MODELLING INTERLEAVED ACTIVITIES USING LANGUAGE MODELS

Eoin Rogers, Robert J. Ross, John D. Kelleher
Applied Intelligence Research Centre
Technological University Dublin
Dublin, Ireland
eoin.rogers@tudublin.ie, robert.ross@tudublin.ie, john.d.kelleher@tudublin.ie

KEYWORDS

Activity discovery ; Activity Recognition ; Interleaving ; Neural language modelling ; Behaviour modelling

ABSTRACT

We propose a new approach to activity discovery, based on the neural language modelling of streaming sensor events. Our approach proceeds in multiple stages: we build binary links between activities using probability distributions generated by a neural language model trained on the dataset, and combine the binary links to produce complex activities. We then use the activities as sensor events, allowing us to build complex hierarchies of activities. We put an emphasis on dealing with interleaving, which represents a major challenge for many existing activity discovery systems. The system is tested on a realistic dataset, demonstrating it as a promising solution to the activity discovery problem.

INTRODUCTION

Given the increasing ubiquity of computer systems in our everyday lives, using them to model, monitor and analyse human behaviour becomes increasingly possible and useful. The field of *activity recognition* studies the design and implementation of such systems (Kwapisz et al., 2011; Kim et al., 2009), which can be useful for applications as diverse as elder care and security.

One persistent issue facing activity recognition is the difficulty in finding suitably annotated datasets. Labelling such datasets can be time consuming, and in many cases is quite difficult due to differences in levels of abstraction and ambiguities about the precise start and end times of activities. In order to address this, the field of *activity discovery* (AD) has proposed the unsupervised extraction of plausible human activities from unannotated datasets (Gjoreski and Roggen, 2017; Cook et al., 2013; Rogers et al., 2016).

Real-word activities are often *interleaved*, meaning that they take place at the same time. This results in sensor readings for multiple activities showing up in quick succession on the data stream. Recognising that this is happening, separating the activities from each other, and recognising that sensor events that do not occur adjacent to each other may be part of the same activity is a significant challenge for existing activity discovery systems.

In this paper, we propose a novel approach to activity discovery that makes use of *neural language models*, developed by the natural language processing (NLP) community, to model the sensor feeds from activity discovery systems and extract useful activities from them. This work builds upon a previous related system presented in (Rogers et al., Forthcoming), but with major changes including the discovery of proper, non-binary activities, and clustering of activities into distinct types. Our approach is also designed to be aware of, and disentangle, interleaved activities. This paper is divided into five remaining sections, outlining the activity discovery problem in more detail, covering prior work in the field, a description of our approach, a description of the experiments we ran to test our approach, and a presentation of our results respectively.

ACTIVITY DISCOVERY

In order to have a clean model description, it is necessary for us to briefly introduce the terminology that we will use later. Formally, an activity discovery system can be modelled as a 5-tuple (Σ, D, A, f, g), where:

- Σ is a set of *event types*;
- D is an ordered sequence of events, $D = \langle d_1, d_2, \ldots, d_L \rangle$ of length L, such that each $d_i \in D$ is drawn from the set Σ. We call this the *dataset*;
- A is a set of *activity types*;
- f is a mapping $f : D \to X^*$, which takes a sequence of events D as input, and returns a set of (possibly non-contiguous) sub-sequences of D as output; and
- g is a mapping $g : X \to A$, where $X \subset D^*$, which takes a sub-sequence produced by f as input, and returns an activity type $a \in A$ as output.

This definition can be made clearer with a concrete example. Supposing we have a dataset $D = \langle d_1, d_2, \ldots, d_N \rangle$. Each $d_i \in \Sigma$ is a sensor event drawn from Σ, our full set of sensor events. In an environment where sensors have been set up in a home, for instance, Σ could consist of events such as *open front door*, *turn oven on*, *flush toilet* and similar domestic events. An *activity*, then, is simply a sub-sequence of D consisting of events that appear to the activity discovery system

Communications of the ECMS, Volume 34, Issue 1,
Proceedings, ©ECMS Mike Steglich, Christian Mueller,
Gaby Neumann, Mathias Walther (Editors)
ISBN: 978-3-937436-68-5/978-3-937436-69-2(CD) ISSN 2522-2414

to be semantically related. For instance, we would expect that events such as *turn oven on*, *open kitchen cupboard*, *open refrigerator* might occur in an activity together, since they tend to occur together temporally. It should be noted that D is not a set of sequences as might be the case in a supervised learning setting; D is a single large dataset from which we extract activities.

Multiple similar activities can then be clustered or lifted into one *type*. The activity discovery system might notice that an activity similar to the one mentioned in the previous paragraph seems to occur nightly, and may cluster them all into a single *making dinner* activity type. The concrete sub-sequences of D are referred to as the *instances* of the *making dinner* activity.

Note that we don't generally expect an activity discovery system to operate with human-like semantic knowledge or expectations in the basic case. Thus, it would not be expected to be able to name the new activity type as *making dinner*, only to identify that the instances involved can be sensibly clustered together. A commercial activity discovery system might well be supplemented with real-world knowledge, with the intention of biasing towards the sort of activities we would expect to find in the environment in which it operates. For instance, knowledge that events relating to a fridge or oven indicates activities relating to food preparation such as *making dinner* are taking place. In many ways this would stray over into being a form of activity recognition as well as discovery. For this reason, we stick to a pure form of activity discovery without any real-world knowledge. We do still expect to be able to discover *making dinner* as an activity, just not to be able to give it a label (*making dinner*) that would be semantically meaningful to a human observer.

PRIOR WORK

A number of existing approaches to activity discovery exist in the literature. Cook et al. (2013) provides a good overview of the field. This paper also introduces an activity discovery system that applies a beam search algorithm using an operator called *ExtendSequence* to discover activities in an unlabelled dataset. Like a number of other systems in the field, this algorithm utilises the *minimum description length* (MDL) principle (Rissanen, 1978, 1989), which proposes evaluating machine learning models by measuring the degree to which they *compress* their input dataset. This is an important principle, and one which will turn out to be useful to our own work also.

Activity discovery can also be carried out by relatively simple systems that utilise topic models (Huynh et al., 2008). Here, the *latent Dirichlet allocation* (LDA) topic model (Blei et al., 2003) is used to model the relationship between sensor events and latent variables which are presumed to represent activities. The model is shown to have good performance, even on a complex dataset. More recently, other models based on statistical models have been proposed: for example, Fang et al. (2019) proposes activity discovery by means of a hierarchical mixture model. Saives et al. (2015) propose using activity discovery to build a model of normal behaviour patterns of a person in order to detect anomalous behaviours that may be of interest to medical professionals.

Related fields also provide an important source of ideas. Grammar induction is a concept from computational linguistics which refers to the derivation of grammar productions for a language given only a dataset. Some forms of grammar induction require labelled input, distinguishing positive and negative examples, but others require only positive examples. In the general case, grammar induction is not a tractable problem, regardless of whether the dataset is labelled or not (Gold, 1967), but tractable approximations have been demonstrated which solve the problem to a degree (Cramer, 2007). This problem is by no means equivalent to activity discovery (and is in fact in many ways harder), but it does involve the induction of structure from a one-dimensional input vector. *Adios* (Solan et al., 2005) induces a grammar by loading a dataset into memory as a graph, with words represented as vertexes and sentences represented as directed edges between these. This representation allows for the identification of *equivalence classes* between words and phrases which share the same input and output edges, which can then be added to the graph as nonterminals. A variant of the Adios approach, which supplements the basic grammar induction algorithm with logical predicates to allow for more accurate induction in a limited linguistic domain is presented by (Gaspers et al., 2011).

The *eGrids* grammar induction algorithm (Petasis et al., 2004) bears a resemblance to the beam search-based system mentioned previously from (Cook et al., 2013). It also uses an MDL-based objective function to guide the search. An interesting deep learning-based grammar induction model using convolutional networks to determine *syntactic distance* (the degree to which two neighbouring words or symbols belong to the same POS phrase) is similar to the approach we present in this paper (Shen et al., 2017). Finally, our approach can also be understood as a non-local variant of the tree structure induction algorithm *Sequitur* (Nevill-Manning and Witten, 1997), which groups input symbols together even if they do not appear contiguously.

Alshammari et al. (2017) present a 3D simulation of a house that can be used to automatically generate datasets for use in activity discovery, activity recognition and related fields. This could prove useful for validating activity discovery systems.

APPROACH

We will now outline the approach that we have taken to activity discovery. From the perspective of somebody using our system, we assume that the input is a dataset consisting of a finite series of discrete sensor events $D = \langle d_1, d_2, \ldots, d_L \rangle$. Each individual sensor event has an associated *type* from a fixed set of types T. We write the type of d_i as $t(d_i) \in T$. The primary output from the system is a series of discovered *activities* $\langle Act_1, Act_2, \ldots, Act_N \rangle$, where each activity is a tuple of the form $(Indexes_{Act_j}, Type_{Act_j})$. $Indexes_{Act_j}$ is a set of indexes into the dataset D, $\{x_1, x_2, \ldots, x_{ActSize(j)}\}$, of length $ActSize(j)$, which can be different for each activity output by the system. $Type_{Act_j}$ is a *type* associated with the discovered activity, analogous to the type of a sensor event discussed above. Act_j and Act_k may share the same type, but they may not share the same set of indexes.

The basic internal operation of our model proceeds according to the following three steps:

- We build a probabilistic model by analysing the dataset. Given a subset of the dataset $D_{i:i+n-1}$, which we call the *sliding window*, we use the model to predict the probability distribution over sensor events $P(d_{i+n+l}|D_{i:i+n-1})$ for all $l \in \{0, 1, \ldots, m-1\}$. We call the subset of the dataset $D_{i+n:i+n+m-1}$ the *lookahead window*, m the *lookahead length*, and l the *lookahead offset*.
- We use the probabilistic model to construct *links* between sensor events if the model is confident that one event can be predicted from the other. Links are grouped together to create the $Indexes_{Act_j}$ part of the output described above.
- Activities are then *clustered* together based on the *similarity of the sensor types present within them.* The clusters are used as the $Type_{Act_j}$ part of the output described above.

A more detailed outline of these stages now follows.

NLP practitioners will of course recognise that the probabilistic model that we describe is a form of *language model*. As is now common in that field, we use a *neural language model* (Bengio et al., 2003) or NLM. We use a modern recurrent design – specifically the LSTM-based (Hochreiter and Schmidhuber, 1997) approach described in (Zaremba et al., 2014) (which is itself adapted from (Graves, 2013)), which allows for the modelling of long-distance dependencies and robustness to noise. The LSTM consists of an input gate i, a forget gate f, and an output gate o. If h_t^l denotes the output of layer l at timestep t, g denotes the *modulated input* and c_t^l denotes the value of an LSTM cell in layer l at time t, activation of the gates is defined as:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} + b \tag{1}$$

We then update the value in the cell:

$$c_t^l = f \odot c_{t-1}^t + i \odot g \tag{2}$$

Where $\odot$ denotes the *Hadamard product* (i.e. elementwise multiplication, as opposed to matrix multiplication). The output h_t^l is then:

$$h_t^l = o \odot tanh(c_t^l) \tag{3}$$

We train m networks, one for each lookahead offset. As a result, one network is responsible for predicting the first item in the lookahead window, another for predicting the second and so on. This process is illustrated in Fig. 1.

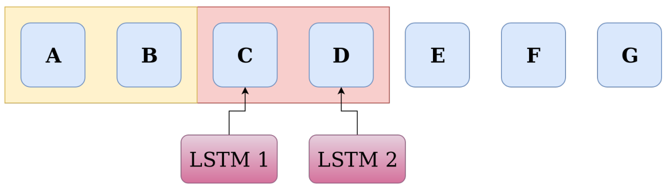

Fig. 1: In this example, events A and B are inside the sliding window, and are fed as input into two LSTMs. Each LSTM has to predict a probability distribution over the corresponding offset within the lookahead window containing C and D.

Building links between activities is carried out in two stages: building simple binary links, and grouping the links together to form activities. The binary links are built using the NLM. First we run the networks over the entire dataset. We build a binary link between the final event in the sliding window and the lth event in the lookahead window if the lth LSTM successfully predicts the lth event from the sliding window (for example, between events B and D in Fig. 1 if LSTM 2 predicted event D). Doing this naively would make the system vulnerable to building links between common events. In the NLP community, this approach is usually solved by removing common words (stop words), but this could reduce the quality of the resulting activities discovered.

As a result, we don't work with probabilities directly, but rather with *probability deltas*. Again looking at Fig. 1, this is the average probability that LSTM 2 predicts D when event B is present in the sliding window *minus* the probability when it isn't present. Thus, we only build a link between B and D if the presence of event B makes the LSTM more confident that event D is to follow. The exact calculation we use is presented in equation 4, where P_{i+n}^l denotes the probability vector produced by the lth LSTM of the $i+n$th item in the dataset (which we denote above as $P(d_{i+n+l}|D_{i:i+n})$).

$$delta(P_{i+n}^l) = \frac{\sum_{k=i+n}^{i+n+n-1} P_k^l}{k} - \frac{P_i^l + P_{i+n+n}^l}{2} \tag{4}$$

A link is built between d_{i+n} and d_{i+n+l} *if and only if this probability delta exceeds a certain threshold.* The thresholds are computed at runtime, since different event types need different associated thresholds for the link-building to work correctly. We experimented with a number of automatic discretisation algorithms. One commonly used method to compute a binary threshold is Otsu's method, which is commonly used in the image processing community. This works by converting an image to greyscale, and then producing a histogram over its pixel intensities. The threshold is the point in the histrogram where the integral of pixel densities to both the left and right of the threshold are equal. This is mathematically equivalent to computing the k-means clustering with $k = 2$ for the pixel

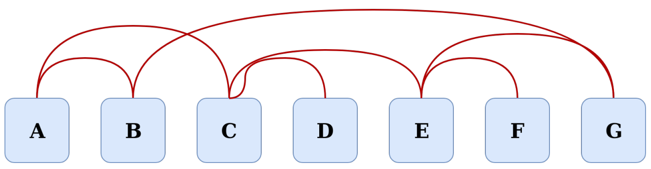

Fig. 2: Links are built between events if an LSTMs probability delta passes a threshold. This threshold is computed dynamically at runtime.

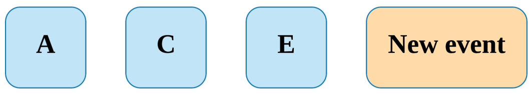

Fig. 3: If events B, D, F and G are all part of the same activity, we can remove these events from the dataset and replace them with a new activity. We can then train and run the system from scratch again, allowing us to build rich hierarchies of activities.

intensities and then taking the average of the two centroids as a threshold. However, this method assumes that the histogram in question has two distinct humps, one for light pixels and another for dark pixels. This is unlikely to be the case for our dataset. In this case, thresholds can be computed automatically by applying Otsu's method, dividing the image into bright and dark sub-images based on the threshold, running Otsu's method over both sub-images, and then using the average of these two thresholds as the threshold for the next iteration. We stop iterating when the threshold begins to converge, i.e. when the threshold computed in two iterations falls below a certain epsilon. This is the method we used to automatically compute thresholds per event type, using probability deltas in place of pixel intensity values. The linking process results in a tangle of binary links between events, as illustrated in Fig. 2.

The links are then grouped together to form activities. For example, if a link is built between the sensor events at indexes 1 and 5, and another link is built between 5 and 8, the system will output an activity consisting of 1, 5 and 8.

The final stage of the three introduced above is clustering. We say two activities have the same type as each other if they share at least 50% of the same event types. We have looked at other, more sophisticated types of clustering, but we have found that the type used has surprisingly little effect on the performance of our system.

Building Hierarchies of Activities

So far, we have maintained a sharp distinction between the types of sensor events and the types of activities. Removing this distinction has an obvious advantage: we can replace the discovered activities in the dataset with their activity types, producing a new dataset that the above process can be repeated on. This allows us to produces *activity hierarchies*, where activities can contain previously discovered activities as members, which allows for the modelling of activities that contain other activities. These are abundant in the real world: for example, a *making dinner* activity could contain a smaller activity such as *chopping vegetables*.

Fig. 3 presents a possible output from such a process. If events B, D, F and G are all found to belong to the same activity, they can be removed and replaced with a new event representing the discovered activity. We can then train and run the system again, which allows this activity to be detected as belonging to other activities. This allows for the building of complex hierarchies of activities, such as the one described in the previous paragraph.

Note that the discovered activity includes events that are not adjacent to each other in the original dataset. For example, event B does not directly neighbour the other events in the activity. This is one of the major strengths of our approach: it does not assume or require that the activities discovered are contiguous. This allows us to deal with one of the must pressing issues in the field of activity discovery, which is that of *interleaving*, where the person or people under observation are carrying out multiple activities in parallel. From the viewpoint of an activity discovery system, interleaved activities usually look like a person switching back and forth between activities, in much the same way that a modern operating system context switches between running processes to allow multitasking. Our approach aims to explicitly address interleaving by disentangling interleaved activities from each other.

One non-obvious aspect of this process is why the new event was placed after activity E. For example, event B was also part of the discovered activity the event is replacing, so would it not make equal sense to place the new event between A and C? We decide where to place the new event based on the number of events it removes from various locations of the original dataset. The event in Fig. 3 removed one event between A and C (event B), one between C and E (event D), and two after E (F and G). Thus, we place the new event after event E.

The process can be repeated as many times as needed to produce a many-layered hierarchy of activities.

EXPERIMENT SETUP

We implemented our system in Python using the TensorFlow (Abadi et al., 2015) machine learning library. We used a four-layer neural language model, with 150 cells per layer for the lowest level of the hierarchy. As we ascended the hierarchy we found that the size increase of the vocabulary due to the addition of discovered activities was straining the network. As a result, we increased the size of the network by 50% for each level: level 1 had 150 LSTM cells, level 2 had 225, level 3 had 337 and so on.

We use the Kyoto 3 dataset from the CASAS smarthome project (Cook and Schmitter-Edgecombe, 2009). This dataset consists of readings from a range of sensors installed in a small apartment. The dataset was gathered by asking a number of participants to perform *activities of daily living* (ADLs) in a natural manner in the apartment. Most of the sensor readings are either binary (they have a simple on/off state), or can only enter one of a handful of states. This means they can be easily converted to the sequence of events format our system expects by creating event types of the form $SensorName\_SensorState$. For example, one of the sensors are referred to as $M17$ in the dataset, and can take the state ON, so $M17\_ON$ becomes an event type in the dataset. For the few sensor types that did have continuous values, we used the *Jenks natural breaks algorithm* (Jenks, 1967) to discretise the data. Our system does not take temporal distance into account, so it cannot, for instance, see large gaps between events. This makes the system's task substantially harder, but it allows us to put our system through much more challenging testing than most activity discovery practitioners settle for.

RESULTS

Evaluating activity discovery systems can be a challenge for a number of reasons. Human annotators may not come to an agreement with each other over the start and end points of activities, which makes working from a gold-standard ground truth quite difficult. For example, when does the activity of $Making\_Dinner$ start? When a person enters the kitchen? When they turn on the oven? In many cases, a ground truth may not even be available (although that isn't an issue for the Kyoto dataset). The output from an AD system may be on a different level of abstraction from the ground truth: for example the system may discover an activity that could be called something like *chop_vegetables*, but the ground truth instead has an activity called *make_dinner*, which *chop_vegetables* would be a constituent of. A good overview of evaluation for activity discovery can be found in (Cook and Krishnan, 2015).

Since we do have access to a ground truth in this experiment, it makes sense to use it, although we must keep the above issues in mind. Because of the abstraction issue mentioned before, we argue that both raw accuracy and F-measures are inappropriate for evaluating this system. Instead, we compare each new event type from each level of the hierarchy using the *precision* metric, i.e. the true positives divided by the sum of the true and false positives. Each event type is then matched with the ground truth activity with which it achieves the highest associated precision.

Given a window length n and lookahead length m of 10, and building a hierarchy of 4 levels, the average precision per level is shown in Table I.

Level number	Average precision
Level 1	0.7694124354455739
Level 2	0.8183002393181837
Level 3	0.82831171138164
Level 4	0.8341719705663135

TABLE I: Average precision score achieved for a 4-level hierarchy

We can see that the results improve the higher up through the hierarchy we ascend. This is expected, since the events become more abstract and thus closer to the (very abstract) activities in the ground truth. We also compute the results per discovered activity (the results presented in Table I is the average over these scores.) These are too large to be presented in full in this paper, but an extract of the full results are presented as Table II.

Event type	Precision
new_event_45	1.0
new_event_46	1.0
new_event_47	0.75
new_event_48	1.0
new_event_49	0.5
new_event_51	0.5

TABLE II: Extract of the full results, showing the precision of each activity type found

We also experimented with different hyperparameter values. In particular, we studied the effect of adjusting both the window and lookahead lengths. Increasing the window length has a moderate negative impact on the observed results, as shown in Table III. This indicates that the extra information provided in the longer sliding window actually ends up confusing the LSTM networks, since they observe conflicting signals as a result of now having multiple activities visible in their input at any one point in time. Most activities, even when highly interleaved, tend to be very "local", with events that constitute the activity being located quite close together in the dataset.

Perhaps less surprising is the strong negative correlation observed between lookahead length and precision, shown in Table IV. This is also to be expected: our system is very good at linking nearby events, but struggles to confidently link distant events, which is very much to be expected when processing sequential data, but the extent of the negative correlation is worth pointing out.

A particularly interesting and encouraging result is the difference in performance when dealing with interleaved and non-interleaved datasets. Table V shows the average precision for the system when given a *non-interleaved* dataset as input. Compared to the results for the interleaved dataset (Table I) we see the lack of interleaving is actually confusing the system, which

Window length	Average precision
10	0.80
15	0.80
20	0.79
25	0.77

TABLE III: Relationship between window length and precision

Lookahead length	Average precision
10	0.80
15	0.64
20	0.56
25	0.48

TABLE IV: Relationship between lookahead length and precision

Level number	Average precision
Level 1	0.7376020200520275
Level 2	0.7744498540343416
Level 3	0.7964471494200084
Level 4	0.8018082799378542

TABLE V: Average precision score achieved when using a non-interleaved dataset

is the opposite to what is usually observed in activity discovery systems. This demonstrates that we have a strong reason to claim that this system is well suited to dealing with interleaved datasets. It also means that performance could likely be boosted further by an ensemble of this system and traditional activity discovery systems, since these results demonstrate that they arguably have different strengths, and combining models with different strengths is generally a good idea when carrying out ensemble learning.

We have already mentioned minimum description length (MDL) in the prior work section on this paper on page 2. (Cook et al., 2013) suggest using this as the basis for another metric for activity discovery systems, namely that of *compression ratio*. Since our system is constructing a hierarchy of activities by removed the sensor events that are found to belong to the discovered activities, the dataset reduces in size over time. Compression ratio alone can serve as a metric, since having a high compression ratio can be a sign that the system is correctly finding activities present in the dataset. Our system compresses the original Kyoto3 dataset to around 36% if its original size.

(Cook and Krishnan, 2015) proposes that the concept of compression ratio could be converted into a more principled metric by measuring how well compression ratio *generalises*. In traditional machine learning, we may be more concerned with how a system deals with novel input compared to how it deals with input seen in the dataset. This allows us to be sure it is learning a signal present in the dataset, rather than just memorising the contents of the dataset. This is typically measured using techniques like holding out a validation dataset from the main dataset. If a system is generalising well, its performance on the testing dataset should be similar to the performance on the training dataset. Likewise, if an activity discovery system compresses the dataset by a certain amount, it should also compress a testing dataset by the same amount. We tested our system using ten-fold cross-validation. The results, presented in Table VI, show clearly that the system is finding activities that generalise well to the test dataset.

Cross-validation	Compression ratio
Fold 1	0.3927149342694845
Fold 2	0.32898505905289627
Fold 3	0.3363159811817841
Fold 4	0.34688667906136067
Fold 5	0.35323690998006696
Fold 6	0.38618623218659726
Fold 7	0.315419492673265
Fold 8	0.32600179330230683
Fold 9	0.4082227112380515
Fold 10	0.3370837650161199

TABLE VI: Ten-fold cross validation of the compression ratio produced by the system

CONCLUSION

This paper introduced a deep learning-based activity discovery system. We have described the system, and also presented results illustrating its performance on a realistic dataset, and an analysis of how the results change in response to a change in the system's meta-parameters. We feel that these results show that the system performs favourably compared to other systems in the field, and could be adapted for use in real-world activity discovery applications.

A number of changes could be made to this system to improve its performance and further test it. We have already mentioned the work of (Alshammari et al., 2017) as a possible means to generate datasets for very in-depth testing. Testing using the Opportunity dataset (Chavarriaga et al., 2013) could also be useful. Changing the design of the network to take into account the temporal information included in the Kyoto dataset, but not utilised by our model, is another possible way to improve the system in the future.

REFERENCES

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

N. Alshammari, T. Alshammari, M. Sedky, J. Champion, and C. Bauer. Openshs: Open smart home simulator. *Sensors*, 17(5):1003, 2017.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3 (Jan):993–1022, 2003.

R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. d. R. Millán, and D. Roggen. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters*, 34(15):2033–2042, 2013.

D. J. Cook and N. C. Krishnan. *Activity learning: discovering, recognizing, and predicting human behavior from sensor data*. John Wiley & Sons, 2015.

D. J. Cook and M. Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of information in medicine*, 48(05):480–485, 2009.

D. J. Cook, N. C. Krishnan, and P. Rashidi. Activity discovery and activity recognition: A new partnership. *IEEE transactions on cybernetics*, 43(3):820–828, 2013.

B. Cramer. Limitations of current grammar induction algorithms. In *Proceedings of the 45th annual meeting of the ACL: student research workshop*, pages 43–48. Association for Computational Linguistics, 2007.

L. Fang, J. Ye, and S. Dobson. Discovery and recognition of emerging human activities using a hierarchical mixture of directional statistical models. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

J. Gaspers, P. Cimiano, S. S. Griffiths, and B. Wrede. An unsupervised algorithm for the induction of constructions. In *2011 IEEE International Conference on Development and Learning (ICDL)*, volume 2, pages 1–6. IEEE, 2011.

H. Gjoreski and D. Roggen. Unsupervised online activity discovery using temporal behaviour assumption. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 42–49, 2017.

E. Gold. Language identification in the limit. i nformation andcontrol, 10: 447-474, 1967.[11] s. *Jain. An infinite class of functions identifiable using minimal programs in all Kolmogorov numberings. I nternational J ournalofFoundationsofComputer Science*, 6(1):89–94, 1967.

A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

T. Huynh, M. Fritz, and B. Schiele. Discovery of activity patterns using topic models. In *UbiComp*, volume 8, pages 10–19, 2008.

G. F. Jenks. The data model concept in statistical mapping. *International yearbook of cartography*, 7:186–190, 1967.

E. Kim, S. Helal, and D. Cook. Human activity recognition and pattern discovery. *IEEE pervasive computing*, 9(1):48–53, 2009.

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.

C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7: 67–82, 1997.

G. Petasis, G. Paliouras, V. Karkaletsis, C. Halatsis, and C. D. Spyropoulos. e-grids: Computationally efficient gramatical inference from positive examples. *Grammars*, 7:69–110, 2004.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

J. Rissanen. *Stochastic complexity in statistical inquiry.* World Scientific, 1989.

E. Rogers, J. D. Kelleher, and R. J. Ross. Towards a deep learning-based activity discovery system. In *AICS*, pages 184–191, 2016.

E. Rogers, J. D. Kelleher, and R. J. Ross. In *Proceedings of the Second IFIP International Conference on Machine Learning for Networking*. Springer, Forthcoming.

J. Saives, C. Pianon, and G. Faraut. Activity discovery and detection of behavioral deviations of an inhabitant from binary sensors. *IEEE Transactions on Automation Science and Engineering*, 12(4):1211–1224, 2015.

Y. Shen, Z. Lin, C.-W. Huang, and A. Courville. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*, 2017.

Z. Solan, D. Horn, E. Ruppin, and S. Edelman. Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences*, 102(33):11629–11634, 2005.

W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.