Toward a Statistical Understanding of the MOS 6502 Microprocessor

Nicholas K. Branigan

June 5, 2020

1 Introduction

Over the past few decades, new tools such as modern multielectrode arrays, calcium imaging, and optogenetics have emerged for collecting brain data [1]. These methods have allowed neuroscientists to peer increasingly clearly onto increasingly great swaths of neural territory. In fact, progress in this area has been so impressive that we can now reasonably anticipate a future in which neuroscientists are no longer limited by the data they can collect. However, in their 2017 paper Jonas and Kording argue that neuroscience is not prepared for this future. They evaluate current statistical approaches in neuroscience on an early computer microprocessor, a fully understood system that has meaningful similarities with neural systems. They find that these methods fail to recover the workings of the computer chip [3]. I take Jonas and Kording's work not as an omen of doom for the neuroscientific project, but as a call to action. We need to be statistically prepared for the datasets that bio-engineers are working to make available, and the microprocessor can help us. We can test methods on it now, and we can compare the results against what we know about this system, which is virtually everything there is to know about it.

2 Related Work

As far as I am aware, researchers have yet to explicitly take up Jonas and Kording's call to action and work on trying to recover an understanding of the MOS 6502 microprocessor from its net-list and transistor state time series. The most closely related work to mine, then, is Jonas and Kording's.

The authors were able to write their paper because of the work of the Visual 6502 team. This group reverse-engineered the 6502 by applying a combination of human and algorithmic judgement upon images produced by a microscope trained on the exposed chip. They were able to build a complete net-list for the processor, which allows for a perfect fidelity simulation of its behavior [2].

Jonas and Kording analyzed both the net-list, which is a nice analogue to the connectome in neuroscience, as well as the time series of transistor states produced by the 6502 simulation while the chip renders three video games: Donkey Kong, Space Invaders, and Pitfall. They applied modern graph-analysis methods to the net-list. To strengthen the analogy to the brain, they transformed the time series of transistor states into one of transistor state changes. In other words, in their transformed data, 1 indicates that the transistor's state at time $t-1$ was 0 and at time t was 1, or that at time $t-1$ was 1 and at time t was 0. They then collapsed these data with time bins of 100 steps. Each time bin, then, includes the number of state changes that the transistor underwent during the 100 time steps corresponding to the bin. This is a rough analogue to a time series of neuronal spike counts, something neuroscientists are familiar with. The authors' most compelling analysis on this series is a dimensionality reduction that they perform with non-negative matrix factorization. This gives them a 6-dimensional vector for each time rather than a 3510-dimensional one (the 6502 has 3510 transistors). Their most impressive finding from the factorization is that one dimension is highly correlated with one of the chip's clock signals and another is highly correlated with the other clock signal. Meanwhile, a third dimension is highly correlated with the chip's read-write cycle. Though these are neat findings, they fall comfortably short of a genuine understanding of how the 6502 processes information.

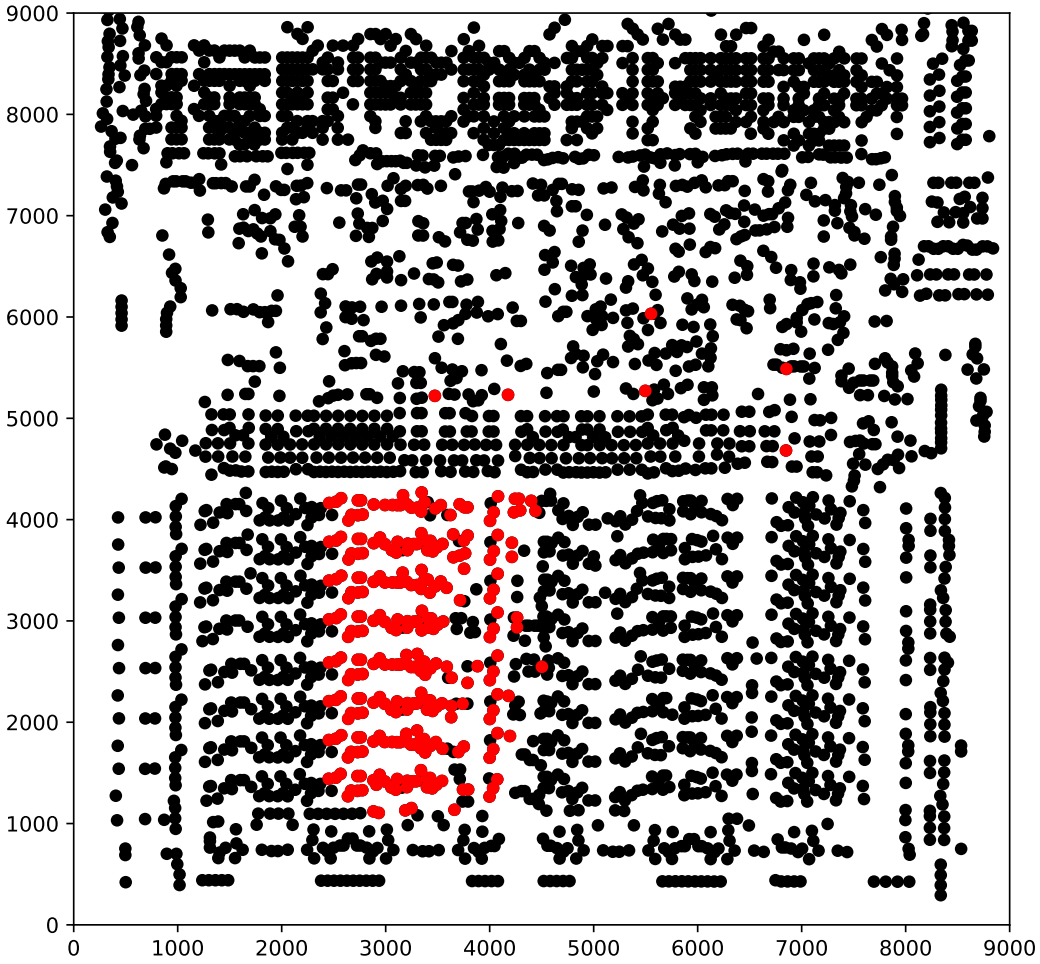

Figure 1: Locations of the 3510 transistors in the MOS 6502. Transistors that are a part of the ALU are marked in red. Transistors not in the ALU are in black.

3 Data

Eric Jonas has made most of the data used in the paper available, with instructions for how to access it located in the neuroproc data repository of his Github. The net-list is captured by two comma separated values files. One describes the positions of the transistors in the chip and the wires to which each is connected, and the other describes the wires with their canonical names (assigned to them by the Visual 6502 team). The "spiking" time series come in gzipped .npy files. After unzipping, each game's time series is around 50 GB. The files are 12,799,999 by 3510 arrays, where value i, j signifies whether transistor j did (1) or did not (0) change state at time i during the 6502's rendering of the video game. I follow Jonas and Kording in binning the spikes into time bins of size 100, giving for each game a 127,998 by 3510 array.

Using the net-list, I identified the 280 transistors in the MOS 6502 that make up its Arithmetic and Logic Unit (ALU). Figure 1 shows these transistors and their locations in the 6502. (The few scattered red dots above the primary rectangular cluster are involved in trafficking ALU outputs and inputs to and from other chip areas.) Then, I randomly chose 280 transistors not in the ALU. In the next section, I attempt to find methods that can identify which of these two sets a transistor belongs to based on its time series.

4 Methods

4.1 Feedforward Neural Network

The problem of recovering the functional role in information processing of transistors ultimately belongs in the unsupervised setting. We want to be able to discriminate between circuit components like transistors with different functional roles without any information about those roles. So, in this section, I merely lay a foundation for the next two by showing that in the presence of labels, we can find a highly nonlinear function that reliably classifies transistors as belonging to either the ALU or the random selection of 280 transistors outside of the ALU. This demonstrates that information regarding which of these sets a transistor belong to is hiding within the time series of transistor state switching activity. I do not take this to be obvious.

I have chosen a neural network with three hidden layers as my highly non-linear function. The hidden layers apply the Rectified Linear Unit (RELU) activation function $f : \mathbb{R} \to \mathbb{R}$, where

$$f(x) = \max\{x, 0\}.$$

The output layer applies the logistic sigmoid function $g : \mathbb{R} \to \mathbb{R}$ where

$$g(x) = \frac{1}{1 + e^{-x}}.$$

Training the network involves calculating its gradient with respect to the model parameters, which are the weights for each node. Then, training proceeds through gradient descent on a cost function. This function attempts to navigate the bias-variance trade-off by penalizing inaccurate predictions as well as large weight values through a regularization whose strength is controlled by a hyperparameter.

For this analysis, I further pre-process the transistor spiking series, which has been narrowed to a 560 by 127,998 array, using Principle Component Analysis (PCA). PCA identifies linear subspaces, which, when the data are projected onto them, preserve as much of the variance in the data as possible. I chose to reduce the dimensionality of the spiking series for each transistor from 127,998 to 100. I standardized the features by subtracting the mean and dividing by the standard deviation before applying PCA; in order for the results of PCA to be meaningful, the features must be centered and on the same scale. I carried out the PCA and standardization using scikit learn.

I trained the neural network on the Donkey Kong data with scikit learn's MLP Classifier. The hyperparameters I tuned were the gradient descent methods (LBFGS, Stochastic Gradient Descent (SGD), and Adam), the strength of the cost function's L_2 penalty α, the learning rate, and the number of units in each layer. I performed a grid search over these hyperparameters. I declared winning settings based on model classification performance on a test set made up of the same 560 transistors' spiking activity from the Space Invaders video game (similarly pre-processed with standardization and PCA). In my initial search, it was fairly clear that SGD was the best minimization algorithm, 10^{-5} was the best learning rate for SGD, and (400, 200, 100) was the best setting for the layer sizes. The best α was somewhat less clear and would change between runs. To account for the effect of random initializations, I fixed the other hyperparameters to their best settings and trained ten times for each α in the space I explored (between 1 and 10^{-9}). Then, $\alpha = 0.001$ was the clear winner. With these settings, I achieved 76.43% accuracy on the test set (and 93.04% on the training data). Since I chose 280 transistors from outside the ALU (the same number of transistors as are in the ALU), this is a balanced accuracy score.

4.2 Clustering

Now, we move to the unsupervised setting, where the goal is to identify the clusters of transistors in the ALU and not in the ALU without the help of labels. The labels are used only to check the accuracy of the clustering assignments.

I tried three clustering methods: k-means, k-medoids, and spectral clustering. I applied these algorithms to the distributions of spiking activity for each transistor. The support for each of these distributions is $\{0, 1, \ldots, 100\}$. The mass of the distribution located at $i \in \{0, 1, \ldots, 100\}$ is the number of occurrences of i

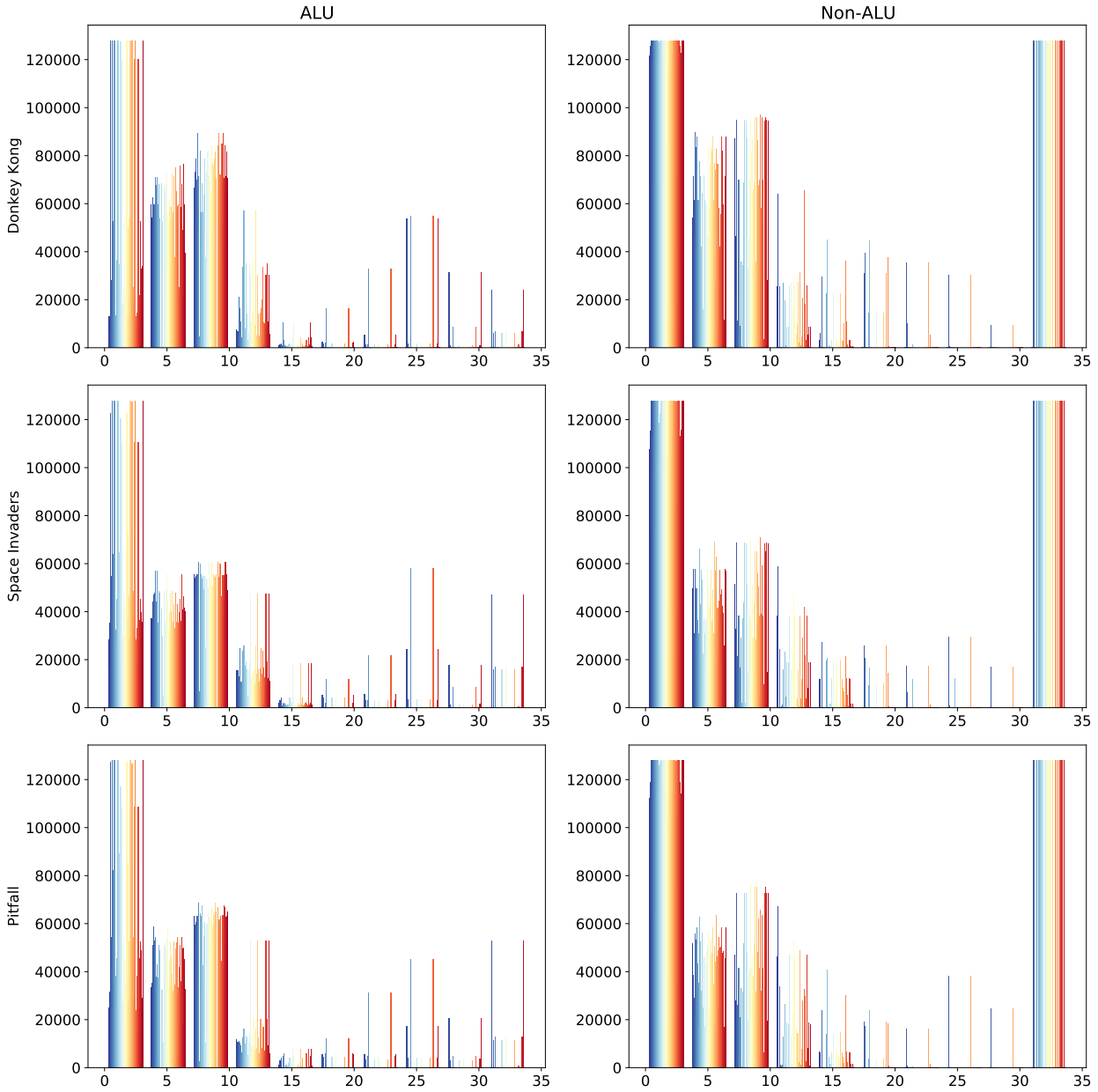

Figure 2: Histograms for each of the games of transistors state switching behavior in the 280 ALU and 280 randomly chosen non-ALU transistors. For each histogram, the 280 transistors are mapped to different colors. Note that these histograms are not normalized, unlike the distributions fed into the clustering algorithms.

over the 127,998 time steps normalized by 127,998. The distributions for the ALU and non-ALU transistors are shown in FIGURE 2 for each of the three games. We must take care in our interpretation of these objects. The distributions I constructed for each transistor and game are not estimates of a *probability* distribution. The 6502 is an entirely deterministic system. Every time we run a game on it, each transistor will behave in exactly the same fashion as it did during the preceding run of the game. There is not an ounce of randomness here. There is only one probabilistic interpretation I can think of to the distributions I have constructed. Suppose we are presented with the question, "What is the probability that a time bin

Clustering method	Data	Accuracy on ALU	Accuracy on non-ALU	Balanced accuracy
k-means	DK	81.79%	44.64%	63.21%
	SI	82.14%	33.57%	57.86%
	PF	81.79%	34.64%	58.21%
k-medoids	DK	81.79%	48.57%	65.19%
	SI	82.14%	50.71%	66.43%
	PF	76.43%	71.79%	74.11%
Spectral	DK	81.79%	41.43%	61.61%
	SI	82.14%	32.5%	57.32%
	PF	82.14%	33.57%	57.89%

Table 1: Clustering algorithm performance. Accuracy on the ALU refers to the classifier's sensitivity to the ALU, and accuracy on non-ALU is its specificity. Since I have chosen the classes to be of the same size, the balanced accuracy is also the overall accuracy. "DK" means "Donkey Kong", "SI" means "Space Invaders", and "PF" means "Pitfall."

for some transistor and some game contains i spikes?" Then, the answer given by looking at the distribution for the game and transistor and picking the mass associated with i is the best answer possible. This is not an estimate for the probability, but the probability. This is all to say that this pre-processing step amounts not to density estimation but to the construction of a perhaps useful lower-dimensional representation of the 127,988-dimensional vectors associated with each transistor for each game.

The k-means algorithm tries to minimize within-cluster variance, or equivalently, the sum of squared deviations between points in a cluster and that cluster's mean, for each cluster. k-medoids is similar to k-means, but instead of minimizing the Euclidean distance between points in a cluster and that cluster's mean, k-medoids minimizes the distance for an arbitrary distance metric between the cluster points and the cluster's medoid. The medoid is the point in the cluster with minimal average distance to the other points in the cluster. Spectral clustering is somewhat more complicated. It begins by forming a similarity matrix whose i, j'th coordinate captures the similarity between point i and j. (Thus, this matrix is symmetric.) A common choice for a measure of similarity, and the one that I have used, is the Radial Basis Function (RBF) kernel or Gaussian similarity function $s : \mathbb{R}^2 \to \mathbb{R}$, where

$$s(x_i, x_j) = e^{-||x_i - x_j||^2/2\sigma^2}.$$

Next, spectral clustering constructs a similarity graph from the similarity matrix and finds a graph Laplacian of the graph's adjacency matrix. For n data points and k clusters, it then computes the k top eigenvectors of the graph Laplacian and forms $U \in \mathbb{R}^{n \times k}$ whose columns are the eigenvectors. Finally, it applies a clustering algorithm to the rows of U to get cluster assignments for each of the n points. A common clustering algorithm choice for this final step, and the one that I have used, is k-means [4][5].

I implemented k-means with scikit learn. Since it is well-known that the k-means algorithm can become ensnared by local optima, I ran it over 1000 random initializations. I set a tolerance of 10^{-6}. I used scikit learn extra to implement k-medoids. I used the L_1 distance metric here. In fact, this was my motivation for trying k-medoids. The L_1 metric would appear to suit these distribution objects better than the L_2 metric that k-means requires. L_1 is reminiscent of the total variation distance between probability measures. I implemented the spectral clustering in scikit learn. As discussed above, I chose the RBF kernel for my similarity measure and k-means for the final-step clustering strategy.

The results are summarized numerically in Table 1 and visually in Figure 3. k-means and spectral clustering achieve roughly the same performance. Both do quite well on the ALU transistors and quite poorly on the non-ALU ones. k-medoids clearly outperforms the other two. On Pitfall it achieves a balanced accuracy of 74.11% through decent performance on both the ALU and non-ALU transistors.

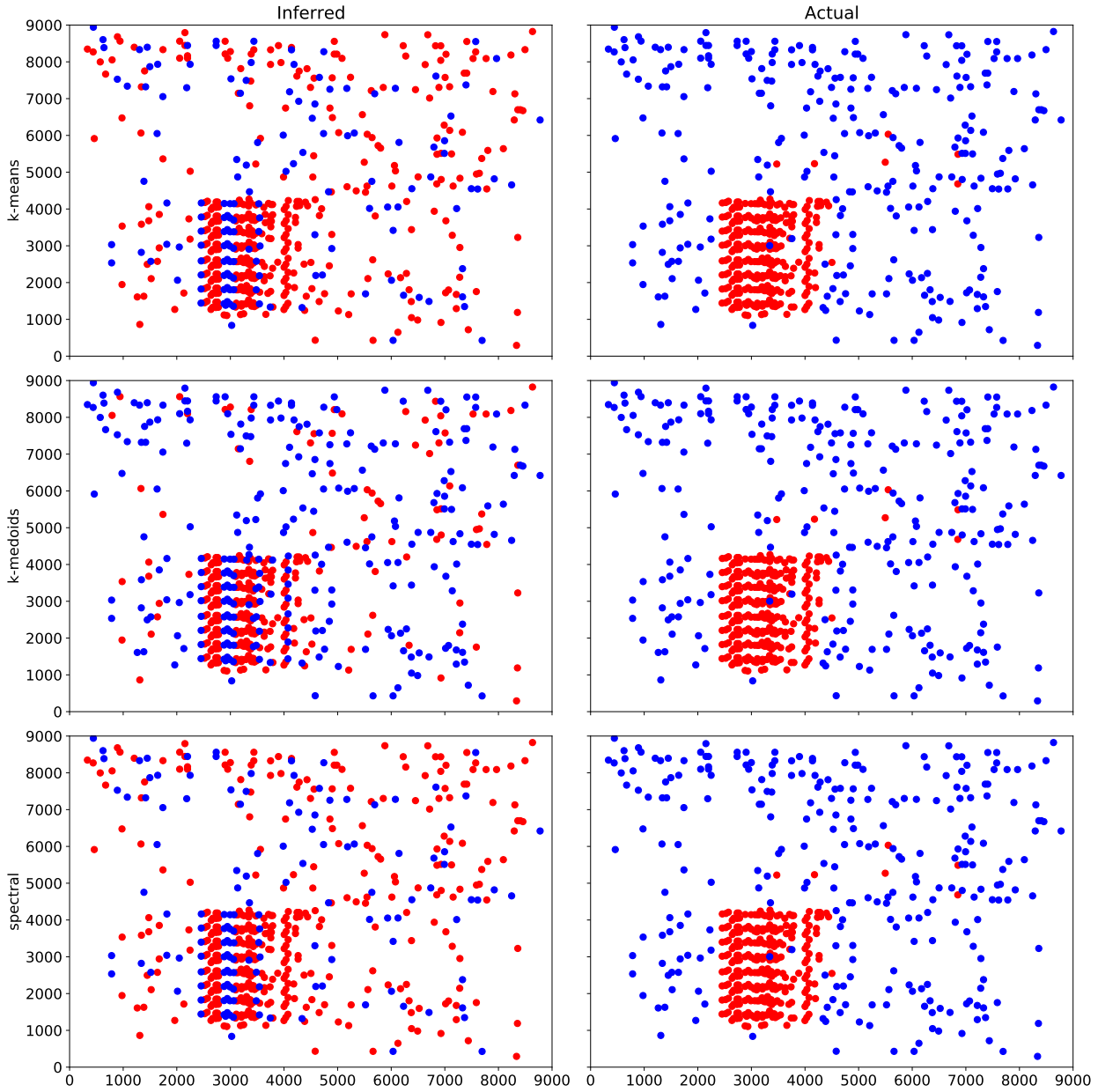

Figure 3: Inferred clusters for each of the three clustering algorithms. Transistors inferred to belong to the ALU or those actually in the ALU are plotted in red. Transistors inferred not to belong to the ALU or those actually not in the ALU are in blue. (I have permuted the inferred labels on the cluster assignments for consistency across the methods.)

5 Conclusion

My neural network analysis confirmed that the time series of transistor state switching behavior carries information regarding the functional role of the transistors. Extracting this information in the unsupervised setting proved somewhat more challenging. Here, k-medoids with an L_1 metric on the transistors' distributions returned the best performance: classification accuracy of 65.19%, 66.43%, and 74.11% on Donkey Kong, Space Invaders, and Pitfall respectively.

There are a few things I would like to try still. First, applying the clustering methods to the PCA data that I fed into my neural network would be a natural next step. Second, clustering algorithms can suffer in high dimensions, so I want to try reducing the dimensionality even further with PCA (5, 10, or 20 may be good dimension choices). Second, the over-performance of k-medoids may be related to its use of the L_1 distance metric. L_1 seems especially suitable to measuring the similarity of the distribution objects. So, I would like to run spectral clustering with a similarity matrix computed using L_1 distance rather than the RBF kernel. Third, I would like to beat the 76.43% accuracy in the supervised setting. I suspect that performance here is a rough upper bound on what we can accomplish without labels, and if the best we can do is on the order of 3/4 classification accuracy, this suggests that the transistor state switching time series is deficient important information. If this is true, we may need to combine it in some way with the net-list and perform an analysis on these joint data.

References

[1] L .Paninski and Jp Cunningham. "Neural data science: accelerating the experiment-analysis-theory cycle in large-scale neuroscience". en. In: *Current Opinion in Neurobiology* 50 (June 2018), pp. 232–241. ISSN: 09594388. DOI: 10.1016/j.conb.2018.04.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0959438817302428.

[2] Greg James, Barry Silverman, and Brian Silverman. *Visualizing a Classic CPU in Action: The 6502*. July 2010. URL: http://visual6502.org/docs/6502_in_action_14_web.pdf.

[3] Eric Jonas and Konrad Paul Kording. "Could a Neuroscientist Understand a Microprocessor?" en. In: *PLOS Computational Biology* 13.1 (Jan. 2017). Ed. by Jörn Diedrichsen, e1005268. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005268. URL: https://dx.plos.org/10.1371/journal.pcbi.1005268.

[4] Ulrike von Luxburg. "A Tutorial on Spectral Clustering". In: *arXiv:0711.0189 [cs]* (Nov. 2007). arXiv: 0711.0189. URL: http://arxiv.org/abs/0711.0189.

[5] Andrew Ng, Michael Jordan, and Yair Weiss. "On spectral clustering: analysis and an algorithm". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic* (Jan. 2001).