Neurális tüskék válogatása mély tanuláson alapuló megközelítések alkalmazásával

Összefoglaló:

A neurális aktivitás tüskéinek detektálása és szétválogatása kulcsfontosságú szerepet játszik az alapvető idegtudományban, de az alkalmazott területeken is, például az agy-számítógép interfészek fejlesztésében. Az évek során számos felügyelt és felügyelet nélküli megoldás jelent meg, különböző összetettségű módszerek alkalmazásával: az egyszerű küszöbalapú módszerek alkalmazásától a mély neurális hálózatokig. Ebben a tanulmányban adaptív küszöbalapú algoritmust alkalmaztunk a tüskék detektálására. Öt különböző hullámformából álló, címkézett, mesterségesen előállított tüske-adatok felhasználásával kétféle módon alkalmaztuk a mély tanulást a tüskeválogatás problémájának megoldására. Az első megközelítés a kérdést osztályozási problémának tekintette, és a lehető legkevesebb tanítási adatot felhasználó konvolúciós neurális hálót kereste. A második megközelítés abból állt, hogy a nyers adatokat további zajokkal terheltük a valós felvételek utánzására, majd egy AutoEncoderrel végeztük el a jelek zajtalanítását. Ezúttal a spike szétválogatás nem felügyelt problémának tekintettük, és spike hullámalak adatait az első tíz fő komponens felhasználásával transzformáltuk, majd az eredményeket X-means klaszterezési eljárással válogattuk szét. A konvolúciós neurális hálózat 98,9% -os osztályozási pontosságot ért el a tesz halmazon, és csak a teljes adat 5% -át használta tanításra. Az X-means klaszterezés 71% -os pontosságot ért el a zajos adatkészleten.

Neural spike sorting using deep learning-based approaches

János Szalma
Budapest University of Technology and Economics
Budapest, Hungary
sz.janni@gmail.com

Tamás Nagy
Budapest University of Technology and Economics
Budapest, Hungary
nagytam606@gmail.com

***Abstract*—The detection and sorting of neural spike activity has a key role in basic neuroscience, but also in applied fields, like in the development of brain–computer interfaces. Throughout the years many supervised and unsupervised solutions were posed, applying methods of varying complexity: from using simple threshold-based methods to deep neural networks. In this study we utilized an adaptive threshold-based algorithm to detect spikes. Using labeled, artificially generated spike data consisting of five different waveforms we applied deep learning in two different ways to solve the spike sorting problem. The first approach considered the issue as a classification problem and used a Convolutional Neural Network with as little training set as possible. The second approach consisted of applying additional noise to the waveform data to mimic real recordings and then an AutoEncoder to denoise the signals. This time the spike sorting was considered an unsupervised problem and was solved by transforming the waveform data using the first ten principal components and then feeding the results into an x-means clustering algorithm. The Convolutional Neural Network achieved a 98.9% classification accuracy on the test with only using 5% of the total data as a training set. The x-means clustering achieved a 71% classification on the noisy dataset.**

Keywords—spike sorting, deep learning, cnn, autoencoder, x-means

I. Introduction

The detection and sorting of single-unit activities in intracortically recorded electric signals have a key role in basic neuroscience, but also in applied fields, like in the development of high-accuracy brain–computer interfaces. Throughout the years many solutions were posed, approaching these issues from widely different angles. These methods range from standard algorithmic solutions with hand-engineered features to end-to-end deep learning-based approaches. While the detection of spike data is considered an easier task as it is often solved using threshold-based approaches, spike sorting is largely accomplished using supervised or unsupervised machine-learning based approaches.

The principal objective of spike sorting is to determine short time frames in a noisy time series, which contain waveforms emitted by biological neurons. Depending on the environmental conditions and the number of monitored neurons, this task can pose many challenges. If multiple neurons are scattered across a given small volume, and some of them fire with similar waveforms, more than one electrode is needed to disambiguate the original source. For this reason stereotrodes or tetrodes are used in practice. The sampling rate should also be carefully chosen. The internal depolarization, which is the steepest voltage change, lasts about 200 μs, with this, the extracellular voltage is capacitively coupled, thus it causes the voltage to rise and fall, this fact suggests a 5 kHz frequency, but it was empirically shown that spikes have important frequency components up to 8 kHz. Thus, based on Nyquist's sampling theorem, 16 kHz is suggested. In practice 20-30 kHz is used. To remove the high frequency noise, it is conventional to use a low pass filter, with a cut-off around the half of the sampling frequency. Some baseline noise is also present in the lower frequency ranges, thus a high-pass filter, with the cut-off frequency of 1 kHz is also advised. The size of the data window should be chosen with respect to the expected length of the spike. It is evident that it should be long enough to fit the span of the spike waveform, but if possible, it should not contain parts of other spikes. Thus, in general, 1 to 1.2 ms long frames are used. [1] [2]

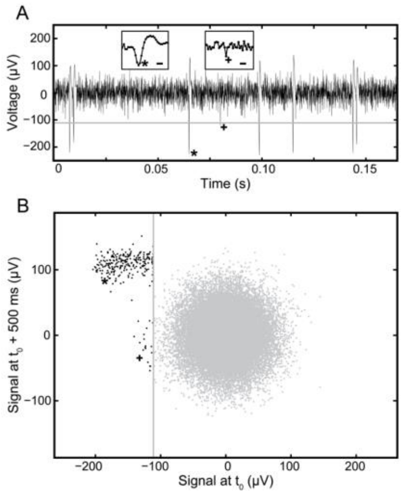

Fig. 1. Spike detection with thresholding. A - Raw waveform. The threshold is just below -100 μV (grey line). The real spikes are marked

with an asterisk (*), the false positives are marked with a cross (+). B - The axes denote the relative voltage differences at given times compared to the moment of detection. Valid detections form a clearly separated cluster. [1]

For the detection, the most commonly used method is thresholding. This means that the mean value of the signal is transformed into 0 µV and a limit is defined around -100 µV. If the voltage value exceeds this limit, it is selected as a spike candidate and further algorithms are applied for verification. For further details, please consult Fig. 1.

Spike sorting is the act of assigning certain waveforms into certain groups, which groups are in correspondence with biological neurons. After detecting the spike, the sorting could happen automatically or manually. In case of manual sorting, the waveforms are transformed into two dimensions, for example with Principal Component Analysis (PCA), then the cluster boundaries are defined by hand. This method has the advantage of the instant applicability of human expertise, but it requires many working hours and in case of complex data, the two-dimensional representation can be a limiting factor. Thus, algorithms, like k-nn, random forest, or even neural networks are used for clustering and classification. [1] [3]

II. MATERIALS

We used artificially generated spike data obtained with 1020 simulated electrodes courtesy of APPERCELL Kft. The simulation spanned one minute for all electrodes resulting in 2.4 million spikes, while the sampling frequency was 20 kHz. Spikes consisted of five different waveforms, which were labeled at the steepest location of the waveform. To make label locations more in line with what is usually considered in the literature and to allow testing of threshold based algorithms, the spike labels were moved to nearest maxima or minima (if both present than to whichever was the largest in absolute value). Spike waveforms were then extracted by taking a 1 ms window around the spike labels. This meant that every sample consisted of 40 time points and had one corresponding label.

III. METHODS

As a first preprocessing step all channels were band-pass filtered with a 2nd order Butterworth filter (150 Hz–2,500 Hz). To perform spike detection we utilized an adaptive amplitude threshold based technique called AdaBandFlt [4] that splits the data stream into 10 ms windows, calculates the root mean squared (RMS) values for each of the initial 100 windows and determines the 25th percentile of RMS distribution. In our dataset the noise level was the same throughout the recording, thus adaptive thresholding was not necessary, but in non-simulated data it can be highly beneficial. After the noise level was estimated for each channel, the threshold was set to be 4 times the noise level. Finally, positive spikes over positive threshold and negative spikes under negative threshold were detected. To account for multiple detections of the same spike waveform a detected peak was tested in 1 ms window, checking if it is the highest peak of either polarity (positive or negative) and if 50% of its amplitude is higher than other peak of the same polarity. If both tests were passed, the detected peak was considered a spike otherwise rejected as noise. The performance of the spike detection algorithm was estimated using accuracy, sensitivity and specificity metrics.

Preliminary exploratory data analysis showed that the class labels are well separable even with very standard approaches such as applying Principal Component Analysis [5] to the waveforms and then using a Random Forest [6] to classify the labels. To this end to accomplish spike sorting we considered additional challenges that emphasized reasonable real-life issues and made the use of deep learning techniques more suitable.

A. *Classification with least data necessary*

To classify the 5 spike labels, the dataset was separated into training, validation and test sets. The training set was used to train the classifiers, the validation set was used to find the optimal hyperparameters, while the test set was used to estimate the generalization capability of the classifier. The data was split into these sets in a stratified fashion after shuffling the samples to avoid additional class imbalance that could result from a random split. The validation and test sets were always the 5% and the 85% of the total data. Multiple test set sizes were considered ranging from 0.1% to 10%. Larger test sets were not considered as above 10% all tested classifiers reached almost maximal classification accuracy. As a deep learning classifier a Convolutional Neural Network [7] (CNN) was considered, while for a baseline comparison Random Forest (RF) and Logistic Regression (LR)[8] was selected. As the spike waveforms were well separable and only contain 40 time points we reasoned that a simple CNN architecture would be able to solve the classification problem and instead of systematically comparing multiple highly different architectures we focused on thorough optimization of the simple architecture. Table 1 shows the layers and the hyperparameter value ranges considered for optimization. The padding for the convolutional layer was set to 'same' and the stride to 1. Besides architectural hyperparameters the batch size (128 or 256) and the model optimizer (adam or nadam) was also chosen by the same high dimensional optimization.

TABLE 1- CNN CLASSIFIER ARCHITECTURE

Hyperparameters	**Layers**		
	1D Convolutional	***Dense1***	***Dense2***
Filter	3-15	-	-
Kernel size	3,4,5	-	-
Kernel regularizer	None, l1, l1_l2	None, l1, l1_l2	None, l1, l1_l2
Bias regularizer	None, l1, l1_l2	None, l1, l1_l2	None, l1, l1_l2
Activation	Swish, sigmoid, relu	Swish, sigmoid, relu	Softmax
Output neurons	-	5	5

a.

To optimize the hyperparameter we used a Tree-structured Parzen Estimation [9] implemented by the Optuna library [10]. The optimization was carried out for 150 iterations, with the first 25 being random grid search iterations. The models trained during the hyperparameter optimization process were trained for a maximum of 500 epochs. The metrics to optimize was the average of the balanced accuracy score (BAS) and the multiclass area under the receiver operating characteristic curve (MAUC) implemented by the scikit-learn library [11]. If this composite classification performance score did not increase at least 0.1% on the validation set over 5 epochs the training was stopped early. By including MAUC besides BAS the model is considered better even if the accuracy does not increase, but the model becomes more confident in its prediction.

The final model trained on the optimized hyperparameters was allowed to train for 1500 batches with a patience time of 15. This was run 5 times to account for differences caused by random weight initialization, then the median of the 5 MAUC and BAS values were taken.

B. *Clustering with additional noise*

To simulate real world scenarios, random noise was added to every waveform. The noise follows Gaussian distribution and its maximal amplitude was chosen as 10% of the largest amplitude in the pure dataset.

After the noise application, we trained an AutoEncoder network on the data. The AutoEncoder consists of two convolutional layers (encoders), two convolutional transpose layers (decoders), and a single convolutional layer that yields the output. It was trained with 30% of our dataset.

After the denoising step, we have experimented with the x-means [12] algorithm. This algorithm is very promising, because it solves (to some degree) the three main pitfalls of the popular k-means, namely:

1. Poor scalability
2. The number of categories (k) has to be manually supplied in advance
3. Finishing prematurely because of local minima

In preparation we used PCA to determine the first 10 principal components. Given that each waveform consists of 40 data points and there are 5 categories, this was a reasonable number.

IV. Results

A. *Classification with least data necessary*

Hyperparameter optimization results of the CNN with using 0.05% of the whole data as a training set (Fig. 1) show that even though test set accuracy is below the best classical classifier (Random Forest) not many hyperparameters made a real difference. The best hyperparameter combination had a batch size of 128, did not utilize any regularization, had swish activation functions, used 13 filters with a kernel size of 3 and optimized using nadam. This produced a validation set accuracy of 92.5%. With the increase of the training set the importance of hyperparameter optimization seemed to be less useful as many different combinations produced very similar classification accuracy results.

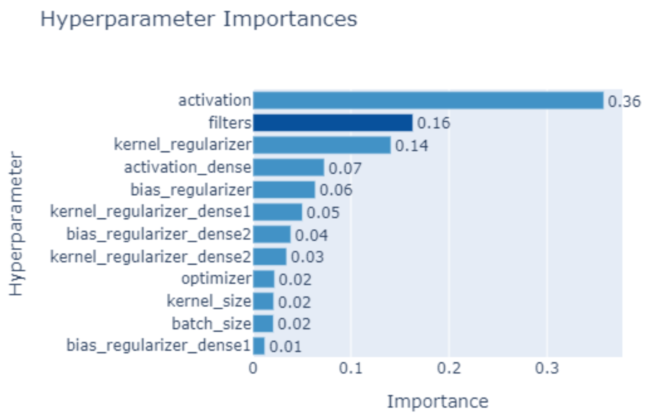

Fig. 2. Hyperparameter importance calculated during the optimization process

Test set classification accuracy (Fig. 2) of the three classifiers show that while using at least 5 or 10 percent of the total data as training set, CNN reached the best classification accuracy together with RF (with 98.9 and 99%, respectively and a 0.9996 MAUC). When using less data RF outperformed both classifiers (96% with 0.05% training set size compared to ~92%). LR achieved the worst classification accuracy overall.

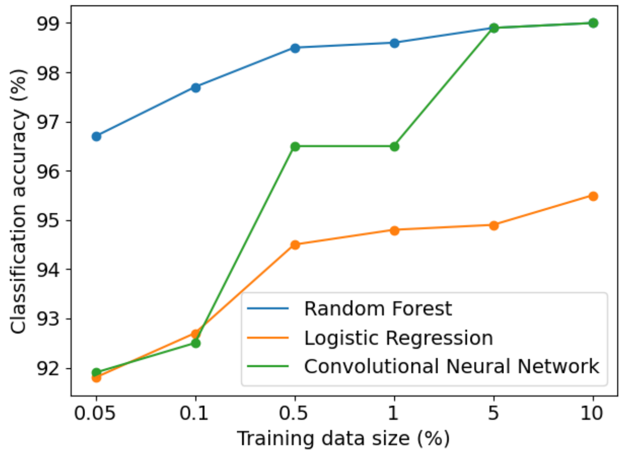

Fig. 3. Test set classification accuracy as a function of training data size for Random Forest, Logistic Regression and the presented CNN model.

A, *Clustering with additional noise*

The AutoEncoder reconstructed the pure signals very well, even in cases where the noised signal was completely unrecognizable for the human eye (Fig. 3). The general form of the spike is almost perfectly restored. On the other hand, the smaller wavelets preceding or succeeding the spike were removed. Given the amount of applied noise this was anticipated.

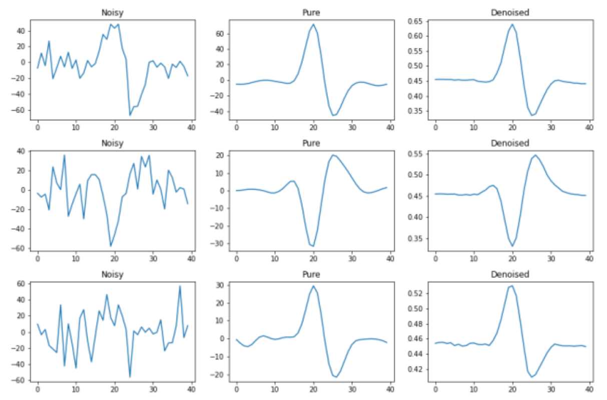

Fig. 4. The removal of artificially added noise with an AutoEncoder network, compared to the pure waveforms.

In the next step, we have run the x-means algorithm on the ten principal components of the dataset. We used the PyClustering [13] implementation of x-means. The results were worse than expected, because the algorithm always returned with the maximum, manually set limit of categories, even after excessive trials with the "tolerance" related to stopping. In the end we manually set the number of maximum categories to five, the number of categories in the dataset. Even with this failure, x-means is a more efficient alternative to k-means.

After this we evaluated the accuracy. The algorithm performed poorly compared to other methods presented in our study, it reached the accuracy of 71.33%. We analyzed the scenario in 3D in Fig 4. Based on this the clusters seemed intermixed, which would explain the poor performance, but we would like to point out that it is not 3, but 10 dimensions, that were available.

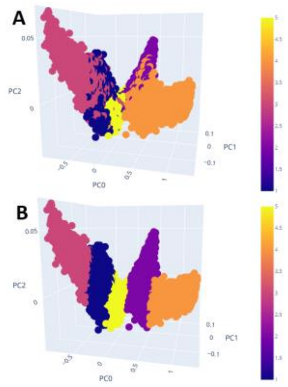

Fig. 5. 3D representation of the first three components yielded by the PCA. Data points on panel A are colored according to the original labels. Colors in panel B show the clusters yielded by the X-means algorithm.

V. Discussion

In this study we tested the performance of deep learning techniques as a part of a pipeline for detecting and sorting spike data. Out of the two approaches assessed in this study the first used a CNN a to classify spikes based on a 40 time point segment and a spike label corresponding to each spike. Spike sorting is often regarded as an unsupervised task, thus the second approach used x-means to cluster the spikes and an autoencoder was considered to denoise a noisy version of the input spike data. In both approaches spike detection was solved with an adaptive threshold-based algorithm.

We have shown that while the CNN classifier using less than 5% of the total as training set could not achieve a better classification accuracy than the RF, with 5% or more the classification accuracies were equal. This tendency corresponds to frequent findings that deep neural networks require more data than classical machine learning models but eventually scale better. Unfortunately, in this case more training data could not further increase classification accuracy thus this better scaling could not be observed. The lower performance of the LR classifier could be caused by the nonlinearity present in the data set as the decision boundary of the LR is linear while RF and CNN produces non-linear boundaries. We successfully used an autoencoder to denoise waveform data. After transforming data into principal components however, this noisy version only achieved a 71.3% accuracy on the test.

The results presented here could be complemented with using real non-simulated spike data, where more complex causal relationships would be present within electrodes. Also, the spike detection and sorting could be achieved within the same neural network by considering each time point as a separate sample and using an architecture with recurrent capability such as a Recurrent or a Long-Short Term Memory Neural Network.

References

[1] D. N. Hill, D. Kleinfeld and S. B. Mehta., „In Observed Brain Dynamics," Oxford Press, pp. 257-270.

[2] P. C. P. P. Petrantonakis, „A simple method to simultaneously detect and identify spikes from raw extracellular recordings," Frontiers in Neuroscience, %1. kötet9, pp. 1-7, 2015.

[3] M. S. Lewicki, „A review of methods for spike sorting: The detection and classification of neural action potentials," Network: Computation in Neural Systems, 1998.

[4] [3] Biffi, E., Ghezzi, D., Pedrocchi, A., & Ferrigno, G. (2010). Development and validation of a spike detection and classification algorithm aimed at implementation on hardware devices. Computational intelligence and neuroscience, 2010.

[5] Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3), 37-52.

[6] Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

[7] Hosmer Jr, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). Applied logistic regression (Vol. 398). John Wiley & Sons.

[8] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. arXiv preprint arXiv:1404.2188.

[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

[10] Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019, July). Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 2623-2631).

[11] Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 24, 2546-2554.

[12] Dan Pelleg, Andrew Moore, „X-means: Extending K-means with Efficient Estimation of the Number of Clusters," Machine Learning, 2002.

[13] PyClustering," [Online]. Available: https://pyclustering.github.io/