Project 34: Reverse Engineering the MOS 6502 Microprocessor

Camille Taltas
ct2522@nyu.edu

Francesca Guiso
fg746@nyu.edu

Tinatin Nikvashvili
tn709@nyu.edu

Abstract

Understanding the relationship between structure and function in the brain is a difficult task in neuroscience, which has been tackled using machine learning techniques. Despite the abundance of human intracranial recordings data, we still have little understanding of how our brains work. This project seeks to lay the ground-work for using a neural network, a predictive model based on our current understanding of how the brain learns – trained on a microprocessor chip called the MOS 6502 – in order to infer some structure of the brain.

1 Introduction

This project aims to understand the inner structure of the MOS 6502, a microprocessor chip, with the goal of developing a model that can ultimately be applied to human intracranial recordings to further our understanding of the brain.

Microprocessors are computation units that execute computations given an input state and an instruction (for example, adding two numbers together), producing an output. Today, many simulators of these chips exist which have been written in a rule-based way using our current knowledge of their structures and inner workings. Our goal is to use such a simulator, more specifically that of an MOS 6502, in order to train a neural network, a model well-suited to model complex systems in a non-rule based manner, to emulate the data. Our model uses the data provided by a simulator and predicts the microprocessor's next state given the previous state.

Our next task is to make sure that this black box model was in fact able to predict the output correctly by learning the actual structure of the chip. Doing so requires us to evaluate our model using the simulator and feeding it different predicted outputs in order to see how invariant or variant it is to shifting inputs. Proving this to be true would mean that we can then apply this model to human intracranial recordings and guarantee its ability to make predictions based on an actual understanding of the inner workings of the brain.

2 Motivation and Previous Work

2.1 Motivation

In neuroscience, we don't understand how the structure of the brain relates to its function, because we lack ground truth data for the brain's inner states. The microprocessor aids us insofar as it provides ground truth data of the system that we can test our understanding against. Specifically, the MOS 6502 has multiple advantages: it is a complex but deterministic function and we have full understanding of the inner workings and the architecture of the chip. Lastly, the 6502 can be completely simulated at the individual transistor level, so this gives us access to a large amount of training data.

We would like a model that is capable of inferring structure just from a predictive function. If we can do so for the 6502 we can apply it to human intracranial recordings hoping to gain a better

Preprint. Under review.

understanding of how the brain works. The way we will tackle this problem is first train a neural network on the traces provided by the simulator, and then evaluate how well the model is doing by comparing the next trace of the simulator with the next trace that the emulator we are building gives us.

Neural nets are not easily interpretable so we will look at how our model performs when we replace the core logic of the simulator with the outputs of our neural network.

2.2 Previous work

The link between structure and function in the brain is an open question in modern neuroscience [1]. In a recent paper [3] seek to apply neuroscience methodologies to learn to reconstruct the structure of the MOS 6502, suggesting that these tools are unable to do so. However, in recent work it has been shown that it is possible to build tools to reconstruct the dynamics of spiking neural activity [5]. In this project, we are trying to use deep learning techniques to reverse-engineer the MOS 6502 structure.

Although the inner structure of the brain is far more complex than that of a microprocessor such as the MOS 6502, the two share some important similarities. "Both systems consist of interconnections of a large number of simpler, stereotyped computing units. They operate on multiple timescales. They consist of somewhat specialized modules organized hierarchically. They can flexibly route information and retain memory over time. [3].

We chose to use the MOS 6502 microprocessor, the microprocessor used in the Apple I, the Commodore 64, and the Atari Video Game System. The 6502 is certainly not a human brain but the differences make studying the chip a lot easier than studying the brain. "The human brain has hundreds of different types of neurons and a similar diversity of proteins at each individual synapse [25], whereas our model microprocessor has only one type of transistor (which has only three terminals). The processor is deterministic while neurons exhibit various sources of randomness. With just a couple thousand transistors it is also far smaller. And, above all, in the simulation it is fully accessible to any and all experimental manipulations that we might want to do on it." [3].

3 Data

3.1 Description

There are many publicly available repositories for the MOS 6502 simulator. However, given that the underlying transistor dynamics of the MOS 6502 chip most closely mimic the neural networks in the human brain, our goal is to see whether an artificial neural network can accurately model the transistor-level dynamics of the Atari system. For this reason, we choose Eric Jonas' adaptation of the simulator written by [2], which simulates all the transistors rather than emulating the instruction and memory content. Television Interface Adaptor (TIA) were also reverse-engineered and the simulator can simulate the wire voltages and transistor states. The simulator allows us to run classic Atari games and generates roughly 1.5GB/sec of state information [3].

We collect our data by running the simulator, which executes a game of Donkey Kong using the MOS 6502. We had to change the code to save the intermediate chip states. For every half-clock, we record a snapshot of the wire states from the main processor (6507) and the television interface (TIA) as well as the ram and i/o timer of the the periphery interface (PIA).

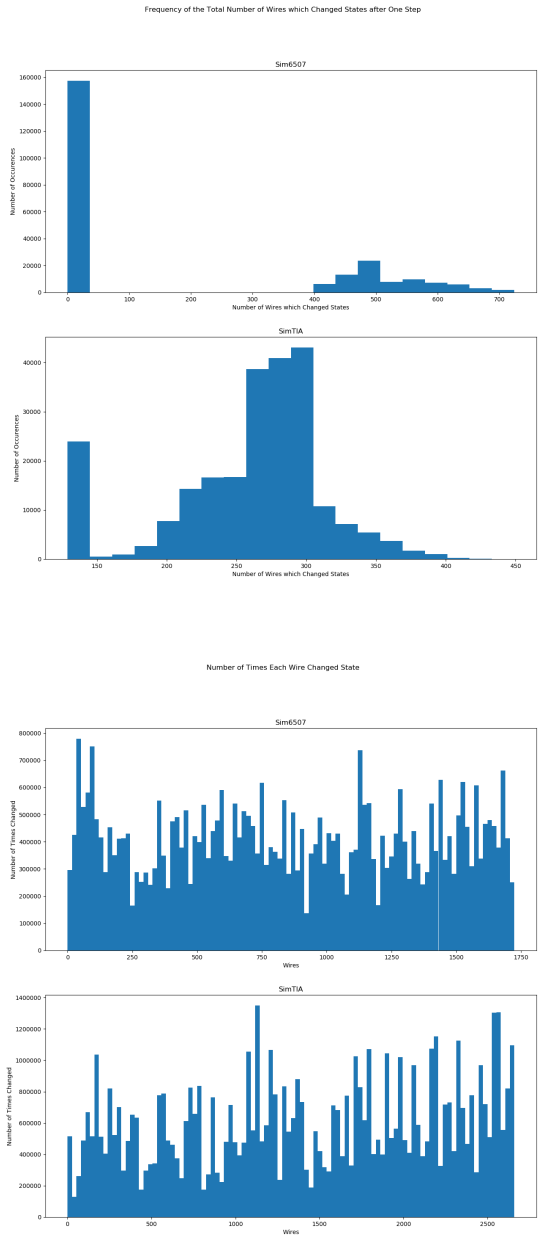
We want our model to predict the microprocessor's next state given the current state, therefore our input and output data will represent snapshots of the chips that are offset by a single time period. For each time-step t, our input x is a 1×1877 dimensional row vector, which represents the states of the RAM and i/o timer from PIA concatenated with the states of the 6507 wires, of which there are 152 and 1725, respectively. Our output y is a 1×4385 dimensional row vector, which represents the states of the 6507 wires at the successive time-step, $t + 1$, concatenated with the state of the 2660 TIA wires, at time-step t. This gives us full imitation of the underlying chips. Our data also takes positive integer values.

Each time step represents an instance in our input data. Our data set is comprised of 236,087 time steps, and, thus, our input and output data each have 236,087 rows. We further split this data into 60% for training, 20% validation, and 20% test sets.

3.2 Data Visualization

In order to better understand and evaluate our model we performed several visualizations on our labels, y.

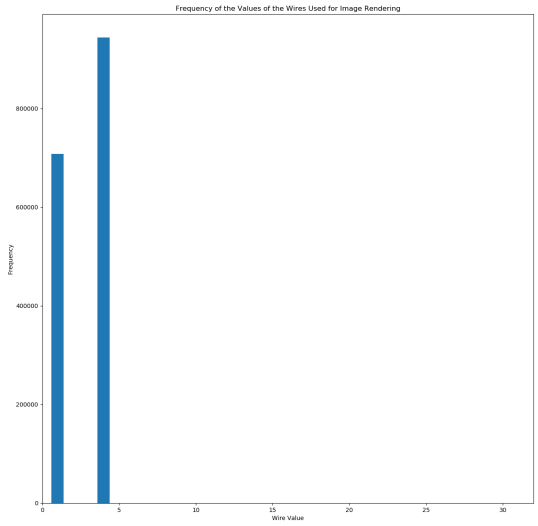
In our simulated data, not all of the wires change state between successive time steps, and those that do do not necessarily change the same number of times. As seen below, we plotted histograms for the total number of wires changed for each time step as well as the total number of changes for a given wire. These visualizations are separated for the 6507 and TIA chips.



This first set of histograms allows us to see that for the main processor changes are sparse. In most cases, more particularly in 160,000 of them, no wires change state over each time step and when they do around 500 of them change at once. On the other hand, the television interface sees more of its wires change from one time-step to another. In most cases, around 250 to 300 wires change in one time-step and there are only around 20,000 occurrences that see no change.

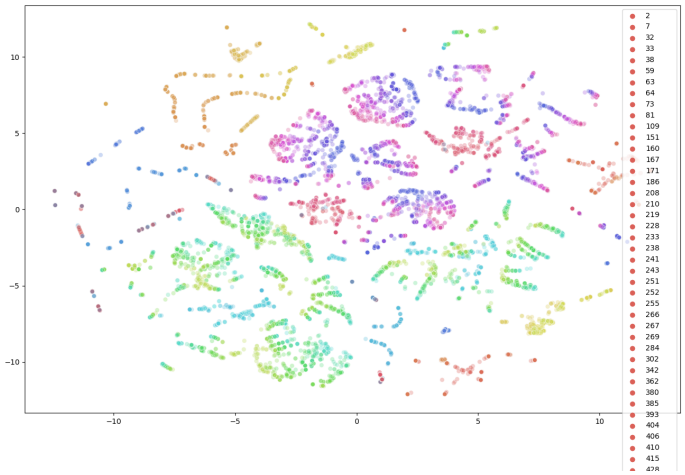
For the second set of histograms, we observe that there is not much locality in the wire state changes. We conclude that by noting that most wires which are close to each other do not change state the same amount of times. Knowing that our data can be sparse on some parts of our labels and that there is no locality in the changes tells us that predicting these changes is going to be a difficult task with a simple model, hence our use of neural networks.

Moreover, the wires that are necessary for producing the images of the game are all concentrated in seven wires which are part of the television interface data. Wires take values $0, 1, 2, 4, 8, 16, 32$ and the frequency at which these values appear can be seen below.



As you can tell pretty clearly here, we only see values 1 and 4 appear within these wires.

In order to provide more information about the distribution of our data, we perform tSNE, which is well-suited for high dimensional datasets, on a subset of size 20,000 of our Y dataframe. Given that our labels were arrays we encoded each unique array in order to be able to color both of these graphs.



Here, we can see that for this random subset of our data, our points are rather well clustered in their own subgroups which are appearing well spaced out and grouped together. Given how our datasets are structured from different parts of the microprocessor the results we get from tSNE are rather

positive in their ability to group labels together given a lower-dimensional representation of our training data.

4 Modeling

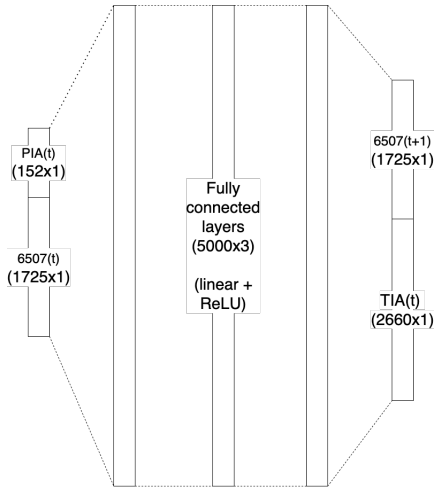
Our proposed approach to infer the structure of the processor based on these input and output states is using a simple feed-forward neural network. We use neural networks as they are well-suited to model complex relationships between inputs and outputs (such as the microprocessor, as well as the brain). We can do so without using recurrent architectures because, despite there being a temporal aspect to our data (which consists of successive states), each trace for each time-step has access to the content of the register and the instruction being executed. Therefore, our model is Markovian – it assumes the future state is independent of the past state given the current state – and so we do not require a model that explicitly accounts for these longer-term temporal dependencies.

While our brain is non-Markovian in its processing of information, this is a property of the microprocessor we are using to develop our neural network model, which albeit simplistic, can provide a basic approach to tackle the problem on which to improve further.

4.1 Algorithm

We use a simple feed-forward neural network with 3 hidden layers of dimension 5,000, and a Rectified Linear Unit (ReLU) as the activation function, as shown in Figure 4.1.

$$ReLU(x) = \max(0, x)$$



Neural Network Architecture with Input, Output, and Hidden Dimensions

For a given instance (time t), our neural network takes an input $x \in \mathbb{R}^{1\times 1877\times}$ and produces an output $\hat{y} \in \mathbb{R}^{1877\times 1}$, mapping $z \in \mathbb{R}^{1877\times 1} \rightarrow \mathbb{R}^{4385\times 1}$ via a sequence of linear and non-linear transformations. We then "discretize" the parts of this real-valued output that correspond to discrete wire states by replacing y_i with the closest possible discrete state ($\arg\min_{s\in 0,1,2,4,8,16,32} y_i - s$). We then train this network using Mean-Squared Error (MSE) as our loss function. We use Adam [4] as our optimizer with a learning rate of 0.001.

$$MSE(y, \hat{y}) = ||y - \hat{y}||_2^2$$

4.2 Implementation

We implemented our model in PyTorch and performed the training, validation, and testing of our models on NYU's Prince Cluster using NVIDIA V100 GPU.

Figure 1: Training and Validation Losses (MSE).

Table 1: Model Performance

Model Performance		
Dataset Size	Validation MSE Loss	Test MSE Loss
Full	24.11	23.96

4.3 Performance

We trained our model for 25 epochs using a batch size of 64. After training is complete, we use the weights that obtained the lowest loss on the validation set to test our model performance on the test set. The results are summarized in Table 1 and 1.

We can improve our modeling approach by conducting more experiments on different configurations of our model architecture, such as increasing the number of layers, their dimensionality, as well as conducting more hyper-parameter tuning and training for more epochs. We could have also tried to use classification methods and corresponding loss functions rather than a MSE loss combined with the prediction discretization.

5 Evaluating our Model: Iterative Loop

For the purpose of this project we are not only concerned with achieving a very low validation loss but also with allowing the model to learn the right representation of the structure of the chip. We can have a model that achieves a very low validation loss without actually learning the underlying structure of the chip, so we need to come up with other ways to evaluate whether our model is inferring structure.

One hypothesis is that there is a tradeoff between the complexity of the model and its ability to emulate the chip correctly and a more simple model capable of learning simpler rules governing the chip's behavior but perform more poorly.

The chip functions in a rule based way, where one state determines the next. Our neural network is meant to emulate, by understanding the structure, this process and accurately predict the next state given the current state. In this section we observe how the predictions evolve over n successive states.

5.1 Implementation

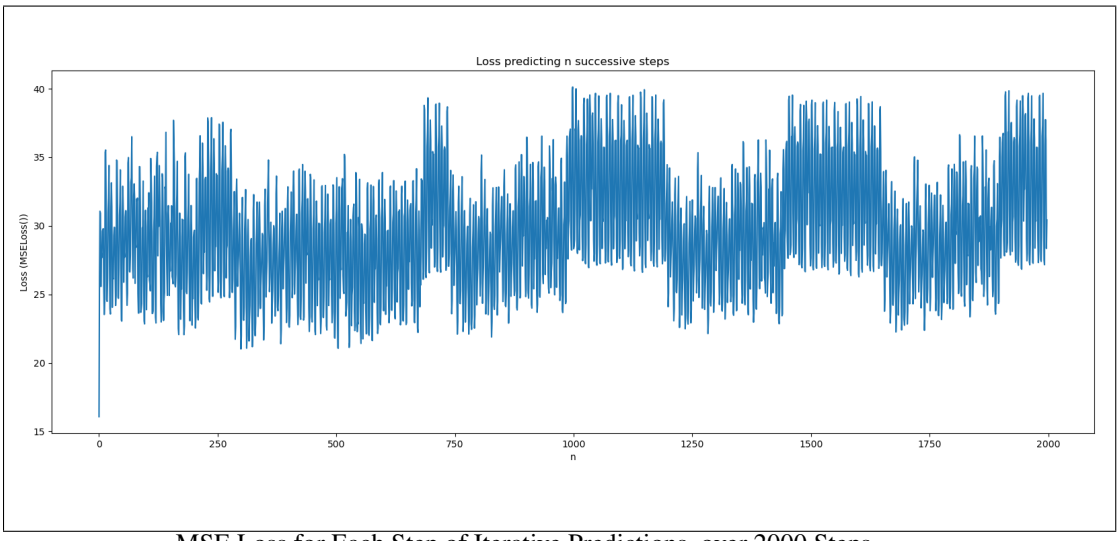
Starting from the first time instance in our data, we make a prediction for the next state of the main processor chip. We then let our model predict the subsequent time step by feeding as input the previous prediction, as opposed to the next state from the simulated data. We repeat this process $n = 2000$ times. We are making predictions in an iterative manner in the sense that, for each time

step t, we obtain a prediction of the main processor's wire states that the model has arrived to based solely on its own predictions for the previous $t-1$ such states.

We record the loss between the ground truth chip state $y = [y_{6507}, y_{TIA}]$ and our predictions $\hat{y} = [\hat{y_{6507}}, y_{TIA}]$ at each time-period. Note that our input and output data differ in that the input is a concatenation of the PIA chip states and the main processor states, while the output is a concatenation of the main processor chip states and the TIA chip states. Therefore, for the purpose of predicting successive states, we focus only on the part of the input that represents the main processor (6507), so that we can use those predictions as the corresponding portion of the input to the next state. Therefore, we do not make iterative predictions of the PIA and TIA, but rather treat these parts of the data as given, updating them for each time-step based on the simulated data.

The results are shown in Figure 5.2.

5.2 Performance



MSE Loss for Each Step of Iterative Predictions, over 2000 Steps.

Given that the iterative predictions of the main processor states are based solely on the model's previous predictions, the fact that the losses are not increasing tremendously even when the process is repeated over 2000 steps, indicates that the model is able to reasonably predict a sequence of successive states for each wire and, thus, indicates that it might understand some underlying structure.

5.3 Improvements

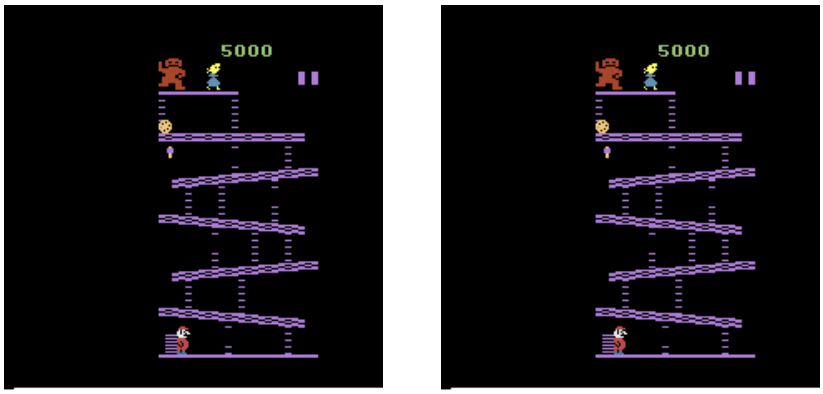
We find two main areas of improvement for the iterative predictions: 1) extending the number of time-steps beyond 2000 to model longer chip computations would allow us to have more evidence that our model is actually understanding the structure of the chip, and 2) modifying the loss criterion to consider only the changes in the main processor states, rather than comparing the entire target with the entire output of our model (which include the TIA chip states) would allow us to identify more accurately if this iterative prediction is actually performing well.

6 Evaluating our Model: Rendering the Images

6.1 Implementation

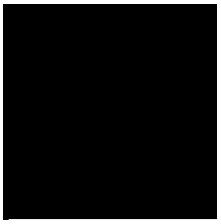
Besides saving chip states at every half clock, we also collect image renderings of DonkeyKong as the simulation runs (see left image below). Yet another way to evaluate whether the neural network is making predictions based on the structure of the chip is to replace the actual circuit simulator with the trained neural network and see if we can get the first correct frame of the game using our neural network's predictions of the TIA states. Pixel luminance and colors of these images are determined by

the wire states of the 7 TIA wires. First, we tested whether it was possible to replicate the rendering of the images using output from the TIA chips overstepping the simulation. Using the chip states from TIA saved at every half clock we were able to replicate image renderings perfectly. The figure below shows the images produced by the simulator (left) and the image we replicated by using the wire states from the 7 TIA chips (right). Now we want to replace the core logic of the simulator with our neural network's predictions for multiple steps to see how well it is working or if it is working at all. As mentioned in Section 5, the validation and test losses do not tell us the full picture in this case. We don't want to know only whether the neural network can make a good prediction at each time step, but also if we replace what is being updated by the chip with the trained neural network, if we can get to the initial screen rendering of DonkeyKong. Since our neural network's prediction is not perfect at each time step, the error will accumulate over time, and getting this screen rendering of the game might not be possible.



6.2 Performance

Figure below shows the image rendering of the first screen of DonkeyKong using neural network's predictions of the TIA wire states. As we originally expected, we weren't able to render the actual screen of the DonkeyKong given that errors accumulate over multiple steps. This indicates that our model needs to be improved until we can get the image rendering that looks like a DonkeyKong frame.



6.3 Improvements

We need to improve our modeling by trying to use the current architecture but changing the loss, do more hyper-parameter tuning, train for longer until we achieve close to 0 training loss i.e overfit to our data. We can also try to change our label to only include 7 TIA wire states that determine the luminance and color of the images. If we can't overfit on the training data, we can change the architecture. Ultimately, the success would be being able to replicate the first image rendering of DonkeyKong.

7 Next Steps

One of the most important questions for our project was to figure out what was the right way to evaluate performance of the neural network. We implemented the iterative predictions and looked at whether we could replicate the screen rendering of DonkeyKong. We could have implemented other evaluation techniques, such as different intervention-based explainability. For example, it would be useful to feed different inputs to our model and see to what extent this intervention is invariant, within our predictions, to these inputs. More specifically, identifying the content of the registers that are not supposed to matter for executing specific instructions and then evaluating whether our predictions vary or not accordingly would give us an idea of whether or not our model understands the structure of the chip when making predictions. Lastly, our final next step would be to take our methodology and apply it to human Electrocorticography (ECoG) or micro electrode array data. If we manage to train a neural network that indeed makes predictions based on the structure of the chip we will be able to test it on intracranial recordings from the human brain and infer its structure in a similar manner.

8 Lessons learned

The main struggles we had with this project were the steep learning curve in understanding the simulator for the MOS 6502 and the large amounts of data we were collecting. In terms of the simulator, this project required a deep understanding of how it worked in order to debug, access the correct data, modify it for our predictions etc. Before starting this project, we assumed that we would only be running the simulator once to get the data and that this project would not require an actual understanding of a Github repository written five years ago by someone outside of our department, which left us with few resources to understand this simulator. Had we known this beforehand, we would have either insisted more upon our advisors to help us understand the simulator with their experience in computer science, contacted the repository author himself, or asked for the help of another student who has experience digging through such extensive and complicated code. The other challenge we faced was the large amount of data we were collecting. We were not expecting to face such space and time constraints and, thus, did not prepare for it accordingly. Had we known the size of our dataset would be a problem we would have probably implemented big data techniques rather than spent time solving memory issues or waiting for our code to run. In retrospect, it probably would have been better to design a database to efficiently store and retrieve the simulator data. The fact that this project explored such uncharted territory, which was the most exciting aspect of the project, also turned out to be one the thing that made it most challenging, in terms of facing problems for which there were no clear resources to consult.

9 Conclusion

In conclusion, reverse-engineering the MOS 6502 was an ambitious goal, which proved to be just as difficult as expected. We managed to train a neural network which produced good results in terms of MSE loss but did not seem promising in its ability to infer the structure of the chip. However, these results are too premature to rule out the feasibility of this task in general. There is a lot more work that can be done to improve our model to better our predictions, and use other techniques that can be implemented to evaluate this model and provide a positive feedback loop on how to better infer the structure of the chip.

10 Student Contributions

Camille Taltas- Data visualization and data collection.

Tinatin Nikvashvili- Rendering of the images and data collection.

Francesca Guiso- Modeling and iterative predictions.

We all participated in writing the report and discussing how to best organize this project.

References

[1] J. J. Hopfield and C. D. Brody. What is a moment? transient synchrony as a collective mechanism for spatiotemporal integration. *Proceedings of the National Academy of Sciences*, 98(3):1282–1287, 2001. ISSN 0027-8424. doi: 10.1073/pnas.98.3.1282. URL https://www.pnas.org/content/98/3/1282.

[2] G. James and E. S. M. S. Barry Silverman, Brian Silverman. Visual6502.org simulation of an atari 2600, 2014.

[3] E. Jonas and K. Kording. Could a neuroscientist understand a microprocessor? *bioRxiv*, 2016. doi: 10.1101/055624. URL https://www.biorxiv.org/content/early/2016/05/26/055624.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[5] C. Pandarinath, D. J. O'Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg, et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 15(10):805–815, 2018.