# Hack The Monkey: a good trade off between speed and efficiency

A. Agostinelli, A. Lerede, V. Pacifico and M. Sarrico
(Hack The Monkey Team)
Dept. of Bioengineering, Imperial College London, London, UK
emails: aa7918@ic.ac.uk, al918@ic.ac.uk, vap2718@ic.ac.uk, mvs918@ic.ac.uk

***Abstract*—In the past years there has been an increasing interest in creating efficient algorithms able to decode brain activity. The study presented focuses on decoding the arm movement of a monkey utilising, as input, the neural activity recorded from the brain. The aim was to both understand the location of the object the animal was reaching and the trajectory of its arm. This problem was therefore divided into two different steps. Firstly the algorithm had to decode the location of the object. This requires a multi-class classification technique as there were 8 fixed angles the monkey could reach. Finally the arm position at every time-step (the trajectory) had to be defined. To do so a real time decoding technique was needed.**

*Keywords– BMI, Decoding, Signal Processing, Machine Learning, Classification, Regression*

## I. Introduction

The brain activity of a monkey moving its arm was recorded. These data were used to design a supervised learning algorithm able to predict which angle the monkey was reaching and find the hand trajectory at every time-step. The first problem required a classification technique: given the neural spike train it was necessary to decode which of the eight angles the animal was aiming at [2]. The second and final task was to define the hand position of the monkey in the Cartesian space at any given time t. Since the monkey had an infinite possibility of movements and the output was therefore defined as continuous, a regression technique was needed [2].

## II. Data Pre-processing

The data recorded were stored in a structure composed by 800 fields (8 angles x 100 trials). Each field containing: the trial id, the firing rate detected by the 98 electrodes over the recording period and the hand position, described in the x,y and z coordinate plane. The data were filtered from noise and represented by binary numbers.

As every trial had a different duration and the hand movement was occurring at different times it was decided to avoid comparing the neural activity at a given time steps over the different trials. It was instead decided to average the spikes recorded by each electrode over time and utilise these rates as inputs to decode the hand position. To do so it was assumed that the different electrodes were recording different firing rates depending on the target angle. By analysing the data the previous hypothesis was confirmed. Thus, to perform angle classification, it was possible to use one firing rate to represent the activity recorded by one electrode during one trial.

## III. Angle Classification

The first step to identify the trajectory was to compute the angle classification; a simple k-nearest-neighbours model ($k$-NN), applied on the firing rate of the 98 signals, performed particularly well in terms of accuracy and computational time. An alternative was to compute the ISIH (Inter Spike Interval Histogram) for any given unit. However, the latter had a worst performance compered to evaluating the difference among firing rates. For this reason the idea was discarded. Hyper-parameters such as the number of neighbours (**k**) and the distance coefficient (**c** as in **Algorithm 1**) were found by grid search and by dividing the dataset in Training Set (70%) and Test Set (30%). Results are shown in figure (1). The overall best accuracy was **0.9708**, obtained with $k = 52$ and $c = 1$.

## IV. Classify the trajectory

### A. Linear regression

Supposing that the angle classification resulted in a 100% accuracy, a linear mapping was implemented to estimate the arm trajectory. This means that the errors estimated in this section only refer to the classification of the trajectory. To set up the linear regression model correctly it was necessary to consider: how to manipulate the inputs, the different spatio-temporal features that the neurons may be coding for and the temporal delay between neuronal spikes and arm response.

The neuronal spike trains was fed into the system in the form of firing rates. The optimal number of time

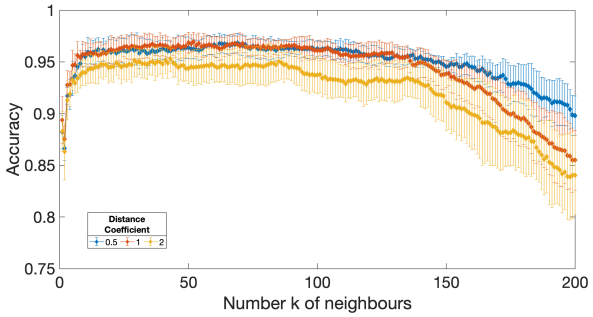


Fig. 1. Classification accuracy for several number k of neighbours, showing mean and standard deviation of 10 runs on 3 different distance coefficients.

steps considered to evaluate the temporal average needed to be estimated. This hyper-parameter of the model was defined as: **TimeSteps**.

The recorded neurons could have behaved as a population, i.e. receiving the same inputs and coding for the same spatio-temporal features or may belong to different sub-populations. For instance, they may be divided in sub-groups activated for different arm orientations. For the sake of simplicity, the 98 recorded units were assumed to belong to the same population, i.e. coding for the same features: either arm position, velocity or acceleration. Therefore, three different models were trained having as labels respectively: **arm position, velocity and acceleration**. Arm velocity and acceleration were calculated utilising the spatial position averaged over the same time steps previously used to evaluate the firing rates.

Finally, the optimal **TimeDelays** hyper-parameter had to be defined. There is a time lag between the neural activity in the brain and the arm movement as the information has to travel from the brain to the arm motor neurons. Since the time steps considered were fairly small (1ms), the arm position recorded at some point in time could have referred to the brain signal several milliseconds ahead. For this reason, it was necessary to shift ahead the neuronal recordings of an optimal number of time steps.

Therefore, three different models were trained and tested (50% of the data as training and 50% as testing), and the optimal hyper-parameters for each of those were estimated by varying:

1) **TimeSteps** $\rightarrow$ 1ms:20ms
2) **position, velocity and acceleration of the arm as labels** (three different ways to train the linear regression model)
3) **TimeDelays** $\rightarrow$ 1ms:5ms:200ms

The performances of all possible combinations were evaluated and compared in term of root mean squared error (RMSE). The best RMSE, ie. the lowest, was found to be **23.9491** for: **TimeSteps = 20ms**, **TimeDelay = 2ms** and by using the **velocity model**. This was the average RMSE across the 8 angles. The best value was found for the $4^{th}$ orientation and was **21.2487**. By plotting the TimeSteps and the TimeDelays versus the RMSE for each angle, the same trend can be observed across all angles. Therefore, the results reported below refer to the average RMSE across angles and are representative of all orientations.

From Fig. 3, it is clear how the velocity and acceleration models are very similar and outperform the position model, yielding always lower errors. Despite the similarity, the velocity model is slightly better then the acceleration one. Therefore, the neurons, considered as a unique population, seem to code for arm velocity.

Fig. 3a reveals how the error is lower for lower values of delays, i.e. the signals seem to travel down the axons within a time frame in the order of units of milliseconds. Fig. 3b shows a minimum at TimeSteps = 20ms for all three models, making it the obvious choice as hyper-parameter. Fig. 2 shows more clearly the combined trend of the TimeSteps and of the TimeDelays corroborating all the above.

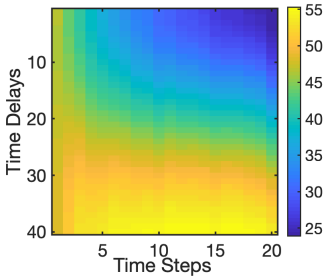


Fig. 2. RMSE Map for various combinations of TimeSteps and TimeDelays for the velocity model

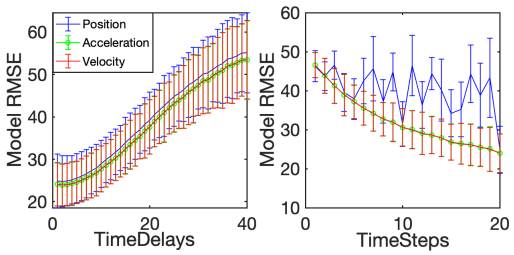


Fig. 3. a (left): TimeDelays trend for TimeSteps = 20ms for the three models — b (right): TimeSteps trend for TimeDelays = 2ms for the three models

### B. Long Short-Term Memory

**Recurrent Neural Networks**

Recurrent Neural Network (RNN) is a class of artificial neural networks characterised by having an internal memory, a property that enables it to be used in applications with time series or sequential data. In fact, in the feed-forward neural networks (NN) the information just flows forward, from the input layer to the output layer, passing through the hidden layers. Thus, the current input is the only one that is considered. In RNN, there are cyclic connections that enable the nodes in the recurrent layer to consider not only the current input but also what it has previously learned for past recent inputs [1]. Therefore, due to its internal memory, a RNN produces an output that is then copied and looped back into the network, enabling this method to be used for prediction using sequential data like time series data.

**Long Short-Term Memory**

Traditional RNNs are limited by the vanishing gradient problem, where the learning of the neural network significantly decreases. This happens since the gradient is vanishingly small, meaning that the weights are no longer updated. Long Short-Term Memory (LSTM) networks are RNNs that address this problem by extending RNNs memory. In LSTM networks, a memory unit called the cell is added to keep previous values of arbitrary time intervals. The information that flows in the cell will depend on an input gate, an output gate and a forget gate [1]. These networks are used in time series predictions since important information in the series may have an unknown time lag between those two useful events and they are capable of storing those past events and use them later as input. By considering this and also that the recorded data consisted of the neural activity time series, LSTM networks for sequence-to-one regression were implemented. Sequence-to-one regression in this case means that the networks map a sequence input, the time series, to one value output, the position of the arm, given by the 2 coordinates, at a specific time step. Since the first part of the model involves a $k$-NN algorithm for detecting the angle that the monkey was aiming for, at this stage the angle is considered to be already known. Thus, for predicting the trajectory of the monkey's arm, 8 LSTM networks were trained, one for each of the 8 angles. As input, the time series across 70 of the 100 trials were used to train the NNs and the remaining trials were used to test them. The network architecture consisted in 6 layers: an input layer, an LSTM layer, a dropout layer, two fully connected layers, and a regression output layer. The dropout layer was added in order to reduce overfitting. By changing the number of hidden units in the LSTM layer, the performance of the model changed a lot. Therefore, for each network this hyper-parameter was tuned to obtain a better performance.

To evaluate this model, the metric used was the RMSE. Fig.4 was obtained by training the 8 LSTM networks, showing the standard deviation and the RMSE averaged over all of them, for each of the 30 trials of the test set (selected randomly from the total 100 trials). As it can noticed the RMSE values were mostly under 20. Besides that, the time that took to train the model was also evaluated. The averaged time across the 10 times the network was trained was 1:06 minutes.

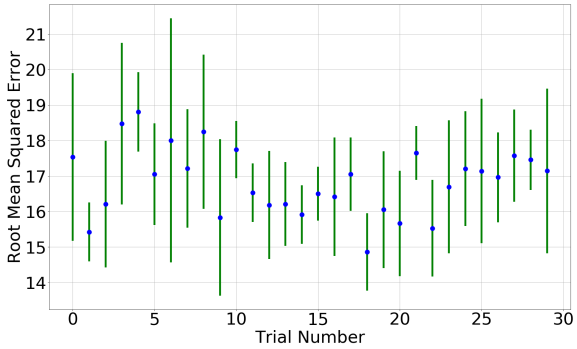


Fig. 4. Averaged Root Mean Squared Error over the eight LSTM networks for the test set trials (blue dot) and correspondent Standard Deviation (green line)

### C. Weigthed $k$-NN

The following approach is a non parametric evaluation of the position purely based on instances of past experiences. Instead of training a model, the regressors computed a weighted average of the $k$-NN positions at that given time window. In particular, the weights are computed as follow:

$$W \rightarrow \frac{1}{D^w + \epsilon}; \qquad \epsilon = 1e - 35$$

Where **D** is the distance calculated by the $k$-NN algorithm and the coefficient **w** changes the proportion of the weights among the neighbours. **Algorithm 1** explains the model for a single time step of 20 ms. This solution was particularly suitable for this problem as the trajectories were constrained by a limited space and well discernible in groups according to the orientation. Thus they were representative of the future trajectories to compute. In

this particular setting there was no need to develop a complete continuous model in the whole space.

***Algorithm 1*** *Weighted kNN for every time-step*

$Input \rightarrow Signal[\Delta t = 20ms](from - N - electrodes)$
*FiringRate(*$FR_j$*)* $\rightarrow mean(Input_j)$ *; j = 1 to N;*
**for** *(i = 1 to length(TestData))* **do**
  $Distance[i] \rightarrow (\sum_{j=1}^{N}(|\ (FR_{ij} - FR_j)\ |)^c)^{\frac{1}{c}}$
**end**
$Distance \rightarrow Distance$*[k smallest distances]*
**for** *(i = 1 to k)* **do**
  $Weight(W_i) \rightarrow \frac{1}{{D_i}^w + \epsilon}; \qquad \epsilon = 1e-35$
**end**
**for** *(i = 1 to k)* **do**
  $NormalizedWeight(NW_i) \rightarrow \frac{W_i}{\sum_{i=1}^{N} W_i}$
**end**
$C_x = \sum_{i=1}^{k} NW_i \times X_i*$
$C_y = \sum_{i=1}^{k} NW_i \times Y_i*$

$* X_i, Y_i = mean(Positions)[\Delta t = 20ms](Input_i)$

From **Algorithm 1** 3 hyper-parameters were extracted:

1) **c** $\rightarrow$ coefficient to calculate the distance between neighbours (Distance Coefficient in the figures);
2) **k** $\rightarrow$ number of neighbours;
3) **w** $\rightarrow$ coefficient to change the proportion of the weights.

The estimation of the hyper-parameters was performed through grid search, dividing the dataset in Training:Validation:Test as 0.65:0.15:0.20. The results obtained are illustrated in Fig. 5. The final error on the Test set was 10.28 using the following hyper-parameters: *k=33* , *w=5*, and *c=2*.

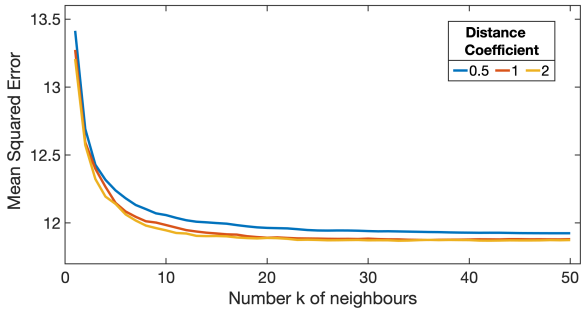


Fig. 5. Regression Mean Squared Error for several number k of neighbours and for 3 different distance coefficients, the parameter **w** is fixed at 5, as in the best setting.

## V. Discussion and Conclusion

When pre-processing the data further techniques could have been exploited. PCA is one of the most important as it has the potential of reducing the number of units recorded or to cluster them in groups, thus further decreasing the time and the computational costs. However, since the accuracy was already high and the computational time low, this approach was not adopted. Additionally, a K-Fold Cross-Validation could have been performed but the data dimension was large enough not to consider this step necessary.

To classify the angles the algorithm chosen utilises the firing rate and the $k$-NN classifier method. This simple model yields high accuracy and low computational time. Those reasons made it the best candidate for the task.

For what concerns the trajectory estimation, the linear regression method had to be excluded as the RMSE error was too high to describe an accurate trajectory. To pursue this method it would have then been necessary to either add a non-linearity to the model (of sigmoidal type) or to better evaluate the presence of sub-populations showing different behaviours. The latter would enable the reduction of the neuronal space of each specific orientation allowing to fit a less noisy linear regression model on each angle. This may improve the accuracy.

Similarly, although the LSTM method was a good solution to predict time series as its performance resulted in a low RSME, the computational time required to train the eight NNs was too high. For this reason, it was decided to reject the linear regression and the LSTM models and opt for a simpler method able to find a good trade off between speed and efficiency: the weighted $k$-NN method. This algorithm is simple to implement, requires low computational cost and predicts trajectories with low RSME. Despite this might not be the optimal method to decode brain activity and to implement real-time BMI applications, given these specific conditions, it was the preferred solution for this specific task.

## References


[1] Abdulmajid Murad and Jae-Young Pyun. "Deep Recurrent Neural Networks for Human Activity Recognition". In: *Sensors* 17.11 (2017), p. 2556. DOI: 10.3390/s17112556.

[2] Rajesh P. N. Rao. Department of Computer Science, Engineering & Neurobiology, and Behavior Program University of Washington, Seattle: Cambridge University Press, 2013.