Spike finder submission

Patrick Mineault

19/05/2017

Introduction

We wish to estimate the number of action potentials emitted by a neuron in a given time bin in a calcium imaging experiment. Specifically, we have access to training data consisting of paired calcium and spike traces, and we wish to predict spikes using only calcium traces on unseen test data. We cast this problem as a supervised learning problem, where:

- x_{ij}: calcium trace of the ith neuron in time bin j
- y_{ij}: number of spikes from the ith neuron at time j
- μ_{ij}: estimate of the number of spikes from the ith neuron at time j

Our goal is then to estimate the parameters of a basis transformation g and an output nonlinearity f such that a loss L is minimized:

$$\begin{aligned} \eta_{ij} &= g(x_{ij}) \\ \mu_{ij} &= f(\eta_{ij}) \\ \text{s.t. } L(\mu_{ij}, y_{ij}) &\text{ is minimized} \end{aligned}$$

While we've cast this problem in the style of a generalized linear model (Theis et al. 2016), we do not limit ourselves to choices of g which lead to convex optimization problems. We examine each component in turn[1].

Basis transformation g

Previous formulations of the spike estimation problem have used linear-quadratic features (Theis et al. 2016). Here we replace this component with a deep convolutional artificial neural network to find a good representation of the input.

[1] The code that fits this model can be downloaded at https://github.com/patrickmineault/spikefinder_submission

The first layer of this network is a standard convolutional layer followed by a rectified linear (ReLU) nonlinearity (Glorot, Bordes, and Bengio 2011), mapping each calcium time series to 32 parallel time series indexed by k:

$$z^0_{ijk} = (x_{ij} * w^0_{jk} + \beta^0_k)^+$$

Hhere $(x)^+ \equiv \max(0, x)$ is the ReLU nonlinearity. We use a large window (33 time points, or 330 ms), batch normalization (Ioffe and Szegedy 2015), as well as a dropout fraction of .3 (Srivastava et al. 2014).

This initial layer is followed by 7 adjustment layers in the style of a residual network (He et al. 2015):

$$z^{l+1}_{ijk} = z^l_{ijk} + \sum_m (z^l_{ijm} * w^{l+1}_{jmk} + \beta^{l+1}_k)^+$$

These adjustment layers use smaller windows (9 time points, or 90 ms). The nonlinear component of each layer is batch normalized. Finally, the output is composed linearly via:

$$\eta_{ij} = \sum_k z^7_{ijk} w^7_k + \beta^7$$

Because this network has 9 layers, we named instances of it after the books of Dante's Divine Comedy, e.g. Inferno for the 9 circles of Hell.

Output nonlinearity

In a preliminary analysis, we regressed windowed calcium traces against spike trains (see also Ringach 2017 for a similar model). We then binned the prediction from the regression and computed the mean number of spikes in each prediction bin to obtain an estimate of the output nonlinearity (Chichilnisky 2001). This consistently revealed output nonlinearities similar to ReLUs, and so we used a ReLU as the final output nonlinearity.

Loss function

Previous research (Theis et al. 2016) has used the correlation coefficient as a measure of prediction accuracy. Unfortunately, the correlation coefficient is difficult to optimize directly. We instead chose a scaled sum of squared error criterion:

$$L(\mu_{ij}, y_{ij}) = \sum_i \min_{\alpha_i} \frac{\sum_j (y_{ij} - \alpha_i \mu_{ij})^2}{\sum_j y_{ij}^2}$$

Here α_i is a set of scalars which are learned alongside the other parameters of the model (w and β). They serve to scale predictions for each cell, in the same way that the correlation coefficient implicitly scales predictions by normalizing against the square root of the norm of the prediction. Thus, this loss function is invariant to scaling.

In the appendix, we show that the loss is equivalent to 1 - ρ^2, where ρ^2 is the square of the cosine similarity between prediction and spike train.

Model fitting

We specified and fit the model with the `tf.contrib.learn` library in TensorFlow (Abadi et al. 2016). We initialized model parameters with the Xavier method and optimized the model with Adam - a scaled version of stochastic gradient descent (Kingma and Ba 2014). We used mini-batches composed of 128 different snippets of calcium traces and parallel spike trains.

Each snippet contained data from 64 consecutive time points (640 ms). We included extra calcium data from $16 + 4 \cdot 7 = 44$ time points before and after each snippet to use `valid` mode convolutions. We used mirror boundary conditions to pad the start and end of each cell's recording. The order of the batches was randomized, so that a mini-batch could contain data from multiple neurons. We fit one large model for all 10 recordings (173 neurons) in this phase.

We monitored goodness-of-fit on a leave-aside validation set to control overfitting by early stopping. Convergence took close to 200,000 iterations.

We created the validation set by leaving aside every 5th snippet of 64 consecutive time points. This method can leak some information between validation folds, a problem we identified late in the spikefinder challenge and did not have time to address. This meant that the submissions were considerably under-regularized. Leaving aside every 5th cell would be a superior validation strategy.

Adaptation and long-range variables

The model described so far uses fixed filters for each recording. Furthermore, it only uses local information ($\approx$ 1 second of data) to estimate spikes from calcium traces.

Previous strategies, on the other hand, have successfully used long-range information to adapt filters. For example, (Pnevmatikakis et al. 2016) uses the auto-correlation of the entire calcium trace to estimate the decay rate of an AR(1)

process, and uses this information to adapt how deconvolution is performed. This means, for example, that the trace is more aggressively deconvolved if the indicator has sluggish dynamics (e.g. GCaMP6s vs. GCaMP6f).

We let the 0th layer filters depend on long-range information:

$$w^0_{ijk} = \sum_{n=1}^{4} \frac{\exp(\kappa_{ijn})}{\sum_n \exp(\kappa_{ijn})} W_{nk}$$

The 0'th layer filters are thus given by a weighted sum - a mixture - of fixed filters, where the weights change from cell to cell and from time to time according to κ_{ijn}. The form of the weighting is inspired by attention networks (Olah and Carter (2016)). There are 4 variants for each 0'th layer filter.

Manually defined long-range features

We manually compiled 32 different long-range features, consisting of summary statistics of the calcium signal in a 5,000 sample window (50s). We computed the 16 following metrics on both the calcium signal and its first derivative:

- Mean, standard deviation, skew, kurtosis
- Quantiles: [.1, .2, .5, .8, .9]
- Autocorrelation at time lags [1, 6, 11, 16, 21, 26, 31]

Let us denote these long-range features by γ_{ijp}. These features γ_{ijp} were processed by two fully connected layers with batch normalization and dropout; the first has 32 hidden features and a ReLU nonlinearity, and the second one has 4 hidden features and no nonlinearity. The output of this second layer gave κ_{ijn}. Thus, the network could combine long-range information in various ways to control the shape of the input filters.

Learned long-range features

We explored an alternative strategy to find long-range features automatically. We used an unsupervised latent variable model to model the calcium traces, and used summaries of these latent variables as long-range features.

We repurposed a mixture density network that was originally designed to model handwriting (Graves 2013; Otoro 2015). The model, a 3-layer recursive neural network with 512 long-short-term memory (LSTM) nodes, is trained to predict the next position of a pen based on its past history. Notably, the network does not simply predict the *most* likely position, but rather the full distribution of next positions via a mixture density. This, combined with the ability to maintain a style over a long range via LSTM nodes, allows the network to generate diverse characters in a variety of self-consistent styles.

We fit the model to the calcium data, feeding in mini-batches of 1024 consecutive samples at a time. We then averaged the values of the 3×512 latent variables over time to obtain one 1536-dimensional latent vector per 1024 consecutive time samples ϕ_{iqr}, where q indexes over time and r over features.

Our next processing step was to reduce this latent vector down to 32 dimensions, to use as long-range features γ_{ijp}. To reduce the dimensionality, we made the following assumption: a good latent variable should vary little across time for a given cell, but it should vary a lot across cells. Of course, several latent variables could carry redundant information, while a good set of latent variables should carry non-redundant information. We used z-scoring to measure variation across cells and singular value decomposition to eliminate the redundancy across latent variables, as follows:

- Let $\psi_{ir} = \frac{\mathbb{E}_q(\phi_{iqr})}{\sigma_q(\phi_{iqr})}$
- Let V_{rp} be the 32 first right singular singular vectors of $\psi_{ir} - \mathbb{E}_i(\psi_{ir})$
- Let $\Gamma_{iqp} = \sum_r V_{rp} \left(\frac{\phi_{iqr}}{\sigma_q(\phi_{iqr})} - \mathbb{E}_i(\psi_{ir}) \right)$
- Tile Γ_{iqp} along the q dimension 1024 times to obtain γ_{ijp}

We then passed these long-range features through a similar 2-layer network as the manually defined long-range features to obtain time-and-cell-varying 0th layer filters.

Refinement

We originally fit one large model for all recordings. We then created refined versions of this model for each of the 10 datasets by a transfer learning process. We took the large model with its learned parameters and ran up to 50,000 extra iterations of gradient descent on just the data from the kth dataset. We used cross-validation to determine the number of extra iterations. We then predicted the response for each dataset using the model refined for it.

Summary of submissions

Name	Description	Test score	Train score
Purgatorio	Learned long-range features + refinement	0.464	0.581
Inferno	Learned long-range features	0.419	0.511
Paradiso	Fixed long-range features	0.410	0.465

Critique

Using *learned* long-range features rather than fixed ones increases training accuracy markedly, but the effect on test set accuracy is more muted. We believe

this is due to the flawed cross-validation used during training, which led us to train the model for more iterations than necessary. While the fixed features have the same dimensionality (32) as the learned features, the learned features are orthogonal. Thus, the model has more opportunity to overfit. This could easily be fixed by using a better validation strategy.

It is surprising that refinement helps to such a large degree, as the model already strives to adapt to the statistics of the signal. Recall that STM (Theis et al. 2016) did not use refinement, but one large model instead. Refinement requires obtaining calibration data, which is not always practical. Perhaps the model does not fully adapt to the data, and more flexible adaptation mechanisms could be considered.

An alternative explanation is that there are idiosyncratic differences in the relationship between calcium and spikes from dataset to dataset. (Ringach 2017) mentions the need for a delay parameter between signal and prediction for one dataset in particular (dataset 5). A recording rig can introduce idiosyncratic delays between spikes and calcium signals – scanning frequency in the optical path and analog filters for the electrophysiology path. One could instead use a loss that is insensitive to delay between prediction and measurement, in the same way that correlation is insensitive to rescaling.

Appendix

Loss and cosine similarity

We'd like to fit a neural network with a scaling invariant loss $L(y, \mu)$ – a loss that remains the same after rescaling the predictor μ.

We would also like this loss to be additive with respect to examples. That is, the metric should have the property that $L(y, \mu) = \sum_{ij} \hat{L}_{ij}(y_{ij}, \mu_{ij})$, where i and j index cell number and time, respectively. Additivity is crucial for stochastic gradient descent. We choose the following loss:

$$L(\mu_{ij}, y_{ij}) = \sum_i \min_{\alpha_i} \frac{\sum_j (y_{ij} - \alpha_i \mu_{ij})^2}{\sum_j y_{ij}^2}$$

Scaling invariance properties holds because of the free α_i. Additivity holds as $\sum_j y_{ij}^2$ is fixed for each cell. We now show that the metric is related to cosine similarity.

First, without loss of generality, let us consider only one cell and drop the i index. The optimal value for α can be expressed in closed form via linear regression:

$$\alpha = \frac{\sum_j y_j \mu_j}{\sum_j \mu_j^2}$$

Thus, we have that:

$$L(\mu_j, y_j) = \frac{\sum_j (y_j - \frac{\sum_j y_j \mu_j}{\sum_j \mu_j^2} \mu_j)^2}{\sum_j y_{ij}^2}$$

After some algebra, we find that:

$$L(\mu_j, y_j) = 1 - \frac{(\sum_j y_j \mu_j)^2}{\sum y_j^2 \sum \mu_j^2}$$

Thus, our similarity function is equivalent to one minus the *square* of the cosine similarity. Cosine similarity is closely related to the correlation coefficient which is used for evaluation.

The formulation of the loss with an explicit α has the computational advantage that we only need local information to compute it; the metric is no longer additive once we replace α with its closed-form optimum. Furthermore, if we absorb the sign of α into μ, this loss and the cosine similarity become related by a monotonic function, and thus they share the same extrema. Thus, this loss formula provides a convenient means of optimizing cosine similarity.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, et al. 2016. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," 14~mar.

Chichilnisky, E J. 2001. "A Simple White Noise Analysis of Neuronal Light Responses." *Network* 12 (2): 199–213.

Glorot, X, A Bordes, and Y Bengio. 2011. "Deep Sparse Rectifier Neural Networks." *Aistats*. jmlr.org.

Graves, Alex. 2013. "Generating Sequences with Recurrent Neural Networks," 4~aug.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Deep Residual Learning for Image Recognition," 10~dec.

Ioffe, Sergey, and Christian Szegedy. 2015. "Batch Normalization: Accelerating

Deep Network Training by Reducing Internal Covariate Shift," 11~feb.

Kingma, Diederik P, and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization," 22~dec.

Olah, Chris, and Shan Carter. 2016. "Attention and Augmented Recurrent Neural Networks." *Distill* 1 (9).

Otoro. 2015. "Handwriting Generation Demo in TensorFlow." http://blog.otoro.net/2015/12/12/handwriting-generation-demo-in-tensorflow/.

Pnevmatikakis, Eftychios A, Daniel Soudry, Yuanjun Gao, Timothy A Machado, Josh Merel, David Pfau, Thomas Reardon, et al. 2016. "Simultaneous Denoising, Deconvolution, and Demixing of Calcium Imaging Data." *Neuron* 89 (2): 285–99.

Ringach, Dario. 2017. "Spikefinder: The Vanilla Algorithm." https://scanbox.org/2017/05/15/spikefinder-the-vanilla-algorithm/.

Srivastava, N, G E Hinton, A Krizhevsky, and others. 2014. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *J. Mach. Learn. Res.* jmlr.org.

Theis, Lucas, Philipp Berens, Emmanouil Froudarakis, Jacob Reimer, Miroslav Román Rosón, Tom Baden, Thomas Euler, Andreas S Tolias, and Matthias Bethge. 2016. "Benchmarking Spike Rate Inference in Population Calcium Imaging." *Neuron* 90 (3): 471–82.