Recurrent Neural Network for brain-machine interfaces: decoding neural data for communication prosthesis

Maxime Cauchois
Department of Statistics
Stanford University
maxcauch@stanford.edu

Julien Boussard
Department of Statistics
Stanford University
julienb@stanford.edu

Theodor Misiakiewicz
Department of Statistics
Stanford University
misiakie@stanford.edu

Abstract

Brain-machine interfaces attempt to translate neural activity into control signals and create a direct pathway between the brain and an external device. In particular, BMIs give hope to impaired patients for communicating via an external computer. The main challenge lies in reliably decoding neural activity, a highly non-linear and noisy time series data, into a particular stimulus. In this project, we build a recurrent neural network to decode covert mental processes of non-human primates from intracortical electrodes. The animals were trained to reach a target on a screen among 48 predefined positions. Our goal is to infer which target the animal plan on reaching from its neural activity associated with motion planning. We used data provided by the Stanford Neural Prosthetic System Lab and constructed a RNN model with one LSTM that shows promising performances.

Github repository: https://github.com/misiakie/CS230-RNN-for-BMIs.git

1 Introduction

Brain-machine interfaces (BMIs) translate neural activity into control signals for external devices, such as robotic limbs or computer cursors. We refer to [2] for more details about BMIs. They promise to aid paralyzed patients, replace missing limbs by brain-controlled prosthetics [1]. Despite recent progress and spectacular proof-of-concept, their performance is partly limited by the ability of the decoder to reliably map neural activity to specific stimulus (e.g. a specific functional reaction in an organ or tissue, for example opening your right hand or focusing on a particular target).

Neurons represent and transit information by firing sequences of spikes in a complex network of dependency. The neural activity is represented by a time-series of per-neuron spike sequences. They reflect both intrinsic neural dynamics and temporal characteristics of stimulus, and display complex nonlinear relationships as well as temporal dependencies. Furthermore, neural responses vary from trial-to-trial even when the same stimulus is presented repeatedly (due to uncontrolled cognitive variables such as attention, biophysical noise, etc.). The goal of the decoder is to reconstruct the stimulus, or certain aspects of that stimulus, from this complex spike sequence that prompted it [3].

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

In this project, we explore the possibility to use a recurrent neural network (RNN) as a decoder. RNNs have demonstrated their power to model nonlinear relationships and complex temporal dependencies in time series. However their application to BMIs have been so far limited (see [4] for a discussion on RNNs). We will work on the dataset provided by the Stanford Neural Prosthetic Systems Lab, directed by Prof. Krishna Shenoy. They recorded neural data from non-human primates that were trained to move a cursor on a screen to reach a target. At each experiment, one of the 48 predefined target turns on and the animal reaches it with his arm. Some trials involve a delay period when the target is lit up but the monkey has to wait before reaching it. This period is also called the "planning phase", because the animal plans on reaching the target and preprocess his arm movement (covert mental process, associated with large neural populations).

Our goal is to use the neural activity during the "planning phase" to predict the target the animal wants to reach. Following the work of [4] and [5], we explore the ability of RNN to effectively recover the target, or the approximate target class from this complex signal. Such a decoder would be very useful in Communication Prosthesis as the targets are representative of a keyboard, on which a patient would type words by "planning" to reach each letters.

2 Dataset and Features

The data were recorded by the Stanford Neural Prosthetic Systems Lab, directed by Prof. Krishna Shenoy, between 2010 and 2014. We refer the reader to [4] for further information on the experimental setup.

2.1 Description of the data

A group of non-human primates were trained to acquire targets with a cursor on a screen. There are 48 targets, dispatched on 3 circles of diameters 8, 10 and 12 cm. Each trial consists in one delay period, when the taget turns on and the primate plans on acquire the target, the reaching phase where the primate reaches the target with the cursor, and the final period where the primate's hand goes back to its initial position. During the delay period, the primate is instructed not to reach the target and waits for a signal (such as the target changing color or shape for example) before reaching the target.

The data was recorded with reaches to the 48 different targets, disposed on 3 circles of 16 targets each. It is important to use enough targets as a typical communication prosthesis is composed of approximately 30 targets (the alphabet plus the special characters).

The neural data was recorded by two 96-electrode arrays implanted in two zones of the primate's motor and pre-motor cortex. They record two patterns of 96 neurons.

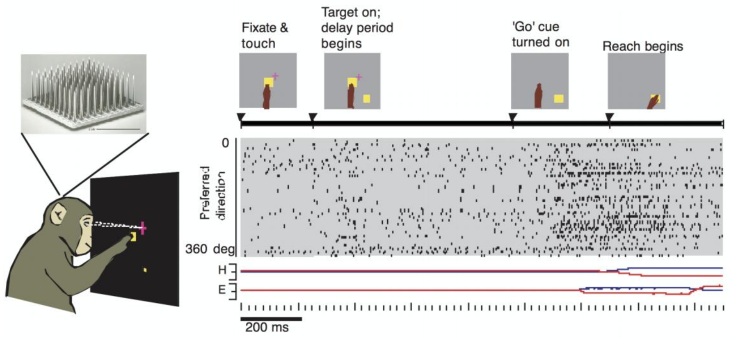

Figure 1: Encoding of the data. A target lights up and the primate reaches the target while the two 96-electrode arrays record the spike pattern.

2.2 Data processing

The data were Matlab structure arrays that we converted in tf.record. The code can be found on our github repository.

The principal components of the data are "spikeRaster" and "spikeRaster2" that represent the neural activity of the two patterns of neurons over time, and "targetPos", that represents the position of the target associated with the trial. We converted "targetPos" to $1 : 48$ as it is easier to represent the class of each target. The trials are not all successful, meaning that sometimes the primate doesn't reach the right target in time. The "isSuccessful" feature allowed us not to use the few unsuccessful trials, as they are not representative of the target.

The "delayTime" feature indicates the length of the delay period (or planning phase). It is important for us as we only take the delay period into account to predict the target. Taking a delay time period that is too short will result in very bad accuracy, while taking a too long period will be bad for BMI applications (the goal is to predict the right target as fast as possible and the delay time is a lower bound for the BMI response). Furthermore, the current dataset do not provide enough successful trials for longer period. Henceforth, for this project, we decided to take a delay time of 300 ms, which allowed us to use 7361 trials. In a future work, when more data will be available, we will compare the accuracy vs. the delay time.

The spikeRaster matrices are very sparse (few spikes). We decided to bin the data in intervals of length 10, 20 or 30 ms, to reduce the dimension of the input. We tried and cross-validated the models on both initial and binned data.

2.3 Data Visualization

To visualize the data, and following the work of [6], we performed a PCA on neural data and projected the data on the two first Principal Components. The centroids of the data corresponding to each targets are displayed.

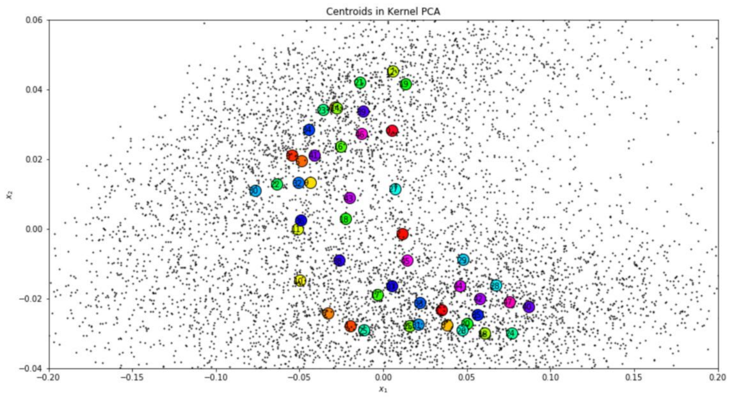

Figure 2: Visualization of the two principal components of the data. Each centroid corresponds to one target

As it is explained in [6], we see that the centroids that are close on this figure correspond to targets that are close in space. It shows that the neural data indeed encode the target classes.

3 Methods

As our data is composed of time-series, it was natural to use a Recurrent Neural Network to fit the function linking neural data to the targets. We tried different neural network architecture, composed by one or two LSTM of 32, 64, or 128 units followed by a Dense layer with as many units as targets class. They were optimized with an Adam optimizer. Our loss function was a cross entropy loss. We used Tensorflow to code our models. You can find our code on Github (follow link at the beginning of the report).

LSTMs, which stands for Long Short-Term Memory, is a popular unit used as a building block for RNN. They are designed to solve the vanishing gradient problem, which typically prevent long range dependencies in the RNN model. To do so, in addition to the standard weights, they include variable gates of range in 0-1 which block or pass on information. These variables are optimized during the training process and allow to learn order dependence in sequence prediction problems.

Our models were overfitting the training dataset. To avoid it, we used Dropout regularization, and L2 regularization. We cross-validated the regularization parameters on the dev set.

We also tried a feed-forward vanilla Neural Network to compare to the RNN and it gave, as expected, lower results. It is not adapted to our problem.

To get better results, and as the spike raster matrices are very sparse, we ran similar models on binned data (see Section 3.2). We were hoping to reduce overfitting with the binned data.

We also ran our models on data corresponding to 24 targets, as it is more representative of the communication prosthesis (around 30 targets). To choose the 24 targets in the best possible way, we performed a cross-validation : we calculated the performance of the RNN on different sets of 24 targets to find the ones that maximize the accuracy.

3.1 Final Model

Our final models are composed of one LSTM layer, with 64 hidden units for 48 targets and 32 for 24, followed by a Dense layer. Our Dropout probability is 0.6 and L2 regularization parameter is 0.01 for 24 target and 0.005 for 48 target.

4 Results

Our main goal, defined with Saurabh Vyas, was to **compare the performance of a RNN to the performances of a Kernel SVM**. If a Kernel-SVM is not commonly used for decoding, it is a technique widely used in Neuro-Engineering for different theoretical applications and is considered to separate the data well. We coded such an SVM, and tuned its cost and γ parameters with Cross-Validation to estimate the best results of such a model.

We compared the SVM and RNN on the dataset corresponding to 24 cross-validated targets and the 48 targets. Of course, we found the 24 targets that maximized the RNN accuracy and not the SVM accuracy, and it might modify the comparisons. However, the 24 targets approximately correspond to the targets that maximize in-between distance and it makes sense that SVM accuracy on this set of targets is close to its maximum accuracy for 24 targets.

Also, using the binned neural data didn't help and resulted in a slight decrease of performance.

The following table list the **accuracy** for the RNN and SVM:

Number targets	SVM	RNN	RNN top 3	RNN top 5
24	25%	54%	86%	95%
48	15%	36%	67%	85%

To get more insight in the results, we calculated the top 3 and top 5 accuracies (is the target in the top 3 or top 5 predictions?). From the PCA visualization and the top 3 and top 5 results, we believed that the RNN was often predicting a "neighbor" target, a target positioned close to the one we were trying to predict. To make sure of our assumption, we divided the set of targets into 4 quadrants (4 groups of targets "up-left", "down-left", "down-right" and "up-right"), 3 circles, and 16 angles (there were 3 circles of 16 targets in total). We got 93% accuracy on the quadrants, 58% on the angles and 63% on the circles. It confirms our beliefs and our interpretation of the top 3 and top 5 predictions.

5 Discussion

Our **RNN widely outperformed the Kernel SVM**. It shows that it is effectively a powerful tool for decoding neural time-series. Our analysis with top 3 and top 5 prediction are also very encouraging. We believe that we could achieve much better performance with more data, especially with 24 targets adequately positioned in space.

However, the performance is still low, making the model unusable as such for a communication prosthesis. Our models suffered a lot from **overfitting** the training data. Although we used a lot of regularization and used binned data, train error was still very low compared to test error. **We strongly believe collecting more data would be very useful** for generalizing our function. For the same reason, we didn't tried deeper networks. Our RNN was already able to fit the training data perfectly without regularization. With much more data, it might be interesting to **try deeper networks** as it might be necessary to understand complex properties of the data.

With more data, we could also **estimate the Bayes error of the problem**. The data and the movement planning process are complex and we cannot estimate what accuracy is theoretically achievable with this particular data. It would be very useful to know how far from Bayes error we are.

6 Future Work

6.1 Preprocessing the Data

Binning the data in intervals of 10, 20, and 30 ms didn't help us get a better accuracy. However, there are other ways to preprocess the data that we would have liked to try. For example, smoothing the data with a Gaussian kernel convolution could be useful to reduce overfitting. Using a Fourier transform on the data could also be efficient to represent the spikes.

6.2 Different Time Delay

With more data, we would have also liked to compare the results obtained with different delay times. Our limitation here is that we didn't have enough data with long delay times (for example more than 400ms). It was then impossible to train and fit a RNN on it. It would have been very interesting and we could have produced a plot of accuracy vs. delay time and calculated the decoder's bitrate. Bitrate is a measure of performance widely used in BMI decoders as it measures the trade-off between the speed of the response and the accuracy.

6.3 Combination of our Model and Novel Approaches

A novel approach for decoding neural data is the **detection of an "error signal"**. New techniques based on neural feedback are able to understand when the target decoded is not right. Combining such a technique with our algorithm could help us improve the accuracy, especially as top 3 prediction is 86% for 24 targets.

7 Contributions

We all contributed equally to all parts.

8 Acknowledgement

We would like to thank Stanford Neural Prosthetic System Lab for sharing their data, Saurabh Vyas and Blue Sheffer for their help during the project. We also thank Kian Katanforoosh, Andrew Ng, and the whole CS230 team for their help and the amazing class.

References

[1] P. Brunner, L. Bianchi, C. Guger, F. Cincotti, and G. Schalk. Current trends in hardware and software for brain–computer interfaces (bcis). *Journal of Neural Engineering*, 8(2):025001, 2011.

[2] P. Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[3] E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science*. McGraw-Hill Medical, 4th edition, July 2000.

[4] D. Sussillo, P. Nuyujukian, J. M. Fan, J. C. Kao, S. D. Stavisky, S. Ryu, and K. Shenoy. A recurrent neural network for closed-loop intracortical brain-machine interface decoders. *Journal of Neural Engineering*, 9(2):026027, 2012.

[5] D. Sussillo, S. D. Stavisky, J. C. Kao, S. I. Ryu, and K. V. Shenoy. Making brain–machine interfaces robust to future neural variability. *Nature Communications*, 7:13749 EP –, 12 2016.

[6] S. Vyas, N. Even-Chen, S. D. Stavisky, S. I. Ryu, P. Nuyujukian, and K. V. Shenoy. Neural population dynamics underlying motor learning transfer. *Neuron (2018) https://doi.org/10.1016/j.neuron.2018.01.040*.