BLE	8.60	5.00	43.00

6 CONCLUSIONS

We presented a HAR framework on a wearable IoT device using stretch and accelerometer sensors. The first step of our solution is a novel technique to segment the sensor data non-uniformly as a function of the user motion. Then, we generate FFT and DWT features using the segmented data. Finally, these features are used for online inference and training using an NN. Our solution is the first to perform online training. Experiments on TI-CC2650 MCU with nine users show 97.7% accuracy in identifying six activities and their transitions with less than 12.5 mW power consumption.

Acknowledgment: This work was supported by NSF CAREER award CNS-1651624.

REFERENCES

[1] Parkinsons Disease Digital Biomarker DREAM Challenge. [Online] https://www.synapse.org/#!Synapse:syn8717496/wiki/. Accessed 04/15/2018.
[2] World Health Organization, Obesity and Overweight. Fact Sheets, 2013. [Online] http://www.who.int/mediacentre/factsheets/fs311/en/. Accessed 03/22/2018.
[3] D. Anguita et al. Energy Efficient Smartphone-Based Activity Recognition Using Fixed-Point Arithmetic. *J. of Universal Comput. Sci.*, 19(9):1295–1314.
[4] M. Arif, M. Bilal, A. Kattan, and S. I. Ahamed. Better Physical Activity Classification Using Smartphone Acceleration Sensor. *J. of Med. Syst.*, 38(9):95, 2014.
[5] L. Bao and S. S. Intille. Activity Recognition From User-Annotated Acceleration Data. In *Int. Conf. on Pervasive Comput.*, pages 1–17, 2004.
[6] G. Bhat, J. Park, and U. Y. Ogras. Near-Optimal Energy Allocation for Self-Powered Wearable Systems. In *Proc. Int. Conf. on Comput.-Aided Design*, pages 368–375, 2017.
[7] A. G. Bonomi, A. H. Goris, B. Yin, and K. R. Westerterp. Detection of Type, Duration, and Intensity of Physical Activity Using an Accelerometer. *Medicine & Science in Sports & Exercise*, 41(9):1770–1777, 2009.
[8] J. Bort-Roig et al. Measuring and Influencing Physical Activity With Smartphone Technology: A Systematic Review. *Sports Medicine*, 44(5):671–686, 2014.
[9] M. A. Case, H. A. Burwick, K. G. Volpp, and M. S. Patel. Accuracy of Smartphone Applications and Wearable Devices for Tracking Physical Activity Data. *Jama*, 313(6):625–626, 2015.
[10] Y. Chen and C. Shen. Performance Analysis of Smartphone-Sensor Behavior for Human Activity Recognition. *IEEE Access*, 5:3095–3110, 2017.
[11] K. Dinesh et al. Signal Analysis for Detecting Motor Symptoms in Parkinson's and Huntington's Disease Using Multiple Body-Affixed Sensors: A Pilot Study. In *Image and Signal Process. Workshop*, pages 1–5, 2016.
[12] DMI International Distribution Ltd. Curved lithium thin cells. [Online] http://www.dmi-international.com/data%20sheets/Curved%20Li%20Polymer.pdf Accessed 04/18/2018.
[13] A. J. Espay et al. Technology in Parkinson's Disease: Challenges and Opportunities. *Movement Disorders*, 31(9):1272–1282, 2016.
[14] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer, 2001.
[15] U. Gupta, J. Park, H. Joshi, and U. Y. Ogras. Flexibility-Aware System-on-Polymer (SoP): Concept to Prototype. *IEEE Trans. Multi-Scale Comput. Syst.*, 3(1):36–49, 2017.
[16] N. Győrbíró, Á. Fábián, and G. Hományi. An Activity Recognition System for Mobile Phones. *Mobile Networks and Appl.*, 14(1):82–91, 2009.
[17] Y. He and Y. Li. Physical Activity Recognition Utilizing the Built-in Kinematic Sensors of a Smartphone. *Int. J. of Distrib. Sensor Networks*, 9(4):481–580, 2013.
[18] R. Jafari, W. Li, R. Bajcsy, S. Glaser, and S. Sastry. Physical Activity Monitoring for Assisted Living at Home. In *Int. Workshop on Wearable and Implantable Body Sensor Network*, pages 213–219, 2007.
[19] M. Kirwan, M. J. Duncan, C. Vandelanotte, and W. K. Mummery. Using Smartphone Technology to Monitor Physical Activity in the 10,000 Steps Program: A Matched Case–Control Trial. *J. of Med. Internet Research*, 14(2), 2012.
[20] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity Recognition Using Cell Phone Accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.
[21] M. G. Lagoudakis and R. Parr. Reinforcement Learning as Classification: Leveraging Modern Classifiers. In *Proc. Int. Conf. Mach. Learn.*, pages 424–431, 2003.
[22] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Trans. Neural Netw.*, 17(6):1411–1423, 2006.
[23] A. Mosenia, S. Sur-Kolay, A. Raghunathan, and N. K. Jha. Wearable Medical Sensor-Based System Design: A Survey. *IEEE Trans. Multi-Scale Comput. Syst.*, 3(2):124–138, 2017.
[24] B. O'Brien, T. Gisby, and I. A. Anderson. Stretch Sensors for Human Body Motion. In *Electroactive Polymer Actuators and Devices*, volume 9056, page 905618, 2014.
[25] J. Park et al. Flexible PV-cell Modeling for Energy Harvesting in Wearable IoT Applications. *ACM Trans. Embed. Comput. Syst.*, 16(5s):156, 2017.
[26] A. Paul and D. P. Mukherjee. Reinforced Random Forest. In *Proc. Indian Conf. on Comput. Vision, Graphics and Image Processing*, pages 1:1–1:8, 2016.
[27] Pérez-López et al. Dopaminergic-Induced Dyskinesia Assessment Based on a Single Belt-Worn Accelerometer. *Artificial Intell. in Medicine*, 67:47–56, 2016.
[28] S. Pirttikangas, K. Fujinami, and T. Nakajima. Feature Selection and Activity Recognition From Wearable Sensors. In *Int. Symp. on Ubiquitious Comput. Systems*, pages 516–527, 2006.
[29] S. J. Preece et al. Activity Identification Using Body-Mounted Sensors–A Review of Classification Techniques. *Physiological Measurement*, 30(4):R1, 2009.
[30] J. R. Quinlan. *C4. 5: Programs for Machine Learning*. Elsevier, 2014.
[31] M. Shoaib et al. A Survey of Online Activity Recognition Using Mobile Phones. *Sensors*, 15(1):2059–2085, 2015.
[32] S. Sridhar, P. Misra, G. S. Gill, and J. Warrior. Cheepsync: A Time Synchronization Service for Resource Constrained Bluetooth LE Advertisers. *IEEE Commun. Mag.*, 54(1):136–143, 2016.
[33] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 2nd edition, 2018.
[34] Texas Instruments Inc. CC-2650 Microcontroller. [Online] http://www.ti.com/product/CC2650 Accessed 04/18/2018.
[35] A. Wang, G. Chen, J. Yang, S. Zhao, and C.-Y. Chang. A Comparative Study on Human Activity Recognition Using Inertial Sensors in a Smartphone. *IEEE Sensors J.*, 16(11):4566–4578, 2016.