Adaptive sampling of lion accelerometer data

Adam Cobb,   Andrew Markham
Dept. of Computer Science
University of Oxford

Abstract

The ability to track and monitor animals in the wild presents a challenge in both hardware and software development. This is a challenge facing all sensor networks, where there is a requirement for more computation to take place at the nodes whilst expending minimal battery life. In this paper we look at applying reinforcement learning on accelerometer data collected from a lion tag to learn a sampling control policy that adapts according to a lion's behaviour. The results show that it is possible to reduce battery consumption by 73 % with a reconstruction accuracy of 51 %.

1 Introduction

Tracking and monitoring animals in the wild is a key challenge with outcomes that inform conservation and management policies. In order to push the boundaries on the types of species that we can monitor, development must take place in both the hardware and software design. However, this challenge is not only limited to animal tracking, as it is becoming universal for all networks coming under the broad umbrella of 'internet of things'. A key challenge stems from the increasing requirement for more computation to take place directly on the nodes, whilst expending minimal battery life. This paper takes data collected from lions in southern Zimbabwe and looks at applying machine learning techniques to adaptively sample according to a lion's behaviour. In particular, we use the method of reinforcement learning to achieve unsupervised adaptive sampling. The objective is to apply these techniques to increase the battery life of the tags and to maintain the reconstructed data at an acceptable level for zoologists to apply further analysis.

The importance of understanding animal behaviour is not just purely pedagogical. Interest in animal tracking also comes from the value of understanding how certain animals spread diseases and from the need to improve our knowledge of wildlife conservation. In the paper by Wilson et al. (2013) [1], they describe the hunting dynamics of wild cheetahs based on measurements such as GPS and accelerometer readings. The work by Wilson et al. is the first to document the hunting characteristics of a 'large cursorial predator in its natural habitat' and highlights the difficult compromise between high accuracy and long battery life. The sampling strategy that Wilson et al. implement is based on their prior knowledge of the likely active times of the day, which is combined with sensor level thresholds being triggered.

Although often effective, basing sampling strategies that use heuristics can be restrictive and not very adaptable. A recent study into the location tracking of wild flying foxes [2] (2016) focussed on a technique referred to as 'Energy- and Mobility-Aware Tracking'. Sommer et al. test three sampling strategies, one of which uses cheap sensor readings to extrapolate the current position of the flying fox from its last GPS reading. The confidence in the extrapolation is given by the variance and once the variance reaches a threshold, another GPS reading is taken. The success of an adaptive sampling strategy, such as in [2], provides motivation for exploring alternative techniques for producing sampling strategies.

Ideally one would like to have a generic sensor that could be attached to any animal and learn anticipated patterns of activity in order to adaptively schedule sensor sampling. This is particularly important for sensors that can not be duty-cycled rapidly, such as GPS, which has a high cost of activating.

Section 2 introduces the accelerometer data set by highlighting its significant features. Q-learning is described in section 3, and the implementations followed by the results are given in sections 4 and 5 respectively. These results are then commented on in section 6.

2 The accelerometer data

This section introduces the data and identifies the key aspects to focus on to tackle the task of adaptive sampling. The overall objective is to reduce the average energy requirement of the sensor whilst keeping the level of accuracy satisfactory for analysis by zoologists. When the lion is active, to achieve a high accuracy, we must sample at a higher rate. Therefore the objective is to sample at a higher rate when the the lion is active and to sample at a lower rate otherwise, such as when the lion is sleeping. The accelerometer data covers 9 days for a particular lion and records an average reading along with variance in each of the x, y and z directions at every second.

The data was collected from lions living at the Bubye Valley Conservancy (BVC) in southern Zimbabwe [3]. Initially the lions were equipped with just accelerometer bio-loggers that could last 4-8 months. Some of which were subsequently replaced with updated bio-loggers that were able to record audio. The capability to continuously record audio dramatically reduces the battery life of the logger, giving an expected life span of 10-14 days. Therefore there would be a significant advantage for further studies if the sampling strategy could be configured on the device to increase the life span.

Adapting the sampling strategy according to the behaviour of the lion means exploiting features from the collected data that are informative about the animal's activity. Looking at Figure 1, it is possible to highlight significant features of the data. The average accelerometer reading in each of the three axes is displayed in the top plot along with the corresponding variance in the axes below. Note that the variance in the figure is smoothed with a median filter for clarity. These samples are taken at 1 Hz. Regions of low variance, such as from around node time 182000 and onwards, are normally interpreted as periods of resting or sleep. The step changes in the accelerometer readings that take place during the resting periods often correspond to the animal changing position in its sleep. Examples of the step changes can be seen in the top graph of figure 1, two of which occur slightly before both 184000 and 187000. In regions of high variance we expect the lion to be active and therefore are interested in sampling at a higher rate to monitor behaviour believed to contain richer information.

Along with the accelerometer data, the tag has access to the Coordinated Universal Time (UTC), which makes it possible to produce representations of the data such as Figure 2. This figure contains the mean hourly variance readings plotted against the time,

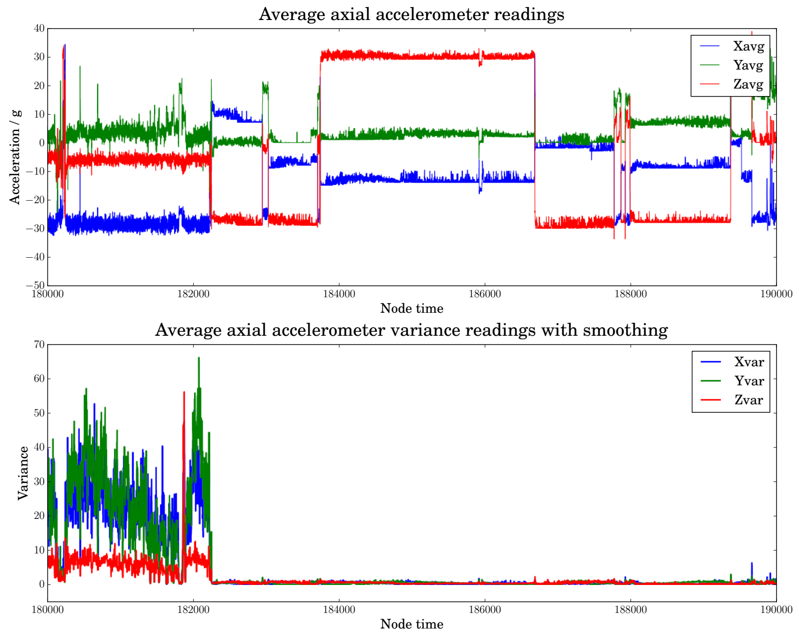

Figure 1: The upper plot displays the average accelerometer readings for each axis, measured every second. The aligned lower plot shows the corresponding variance associated with the above readings at each second. A median filter has been applied for smoothing across the variance values for clarity. The node time is the number of seconds since the tag has been running and this figure contains a slice of this time.

starting at hour zero when the tag starts recording at GMT time, 18:00:00 21 November 2014. We can see that the lion's mean variance tends to peak during the night, confirming that lions are predominantly nocturnal.

2.1 Accuracy and energy usage

The raw accelerometer data is sampled at 1 Hz. This original data is defined as the ground truth for the duration of this paper and provides a target for the most accurate reconstruction. An appropriate measure for the accuracy and one which is used, is to take the mean squared error (MSE) between any reconstructed accelerometer readings and the original readings. As the data is collected over the three axes, the MSE is calculated by averaging across all the three axes.

The data only consists of accelerometer readings. It follows that it is safe to assume that each sample expends one unit of energy. However, if extending to multimodal data that includes audio and GPS readings, the large discrepancies in the energy requirements from the respective sensors would have to be taken into account.

3 The application of Q-learning to the data set

In this section, the theory of reinforcement learning is introduced in the form of Q-learning. Following an overview, two different implementations are presented.

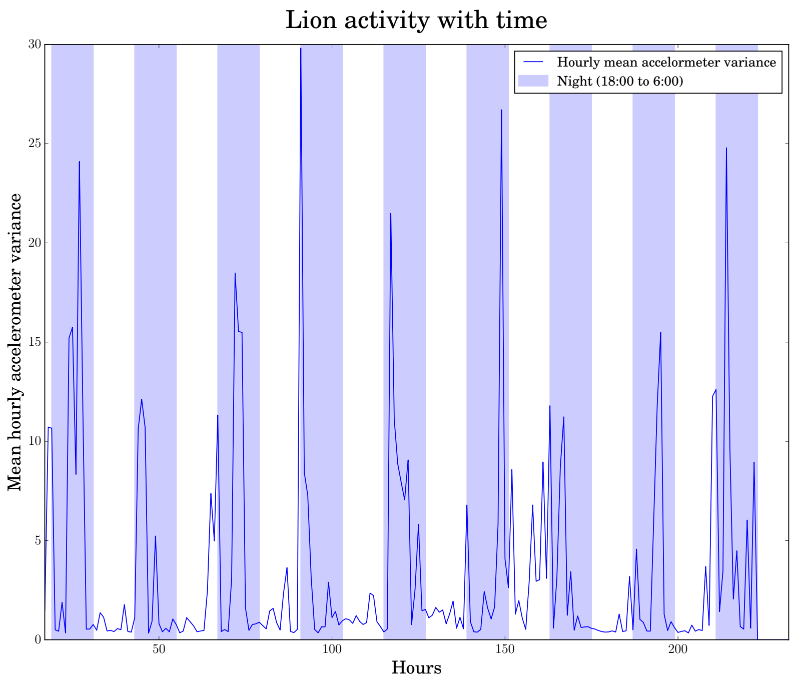

Figure 2: The mean hourly variance plotted against UTC time. The time of day is split into daytime and night-time, where the background blue shading corresponds to the night-time readings.

3.1 Q-learning: theory and rewards

Reinforcement learning is a technique that enables agents to learn about how to select actions as they interact with an environment, with the aim of achieving a certain goal. As an example, the agent could be a robot and the environment could be a maze that it must navigate to get to its goal state. The complexity of these environments can range from a static and completely observable environment to one which is dynamic and only partially observable to the agent. Rewards are assigned to an agent once it has acted upon its knowledge of the state. These rewards inform the agent how well it is doing and whether it was correct to select the last action. Therefore the aim of the agent is to choose actions in states that it has learnt will give it the highest reward. The reward can be defined in terms of either looking ahead to the next time step or including a weighted sum of all future rewards. An overview of reinforcement learning along with some of the different variations in methodology can be found in Sutton and Barto's book 'Reinforcement Learning: An Introduction' [4].

Q-learning is a reinforcement learning algorithm, where the Q refers to the action-value function that takes a state and an action as an input and outputs the value of the state-action pair. Introduced by Watkins (1989) [5], the algorithm updates the Q-values according to

$$\mathbf{Q}(\mathbf{S}_t, A_t) \leftarrow \mathbf{Q}(\mathbf{S}_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a \mathbf{Q}(\mathbf{S}_{t+1}, a) - \mathbf{Q}(\mathbf{S}_t, A_t) \right]. \tag{1}$$

This equation uses the same notation as [4], where α, γ and R_{t+1} each correspond to the learning rate, discount factor and the reward received at the next time step respectively. The learning rate determines the size of the update step in learning the parameters associated with the Q function. These parameters depend on the version of the algorithm being implemented, and are described in the subsequent sections. The discount rate weights the significance of future rewards when updating Q and fits $0 \leq \gamma \leq 1$. A discount rate of zero only takes into account immediate rewards, whereas values nearer to 1 force the agent to be more concerned about the possible future values of Q that it could receive. R_{t+1} is the reward received at the next time step after being in the state, $\mathbf{S}_t$, and taking action, A_t. The form in which equation 1 is written highlights that the $\mathbf{Q}(\mathbf{S}_t, A_t)$ is updated by the difference between its current value and the combined total discounted rewards received at the next step, where this update is controlled by the learning rate. Note that the future rewards are calculated by taking the maximum over the actions given at the next time step, which is then discounted.

Therefore to do a one step Q-learning update the tuple, ($\mathbf{S}_t$, A_t, $\mathbf{S}_{t+1}$, R_{t+1}) contains all the required values. These consist of the state-action pair at the current time step along with the state and received reward at the next time step.

Furthermore, as the requirement is a continuous online interaction with the environment, there is always a compromise between exploiting the current state by picking the highest expected reward and exploring less-well known areas of the state space that may lead to larger rewards. This exploitation vs. exploration paradigm often depends on our knowledge of the system. An example could be to set the exploration at a high level at the start, enabling the parameters of the Q function to be learnt faster at the beginning. This rate can then be reduced with time, as the algorithm's confidence in the behaviour of the system increases.

3.2 Tabular Q-learning

The action-value function, Q, can take different forms. Selecting an appropriate representation depends on the number of actions and the size of the state space. It can also depend on the computation power and memory availability of the system implementing the algorithm.

If the state space is relatively small and the number of actions is equally small, then every combination of the state and action space could be represented in a lookup table. For the lion data, the state space can be split up into discrete inputs to ensure the lookup table is kept at a manageable size, whilst containing the necessary features required for selecting appropriate actions.

In this paper, the state space is split according to the time of day and the variance level. To keep the table small, the variance level is given two discrete labels, *high* and *low*. The threshold for the *high* level is empirically chosen to be 5 as the data suggests this to be a suitable level. Although there are various ways of dividing the time into discrete periods, an example is to take each hour of the day as one of the states. Therefore this gives a state space of size 48, due to there being 24 hours for both the *high* and *low* variance levels. Given the two actions associated with the two different sampling rates, this lookup table would have a dimension of 48×2. Hence it is possible to imagine that the number of elements in the table increases exponentially in the number of states. Another example could be to split the day into quarters to reduce the table's dimensions to 8×2. If it is necessary to have a large state space, the problems of representing this state space can be overcome by approximating the table of Q-values with a function that outputs a value for each state-action pair.

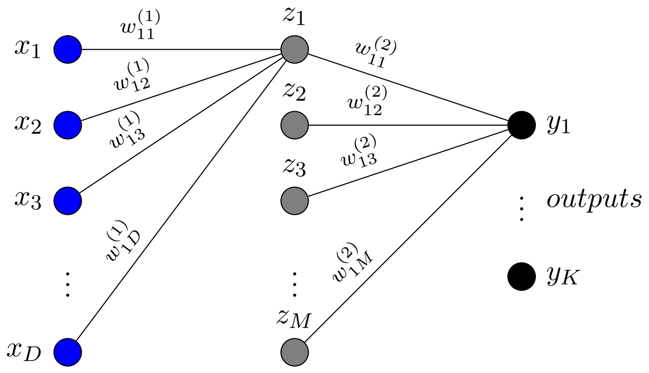

Figure 3: A neural network with two hidden layers containing D inputs, M neurons in the first layer and K in the second layer. All the layers are fully connected, however for clarity only the neurons outputting z_1 and y_1 are shown connected.

3.3 Deep Q-learning

One possible approach to dealing with too large a state space is to approximate the table with a neural network (NN). As an example, Tesauro's work on the computer program, 'TD-Gammon' (1995) [6], used neural networks along with backpropagation to achieve near human expert level at backgammon. However more recently there has also been a lot of work in combining Q-learning with NNs, due to the progress being made in deep learning [7, 8].

3.4 Neural network overview

A NN consists of layers of *neurons*. Each neuron, i, receives a weighted input, $\mathbf{w_i^{(1)}} \bullet \mathbf{x}$, where $\mathbf{x}$ is an input vector containing elements $1 - D$ and $\mathbf{w_i^{(1)}}$ is a vector of weights corresponding to each neuron. The superscript refers to the layer associated with the weights. The weighted input is passed through a non-linear function called the activation function, $h()$. The output of each neuron is a scalar value, z_i, normally limited to reside between -1 and 1 or 0 and 1. The non-linearity causes each neuron to behave as a switch that turns on when stimulated. A single layer's outputs $\mathbf{z}$ can then be treated as the input vector to the next layer. This nomenclature, taken from Bishop [9], is displayed in figure 3, which displays the feed-forward architecture for two of the neurons in a NN, with two hidden layers. The equation for the output from a single neuron, i, in the first hidden layer is:

$$z_i = h\left(\mathbf{w_i^{(1)}} \bullet \mathbf{x}\right). \tag{2}$$

The weights of the NN must be learnt through backpropagation, which is the process of applying the chain rule from the output, to calculate the derivatives with respect to the weights of the neural network. These derivatives are then used to update the weight parameters. In the context of reinforcement learning, equation 1 is the objective function that the NN is optimised to approximate.

Due to the nature of reinforcement learning, it is not always clear how to implement online training of a NN to ensure that it generalises well. In 'TD-Gammon', the stochastic nature of backgammon meant that the training data contained an even distribution of examples, enabling the neural network to generalise to entire state space. In many reinforcement learning applications consecutive samples are highly correlated and this leads to several disadvantages pointed out by Mnih et al. [7]. One issue from updating on many subsequent samples all residing in a small area of the state space, is that it can cause the network to get stuck in local minima. In fact, this problem is particularly significant for the Lion data, as each hour contains 3600 samples. Therefore training at each time step on this uneven data would be not produce the desired results as the weights get stuck. The solution given in [7] advises using a technique called *experience replay.* Instead of updating the weights at each step, the tuple consisting of ($\mathbf{S}_t$, A_t, $\mathbf{S}_{t+1}$, R_{t+1}) is stored in the experience replay memory. A randomly selected batch of experience replays are then sampled and used to train the network. Thus, encouraging the batches to contain a broader spread of state-action pairs. In some cases, if there is prior knowledge available, it may be possible to preprocess the experience replay memory to contain a diverse distribution of samples. As an example, in the lion data set we can ensure that an even number of samples from each hour is stored in the experience replay memory.

3.5 Extension to recurrent neural networks

As an alternative to feedforward neural network Q-learning, recurrent neural networks (RNNs) can be adopted instead. RNNs, unlike the previously mentioned feed-forward networks, include cycles in their topology [10]. A cycle could take the form of a connection between a layer and itself. This would make a layer, $\mathbf{z_t}$, a function of itself at a previous time step, $\mathbf{z_{t-1}}$, along with the current input values, $\mathbf{x_t}$, and with parameters $\mathbf{W}$.

$$\mathbf{z_t} = f\left(\mathbf{z_{t-1}}, \mathbf{x_t}; \mathbf{W}\right). \tag{3}$$

The recurrence, evident in equation 3, enables RNNs to model time dependencies in the inputs. Therefore an RNN is able to take into account previous states of a lion's behaviour as well as the current states, which means using an RNN would exploit the data's structure. They still have the advantages of the typical feedforward architectures, in that the number of weights required is smaller than storing every value in a table. Thus, mapping better to the challenges of resource constrained optimisation.

However, training RNNs is difficult due to issues with exploding and vanishing gradients as mentioned in [11]. One way of resolving this issue is by utilising a different architecture, the Long Short Term Memory (LSTM) architecture [12]. An LSTM combats the vanishing gradient problem by including a gating mechanism within each layer. In order to gain further insight into the LSTM architecture a good reference for the formulae is in [10].

Figure 4 shows the deep Q architecture implemented in for this data set. The LSTM layer receives a sequence of three consecutive state inputs in time, $\mathbf{x_{t0}}$, $\mathbf{x_{t1}}$ and $\mathbf{x_{t2}}$. The LSTM layer feeds into a dense layer, which is followed by a softmax layer that assigns values to the two different actions, Q_{a1} and Q_{a2}.

3.6 Implementation on the lion accelerometer data

Implementing Q-learning on the lion accelerometer data goes through the following stages:

- Initialise the Q-function parameters, either the tabular values or the network weights.
- Select whether to explore or exploit according to the exploration-exploitation probability.
    - If explore:

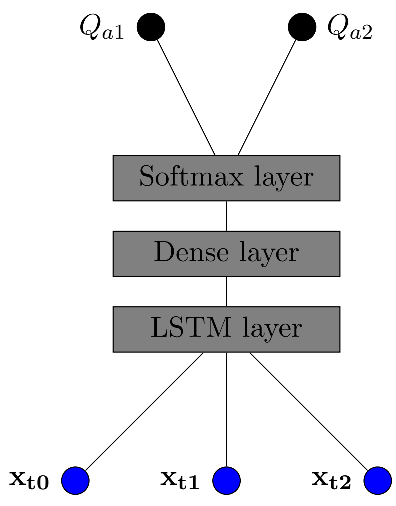

Figure 4: Network architecture for deep Q-learning on the lion data set. The LSTM layer takes in a sequence of three consecutive states in time. The output is passed to a dense layer and then a softmax layer, which approximates the Q-values for the two actions.

    * Sample at the maximum rate and assign rewards for the different sampling rates. The results are either stored in the experience replay or used to directly update the table depending on the Q function approximation.

- If exploit:
    * Select the action corresponding to the highest Q value given the current state.

This process is repeated with the exception that for deep Q-learning, the network is trained at the end of the cycles when the experience replay memory is full.

Although the concept of giving rewards for actions has been mentioned, it is not necessarily clear how these rewards are to be defined. We must penalise sampling at a high rate when energy is being used to monitor a stationary lion and reward sampling at a high rate when the lion is moving around.

A possible reward function, received at the next time step R_{t+1}, based on the MSE between the original readings, x_s, and the reconstructed readings, $\hat{x}_s$, could be the following:

$$R_{t+1} = \begin{cases} -\frac{1}{S}\sum_{s=t}^{t+S}(x_s - \hat{x}_s)^2 - \lambda, & \text{if sampling period is } S_l \\ -\lambda S, & \text{if sampling period is } S_u \end{cases} \tag{4}$$

where,

$$S = \frac{S_l}{S_u} \tag{5}$$

The S_l corresponds to the sampling period of selecting the lower frequency action and S_u to the upper. The reconstructed readings are taken as a linear interpolation between the time at t and the time at the next sample, $t+S$. The compromise between the accuracy and energy is given by the value of λ. The λ penalises the number of samples and therefore the expenditure of energy. When sampling at the lower rate, only one sample is taken and therefore the energy penalty is λ. At the higher rate, S samples are taken and is penalised accordingly.

Unfortunately equation 4 produces a reward function that is too noisy for learning the parameters of the Q function consistently. An alternative reward function and one that appeals to reinforcement learning due to its simplicity is counting the number of samples with a high variance. Therefore sampling at a higher rate is rewarded according to the number of high variance samples in a batch. The $-\lambda S$ term is included to penalise the energy usage of sampling at the higher rate. A reading has a high variance if it is above a given threshold, v. The indicator function $\mathbb{I}\left(var(x_s) \geq v\right)$ in equation 6 equals 1 when the variance of the sample is above the threshold and 0 otherwise. This equation,

$$R_{t+1} = \begin{cases} 0, & \text{if sampling period is } S_l \\ \sum_{s=t}^{t+S} \mathbb{I}\left(var(x_s) \geq v\right) - \lambda S, & \text{if sampling period is } S_u \end{cases} \tag{6}$$

like the equation 4, is only applied during the exploration phase. Although for this particular reward function, if the higher sampling rate is chosen when in the exploitation phase, then rewards can still be calculated as they depend on sampling at the maximum rate.

So far the algorithms have assumed that there are only two actions, sampling at a lower rate or at a higher rate. However the algorithms can be adapted to include a larger choice of actions by extending the reward function to cover more rates.

3.7 The challenges of implementing on an embedded system

As the intention is to run these algorithms on an embedded system, it is important to mention the limitations of these devices. The current tag contains an 8 bit microcontroller with 8 KB of RAM. Future versions of the tag could have up to 128 KB on a 32 bit ARM core. Therefore computation and memory are key areas that limit the capability of applying machine learning techniques such as Q-learning on the bio-loggers. 8 KB of RAM would limit the number of weights used in the NN for deep Q-learning. Even using half precision floats would only allow for $4,000$ weight parameters. This number is to be compared with the $67,842$ parameters used in this report. It is very unlikely that, even with further experiments, the current architecture could work with the number of weights reduced by over an order of magnitude. Furthermore, the experience replay memory must be stored in order to apply online training. As NNs require a lot of data to train, the memory requirements for this procedure would not be well suited to the microcontroller. Finally, the order of complexity of the NNs comes from multiplying matrices in both the feed forward stage and the training stage. Returning to figure 3, producing all the outputs $y_1 - y_K$ requires $D \times M \times K$ calculations. This number of calculations is not desirable as one of the key concerns is conserving battery life and each calculation has an energy cost.

Slightly more promising is the tabular Q-learning as it is possible to control the size of the state-action space by splitting it into discrete values. As an example, 8 states and 2 actions would only use about 64 B of RAM at single-precision. Additionally, there is not the same requirement for storing the experience replays and the level of computation is much lower because equation 1 is the only calculation applied at each time step.

Due to the level of computation and memory requirements of the Q-learning algorithms, it is also important to test out a basic heuristic to see if the Q-learning provides the improvement needed to make it worth using. An appropriate heuristic is to look at the variance level over the past few samples and make a decision on the rate to sample the next batch of incoming readings. Section 4 gives details of this method.

4 Experimental Comparison

In this section, we describe the algorithms and experimental setup. In all of the algorithms, the choice of actions is either the maximum sampling rate of 1 Hz or the lower sampling rate of $^1/_{32}$ Hz. Thus the sampling periods for the upper, S_u, and lower, S_l, rates are equal to 1 s and to 32 s respectively. All algorithms are run on the nine day data set with the MSE and the total number of samples both recorded for each implementation. The MSE is used as the metric for accuracy, whereas the measure for energy usage is indicated by the samples taken. For each sampling scheme, a pareto-optimal curve is produced by varying the regularising parameter, λ. We now introduce the details of each of the sampling strategies that produce the results in section 5.

4.1 Heuristic approach

The heuristic-based approach follows equation 6 but looks at the last S samples instead of calculating the reward over the next S samples.

4.2 Offline oracles

The two offline methods assume that all the data is available. Each of the algorithms separately follow equations 4 and 6. Thus the ideal pareto-optimal curve is given by calculating the reward according to each algorithm's equation and retrospectively selecting the action at each time step that gives the highest reward. The offline method that uses equation 4, directly compares the MSE with the number of samples taken. Thus providing the upper bound on performance. The alternative offline scheme uses equation 6, which is the Q-learning reward function. Therefore, this sampling scheme gives the expected upper bound performance for the Q-learning methods.

4.3 Tabular Q-learning

For this tabular method, the state space is partitioned into eight discrete values. The time is divided into the hours between $0-6$, $6-12$, $12-18$ and $18-24$. Each state in time can then either take the low or the high variance value according to the threshold. The value of 0.9 is chosen as the probability of exploitation.

4.4 Deep Q-learning

To implement the deep Q-learning algorithm, we use the Keras library in Python [13] with a Theano backend. The first layer is an LSTM layer that receives a sequence of the last three input states, where the input state contains the hour of the day, the mean variance across the axes and the three accelerometer readings. The output of the LSTM layer is passed to a dense layer of 128 units, which then feeds to a softmax layer to produce the two Q values for the two action choices. The probability of choosing exploitation over exploration is also chosen as 0.9.

4.5 Random selection

The random selection sampling scheme gives a lower bound on the performance. Unlike the other algorithms, the pareto-optimal curve is produced by varying the action selection according to a probability that moves from 1 to 0 for the highest sampling rate. Therefore, each experiment has a constant probability of selecting each action throughout one implementation. We expect all sampling strategies to do better than this lower bound.

5 Results

Figure 5 displays the pareto-optimal curve for the implementation of the different sampling strategies. The vertical axis is the number of samples taken when running an instance of an experiment over the nine days of accelerometer data. The horizontal axis is the MSE, defined as the sum of the squared errors between the original accelerometer readings and the reconstructed accelerometer readings. Each sampling scheme is colour coded, where each point within a scheme varies according to the λ parameter, except for the random sampling scheme.

In order to interpret the results of the figure, the vertical axis can be viewed as being directly proportional to the energy usage and the horizontal axis as the accuracy. As an example, taking a point from the heuristic sampling scheme, it is possible to achieve a 73 % reduction in battery usage with a reconstruction accuracy of 51 %. This is calculated by treating the maximum MSE as the one given by sampling at the lowest rate for the entire duration of the data set. It then follows to calculate the reconstruction accuracy as a linear sliding scale between this maximum, and a zero MSE when exactly reconstructing the original data. The reduction in battery life comes from comparing the number of samples taken during an experiment with the number of samples at the maximum rate. Depending on the demands, it is possible to pick a sampling strategy and then set the parameters according Figure 5.

6 Conclusion

The results in figure 5 suggest that using the simple heuristic method to select the sampling rate will perform the best. However, the performance of the tabular Q-learning method is not significantly worse than this heuristic-based approach and there is potential for the tabular approach to tend to the heuristic one with more data. Both the larger complexity and memory usage of deep Q-learning suggest that it is not worth pursuing this method further, unless there are changes in the size and structure of the data or if the objective of the problem is changed.

The weak structure of the data may have hindered both Q-learning strategies because attempts to find structure in the data from the reinforcement techniques could present too much of a challenge. This weak structure could also explain why the gap in performance between the offline and online strategies is significant, due to the difficulty in making good predictions on the unstructured data. The accelerometer data only covers nine days for one particular lion and it may not be possible to infer any consistent daily behaviour from this sample size. This could explain why using a heuristic that takes a look at the variance of the previous few time steps has performed so successfully.

6.1 Difficulties with training for deep Q-learning

The difficulties in training the RNN is apparent from the worse pareto performance curve. These difficulties may stem from the fact that NNs require a lot of data and the data available may not be enough to achieve optimal performance from the deep Q-learning paradigm. As an example, applying the same architecture for character prediction needs

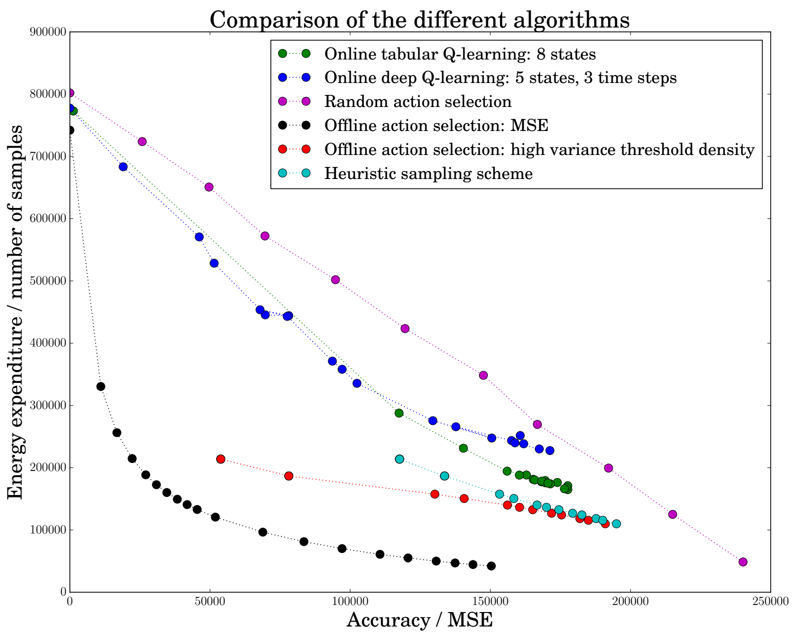

Figure 5: This figure displays the optimal pareto curves when compromising between the number of samples taken and the MSE for all of the experiments. The black and the red curves are both offline sampling schemes. The black curve follows equation 4 and the red follows equation 6. The magenta curve is a sampling scheme that selects the highest sampling rate according to a probability that varies to produce the pareto curve. The cyan curve is based on the heuristic discussed in 4.1. The blue and green points correspond to the pareto curves of the deep and tabular Q-learning sampling schemes respectively.

around 100,000 examples before it starts to achieve a notable level of performance [14]. Another issue comes from the data not being smoothly distributed across the state space. Therefore it is difficult to ensure that the experience replay memory contains an even population of tuples that cover the available state space. This coverage is a requirement for ensuring the NN does not get stuck in local minima and generalises well.

The gains to be made from deep Q-learning algorithm might be more attuned to multi-modal data that combines the accelerometer data with audio and GPS readings. Incorporating inhomogeneous sensor readings would add another level of complexity to the problem that would need more sophisticated sampling strategies. These sampling strategies would have to take into account the energy expenditure and information content when sampling from different sensors. In this case, a deep neural network might perform better at extracting this potentially much larger space of features. However, the addition of further data streams would also encounter many of the same training problems and may not outperform tabular Q-learning or a more advanced heuristic.

6.2 Suggestions for further work

As the heuristic performed better than the reinforcement learning methods, this should be taken as the baseline to improve on in further research. A possible improvement to be made on the heuristic method could be to combine it with tabular Q-learning to adapt the variance threshold level according to the time of day. As an example, if the lion were more active at night, the tabular Q-learning would be used to learn a lower threshold to make it more likely to trigger the higher sampling rate at night.

Furthermore, it would be worthwhile in performing the same experiments on other lions to see if the reinforcement learning techniques develop individual strategies for each lion. Learning these strategies may in itself be a useful tool for comparing how different lions behave and whether certain ages or sexes of lions are active at different times.

Putting these algorithms on embedded systems is another challenge and also a step that must be taken in order to make further progress. A benefit of the higher performance of the heuristic method is that it is also the most suited for implementation on such a device. This benefit is also true of tabular Q-learning, as long as the state space is limited to easily fit in the memory. In comparison, a lot more work is needed to be able to put a deep network on a small, low energy device such as the ones that are used for tracking the lions.

References

[1] Alan M Wilson, JC Lowe, K Roskilly, Penny E Hudson, KA Golabek, and JW McNutt. Locomotion dynamics of hunting in wild cheetahs. *Nature*, 498(7453):185–189, 2013.

[2] Philipp Sommer, Jiajun Liu, Kun Zhao, Branislav Kusy, Raja Jurdak, Adam McKeown, and David Westcott. Information Bang for the Energy Buck: Towards Energy- and Mobility-Aware Tracking. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, pages 193–204. Junction Publishing, 2016.

[3] Paul Trethowan, Andrea Fuller, Anna Haw, Tom Hart, Andrew Markham, Andrew Loveridge, Robyn Hetem, Byron du Preez, and David W. Macdonald. Getting to the core: surface temperatures reveal neither the ecological function nor the thermal implications of the lion's mane, 2016. Unpublished.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. 1998.

[5] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

[6] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[8] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[9] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[10] Bernardo Pérez Orozco. A succinct introduction to LSTMs, 2016. Unpublished.

[11] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[14] François Chollet. Example script to generate text from Nietzche's writings. https://github.com/fchollet/keras/blob/master/examples/lstm_text_generation.py, 2016.