Using embedding spaces to visualize and guide refinement of a simple CNN

Peter Rupprecht[1,2], Stephan Gerhard[1], Rainer Friedrich[1,2]

1 Friedrich Miescher Institute for Biomedical Research, Maulbeerstrasse 66, CH-4058 Basel, Switzerland
2 University of Basel, Basel 4003, Switzerland

Basic CNN structure

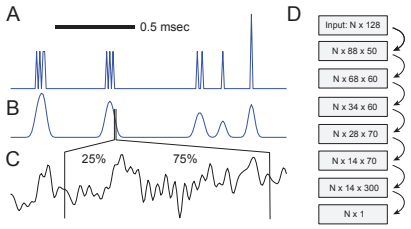
In order to infer a spike probability for time bin j (Fig. 1A), we used the calcium trace located around j, including 25% before and 75% after j, totaling to 128 samples, *i.e.*, 1.28 sec (Fig. 1C). We trained a convolutional neural network to use these 128-wide windows to predict the corresponding spiking probability. To facilitate gradient ascent, we smoothed the discrete spiking ground truth with a Gaussian filter ($\sigma = \sqrt{2}$ sec, Fig. 1B).

We implemented the convolutional neural network in Python using Keras [3] with the Tensorflow [1] backend. The network architecture is shown in Fig. 1. The convolutional filter size, particularly for the first layer, was chosen rather large, since simple CNNs with 3 or 4 convolutional layers with small input filter sizes (3, 5 or 7) performed poorly (data not shown). No zero-padding was used. The numbers of filters were chosen to increase with depth in order to allow for a larger capacity to represent higher-order features. Standard *relu* activation units were used after each convolutional and dense layer, except for the last dense layer, where a linear activation was used to allow the output of continuous spiking probabilities.

All parameters were chosen based on intuition gained through a small exploratory hyperparameter study using diverse 3- and 4-layer CNNs with varying filter sizes, filter numbers and input window sizes. We tried to control overfitting by randomly omitting single neurons from the training data and checking predictive performance of the CNN model for the respective omitted neuron.

Generation of an embedding space based on mutual predictive power

Although the above-described CNN performed well when it came to fitting single datasets of the ground truth, one single model trained on all datasets usually performed not as well for any of the datasets as the same CNN trained on the respective dataset alone. To better understand this, we wanted to quantify how well a model that had been fitted to predict spikes for neuron i can make the same kind of predictions for neuron j. To this end, we fitted a low-capacity CNN (with two locally connected convolutional layers and one dense layer) for each neuron i. The small size of the network together with a high dropout rate during training (50% after each layer) was used to prevent overfitting. This model was then applied to predict spiking probabilities both



Layer	# filters/neurons	Filter size	# parameters
Convolutional	50	41	2.1k
Convolutional	60	21	63.1k
MaxPool	-	2	-
Convolutional	70	7	29.5k
MaxPool	-	2	-
Dense	300	1	21.3k
Dense	1	1	4.2k

Figure 1 – The basic convolutional neural network. **A,B**. The spiking ground truth was smoothed to facilitate gradient descent. **C**. For each time point of the smoothed spiking trace, a 1.28 sec time window of the calcium trace was selected. **D**. The N inputs were transformed using a neural network with three convolutional layers. The numbers in the boxes indicate the output size of the respective layers.

for neuron i and all neurons $j \neq i$. This matrix of 'predictive power' (measured with the Pearson correlation coefficient between prediction and ground truth, identical to the evaluation of the Spikefinder competition), is shown in Fig. 2A. For instance, row 55 shows how well spikes of neuron 55 can be predicted by the networks generated by all other neurons. Column 55, on the other hand, shows how well the model generated by neuron 55 can predict spikes of other neurons. The 5% neurons that were worst at predicting their own spiking were discarded from the following modeling, assuming bad recording quality that is not suited for inclusion into a training dataset.

Normalization over columns, symmetrization of the matrix and averaging over datasets yields a matrix of predictive power, generating a matrix of proximity in prediction-space between datasets (Fig. 2B). A PCA of this matrix results in an embedding space that we limited to two dimensions due to the low number of datasets. Datasets close to each other in the embedding space (e.g. 2 and 4) can predict each other's spikes very well, whereas datasets distant from each other in space (e.g. datasets 4 and 5) fail to do so.

Generation of an embedding space based on the similarity of statistical properties

Using this approach, it is however not yet possible to map a neuron of a new dataset of unknown properties onto the right location of the embedding space above. To solve this problem, we calculated the following statistical properties of the raw calcium time traces (Fig. 2D):

- coefficient of variation, kurtosis, skewness
- autocorrelation of the calcium time trace with its future value in 0.5, 1 and 2 seconds
- generalized Hurst exponents of order 1-5
- the power spectral density at different frequencies between 0.1 and 3.6 Hz

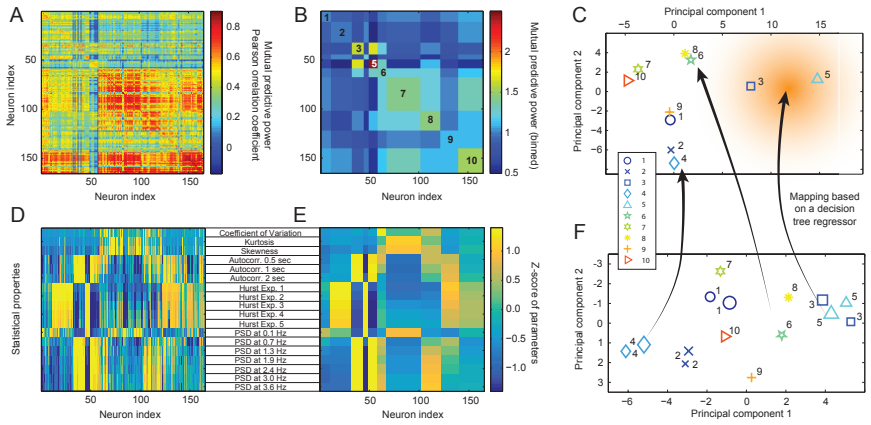


Figure 2 – Using embedding spaces to choose datasets for focused retraining. **A.** Matrix of mutual predictive power, measured using the Pearson correlation coefficient between prediction and ground truth. **B.** Same as (A), but normalized for columns and binned to datasets. **C.** Principal component analysis applied to (B), keeping the first two PCs. **D.** Statistical parameters quantified for single neurons, standardized. **E.** Same as (D), but binned for datasets. **F.** 2D principal component space generated using (E). Symbols with numbers to the right are from training datasets, used to span the PCA space; symbols with numbers to the left are from the test dataset and were projected into the PCA space.

We did not attempt to find a minimal set of predictive properties to reduce computation time here, but used dimensional reduction techniques to automatically extract the relevant independent components. After averaging the standardized values over datasets (Fig. 2E), we used the two first principal components to generate a map of proximity in statistical property space (Fig. 2F). This map was generated using the training datasets (numbers located on the right side of the symbols). Test datasets were mapped into this PCA space (numbers on the left side of the symbols).

To generate a mapping between the locations of the datasets in the two embedding spaces, we fitted a simple regressor (*DecisionTreeRegressor* from the scikit-learn package[4]) to the training datasets (schematic arrows in Fig. 2C,F). We then used this mapping to determine the position of the test datasets in the embedding space of mutual predictive power.

Retraining of the basic model within a local environment of the embedding space

Once the position in the embedding space is known for a dataset, the model that had been trained before on all datasets is retrained, but preferentially with neurons from datasets that lie close to the position in the embedding space. We weighted this preference with a function that decays exponentially over distance in the embedding space, as indicated by the red shading in Fig. 2C. Again, the functional form of the decay and the decay constant have been chosen heuristically without systematic optimization, since our goal was to showcase the power of our embedding space approach rather than find-

ing a global optimum.

Embedding spaces as an intuitive interface to understand model selection

This focused re-training is reminiscent of other approaches to conditional computation based on attention mechanisms. Given that any attention-based network can be replaced by a large enough not explicitly conditional network,[2] it seems clear that a large-enough CNN could make the embedding spaces dispensable without compromising performance, provided that regularization is properly performed. For example, a deep network could be used to learn the mapping from the raw time traces to the position in the predictive embedding space in Fig. 2C, replacing our explicit mapping based on extracted statistical properties. A powerful enough network applied to the original task of mapping calcium traces to spikes would include this mapping implicitly in its internal structure.

However, embedding spaces as a visual and explicit intermediate step for model refinement are more easily accessible for users, allow the use of relatively small convolutional neuronal networks and can highlight similarities and differences between datasets. For example, it is interesting to see that in both embedding spaces, datasets 3 and 5 cluster together, whereas dataset 8, which uses the same calcium indicator (GCaMP6s) in the same brain region (V1), is in proximity of dataset 6 (GCaMP5k in V1). Similarly, the datasets that use OGB-1 as indicator (1,2 and 4) tend to occupy similar regions of the embedding spaces, but datasets 1 and 2 which have both indicator and brain region (V1) in common, do not cluster in an obvious manner.

This indicates that model selection is not only based on the calcium indicator and the brain region, but on hidden parameters, *e.g.*, signal-to-noise of the calcium recording, sampling rate, spike rate, temperature, indicator concentration, or others. To reliably comprise these possible hidden parameters with embedding spaces, it will be necessary to increase the number of datasets in order to support as many possible types of datasets as possible. However, the unknown dimensionality of this hidden parameter space makes it difficult to predict how many datasets would be required.

All of the code together with instructions on a) how to use it to reproduce our results and b) how to apply the networks on unknown calcium imaging data for deconvolution can be found online on Github[5].

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

[3] Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

[4] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[5] Rupprecht, P. and Gerhard, S. (2017). Spikefinder elephant. `https://github.com/PTRRupprecht/Spikefinder-Elephant`.