py-oopsi: the python implementation of the fast-oopsi algorithm

Benyuan Liu

Abstract

Fast-oopsi was developed by joshua vogelstein in 2009, which is now widely used to extract neuron spike activities from calcium fluorescence signals. Here, we propose detailed implementation of the fast-oopsi algorithm in python programming language. Some corrections are also made to the original fast-oopsi paper.

Index Terms

python, fast-oopsi, spikes, calcium fluorescence, connectomics

I. FAST-OOPSI, A BRIEF VIEW

Oopsi, from vogelstein [1], [2], is a family of optimal optical spike inference algorithms. Here, we focus on the development of the fast-oopsi, which was originally published in [2]. We will port the MATLAB implementation to python. Sec II, III, IV, V and VI are digests from the original paper by vogelstein [1].

The python implementation, py-oopsi, can be obtained at https://github.com/liubenyuan/py-oopsi.

II. CALCIUM FLUORESCENCE MODEL

Let $F$ be a one-dimensional fluorescence trace. At time $t$, the fluorescence measurement $F_t$ is a linear Gaussian function of the intracellular calcium concentration $[Ca^{2+}]_t$ at that time:

$$F_t = \alpha [Ca^{2+}]_t + \beta + e_t, \quad e_t \sim \mathcal{N}(0, \sigma^2)$$

1Benyuan Liu is with The Department of Biomedical Engineering, Fourth Military Medical University, Xi’an, 710032, China.
E-mail: liubenyuan@gmail.com, lbyoopp@163.com
Manuscript received May 26, 2014.
\( \alpha \) determines the scale of the signal, \( \beta \) absorbs the offset. \( \alpha \) and \( \beta \) may be learned independently per neuron. The noise \( e_t \) is assumed to be i.i.d distributed.

The calcium concentration jumps \( A \ \mu \text{M} \) after each spike and decays back down to baseline \( C_b \ \mu \text{M} \), with time constant \( \tau \),

\[
[\text{Ca}^{2+}]_{t+1} = (1 - \Delta/\tau)[\text{Ca}^{2+}]_t + (\Delta/\tau)C_b + An_t,
\]

where \( \Delta \) is the frame interval. The scale \( A \) and \( \alpha \), baseline \( C_b \) and \( \beta \) are not identifiable, therefore, we may let \( A = 1 \) and \( C_b = 0 \) without loss of generality. \( n_t \) indicates the number of times the neuron spiked in time \( t \), we may also write it as a delta function \( \delta_t \).

Finally, letting \( \gamma = (1 - \Delta/\tau) \), we have

\[
C_t = \gamma C_{t-1} + n_t
\]

and (the filtering model)

\[
C[z] = \frac{1}{1 - \gamma z^{-1}} N[z]
\]

Note that \( C_t \) does not refer to the absolute intracellular concentration, but rather, a relative measure \[2\]. The simulated calcium trace can be generated if we synthetically generate \( n_t \) from a probability distribution. To complete the generative model, we assume spikes are sampled according to a Poisson distribution,

\[
n_t \sim \text{Poisson}(\lambda \Delta)
\]

where \( \lambda \Delta \) is the expected firing rate per bin, \( \Delta \) is included to ensure that the expected firing rate is independent of the frame rate \[2\].

### III. BAYES MODEL

We aim to find the most likely spike trains \( \hat{n} \) given the fluorescence \( F \),

\[
\hat{n} = \arg \max_{n_t \in N_0, \forall t} p(n|F)
\]

Using Bayes’ rule,

\[
p(n|F) = \frac{1}{p(F)} \cdot p(F|n)p(n)
\]

given that \( p(F) \) merely scales the results, we rewrite \(6\) as,

\[
\hat{n} = \arg \max_{n_t \in N_0, \forall t} p(F|n)p(n)
\]
and we already have,

\[ p(F|\mathbf{n}) = \prod p(F_t|C_t), \quad (9) \]
\[ p(\mathbf{n}) = \prod p(n_t), \quad (10) \]

where,

\[ p(F_t|C_t) = \mathcal{N}(\alpha C_t + \beta, \sigma^2), \quad (11) \]
\[ p(n_t) = \text{Poisson}(\lambda \Delta) \quad (12) \]

The Poisson distribution penalize sparsity (a sparse prior).

Finally, we have the cost function,

\[ \hat{n} = \arg \max_{n_t \in \mathbb{N}_0} \sum_{t=1}^T -\frac{1}{2} \frac{(F_t - \alpha C_t - \beta)^2}{\sigma^2} \exp\left\{ -\frac{1}{2} \frac{(F_t - \alpha C_t - \beta)^2}{\sigma^2} \right\} \exp\left\{ -\lambda \Delta \right\} \left(\lambda \Delta\right)^{n_t} \]
\[ \exp\left\{ -\lambda \Delta \right\} \left(\lambda \Delta\right)^{n_t} \rightarrow \left(\lambda \Delta\right) \exp\left\{ -n_t \lambda \Delta \right\} \quad (13) \]
\[ \hat{n} = \arg \max_{n_t \in \mathbb{N}_0} \sum_{t=1}^T -\frac{1}{2\sigma^2} (F_t - \alpha C_t - \beta)^2 - n_t \ln \lambda \Delta - \ln n_t! \quad (14) \]

However, solving for this discretized optimization problem is computational intractable.

### IV. APPROXIMATE BAYES FILTER

We can approximate the Poisson distribution with an exponential distribution of the same mean,

\[ \exp\{ -\lambda \Delta \} \left(\lambda \Delta\right)^{n_t} \rightarrow \left(\lambda \Delta\right) \exp\{ -n_t \lambda \Delta \} \]

and consequently,

\[ \hat{n} = \arg \max_{n_t > 0} \sum_{t=1}^T -\frac{1}{2\sigma^2} (F_t - \alpha C_t - \beta)^2 - n_t \lambda \Delta \quad (16) \]

note that \( n_t \in \mathbb{N}_0 \) has been replaced by \( n_t > 0 \), since exponential distribution can yield any nonnegative number \( [2] \). The exponential approximation imposes a sparsening effect, and also, it makes the optimization problemconcave in \( C \), meaning that any gradient descent algorithm guarantees achieving the global maxima (because there are no local minima).

We may further drop the constraint (nonnegative) by adopting interior point method,

\[ \hat{C} = \arg \max_C \sum_{t=1}^T -\frac{1}{2\sigma^2} (F_t - \alpha C_t - \beta)^2 - (C_t - \gamma C_{t-1}) \lambda \Delta + z \ln(C_t - \gamma C_{t-1}) \quad (18) \]

this cost function is twice differentiable, one can use the Newton-Raphson technique to ascend the surface.
V. Matrix Notation and the Newton-Raphson Solver

To proceed, we have

\[
MC = \begin{bmatrix}
-\lambda & 1 & 0 & \cdots & 0 \\
0 & -\lambda & 1 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & -\lambda & 1 \\
0 & \cdots & 0 & 0 & -\lambda
\end{bmatrix}
\begin{bmatrix}
C_1 \\
C_2 \\
\vdots \\
C_{T-1} \\
C_T
\end{bmatrix}
= \begin{bmatrix}
n_1 \\
n_2 \\
\vdots \\
n_{T-1}
\end{bmatrix}
\]

(19)

\(M\) is a \((T - 1) \times T\) matrix. Now letting \(\mathbf{1}\) be a \((T - 1) \times 1\) column vector, \(\lambda = (\lambda \Delta) \mathbf{1}\), \(\alpha\) and \(\beta\) a \(T\)-dimensional vector, \(\odot\) to indicate element-wise operations, then

\[
\hat{C} = \arg \max_{MC \geq 0} -\frac{1}{2\sigma^2} \|F - \alpha C - \beta\|_2^2 - (MC)^T \lambda + z \ln_\odot (MC)^T \mathbf{1}
\]

(20)

We instead iteratively minimize the cost function \(\mathcal{L}\) (called post in our python implementation) where,

\[
\hat{C}_z = \arg \min_{C} \mathcal{L}, \quad \mathcal{L} = \frac{1}{2\sigma^2} \|F - \alpha C - \beta\|_2^2 + (MC)^T \lambda - z \ln_\odot (MC)^T \mathbf{1}
\]

(21)

\(\mathcal{L}\) is convex, when using Newton-Raphson method to descend a surface, one iteratively computes the gradient \(\mathbf{g} = \nabla \mathcal{L}\) (first derivative) and Hessian \(\mathbf{H} = \nabla^2 \mathcal{L}\) (second derivative) of the argument to be optimized. Then, \(C = C - s \mathbf{d}\), where \(s\) is the step size and \(\mathbf{d}\) is the step direction by solving \(\mathbf{H} \mathbf{d} = \mathbf{g}\). The gradient and Hessian, with respect to \(C\), are

\[
\mathbf{g} = -\frac{\alpha}{\sigma^2} (F - \alpha C - \beta) + M^T \lambda - z M^T (MC)_{\odot}^{-1}
\]

(22)

\[
\mathbf{H} = \frac{\alpha^2}{\sigma^2} \mathbf{I} + z M^T (MC)_{\odot}^{-2} M
\]

(23)

\(s\) is found via backtracking linesearches. \(M\) is bidiagonal, so \(\mathbf{H}\) is tridiagonal, \(\mathbf{d} = \mathbf{H}^{-1} \mathbf{g}\) can be efficiently implemented in matlab by assuming \(\mathbf{H}\) is a sparse matrix. In python, we may use sparse linsolvers (linsolve.spsolve) to efficiently find \(\mathbf{d}\). Once \(\hat{C}\) is obtained, it is a simple linear transform to obtain \(\hat{n}\), via \(\hat{n} = MC\). We will normalize \(\mathbf{n}\) by \(\mathbf{n} = \mathbf{n}/\max(\mathbf{n})\) after convergence.

VI. Parameters Initialize and Update

The parameters \(\theta = \{\alpha, \beta, \sigma, \gamma, \lambda\}\) are unknown. We may use pseudo expectation-maximization method, (1), initialize the parameters, (2) recursively computes \(\hat{n}\) and updating \(\theta\) given the new \(\hat{n}\) until the convergence is met.

\(^1\)contrary to \(^2\), but alike fast-oopsi.m, we choose \(M\) as a sparse \(T \times T\) matrix, and \(\mathbf{1}\) as \(T \times 1\) column vector. Therefore we have \(n_0 = C_0\), we will correct \(n_0 = \epsilon\) after convergence.
The scale of \( F \) relative to \( n \) is arbitrary, therefore, \( F \) is firstly detrended, and then linearly mapped between 0 and 1.

\[
F = \text{detrend}(F), \quad F = (F - F_{\text{min}})/(F_{\text{max}} - F_{\text{min}}),
\]

(24)

Next, because spiking is sparse in many experimental settings, \( F \) tends to be around baseline, \( \beta \) is set to the median of \( F \). We use median absolute deviation (MAD) and correction factor \( K \), as a robust normal scale estimator of \( F \) where \( K = 1.4826 \). Previous works showed that the results \( \hat{n} \) and \( \hat{C}_z \) are robust to minor variations in the time constant, we let \( \gamma = 1 - \Delta \). Finally, \( \lambda \) is set to 1Hz, which is between baseline and evoked spike rate for data of interest.

\[
\alpha = 1, \quad \beta = \text{median}(F), \quad \sigma = \text{MAD}(F) \cdot K = \text{median}(|F - \beta|) \cdot K, \quad K = 1.4826
\]

(27)

\[
\gamma = 1 - \Delta/(1\text{sec}), \quad \lambda = 1\text{Hz}
\]

(29)

Then, given \( \hat{C} \) and \( \hat{n} \), we may (approximately) update \( \theta \) by,

\[
\hat{\theta} \approx \arg \max_\theta p(F, \hat{C} | \theta) = \arg \max_\theta \ln p(F|\hat{C}; \{\alpha, \beta, \sigma\}) + \ln p(\hat{n} | \lambda)
\]

(30)

where,

\[
\hat{\lambda} = \arg \max_{\lambda > 0} \sum_{t=1}^T [\ln(\lambda \Delta) + \hat{n}_t \lambda \Delta]
\]

(31)

\[
\{\hat{\alpha}, \hat{\beta}, \hat{\sigma}\} = \arg \max_{\alpha, \beta, \sigma > 0} \sum_{t=1}^T \left[ -\frac{1}{2} \ln(2\pi \sigma^2) - \frac{1}{2} \left( \frac{F_t - \alpha C_t - \beta}{\sigma} \right)^2 \right]
\]

(32)

We have (by taking the derivatives and letting them equal zero),

\[
\hat{\lambda} = \frac{T}{\Delta \sum_t n_t},
\]

(33)

\[
\hat{\alpha} = 1,
\]

(34)

\[
\hat{\beta} = \frac{\sum_t (F_t - C_t)}{T},
\]

(35)

\[
\hat{\sigma}^2 = \frac{\sum_t (F_t - C_t - \beta)^2}{T} = \frac{\|F - C - \beta\|^2}{T}
\]

(36)

where \( \hat{\lambda} \) is the inverse of the inferred firing rate, \( \hat{\alpha} \) can be set to 1.0 because the scale of \( C \) is arbitrary, \( \hat{\beta} \) is the mean bias, \( \hat{\sigma} \) is the root-mean-square of the residual error.

\[\text{corrections to [2]: 1), add detrend to } F, \quad 2), K = 1.4826 \text{ and it is multiplied (not divided by) } \text{MAD}(F).\]
VII. IMPLEMENTATION OF OOPS

Matlab implementation is available, here we focus on the python migrant, and correct some typos in [2] as needed. The python code itself explains all, see [IV] [V] and [VI] for detailed documentary. Pseudo code can be found in Algo 1. Algo 2 describe the subroutine MAP, Algo 3 describe the subroutine update.

Algorithm 1 Pseudo code (python) for fast-oopsi
1: Initialize parameters $P: F = \text{detrend}(F), F = (F - \text{min}(F)) / (\text{max}(F) - \text{min}(F)), \alpha = 1.0, \beta = \text{median}(F), \lambda = 1.0, \gamma = 1 - \Delta, \sigma = \text{MAD}(F) \cdot 1.4826, T = \text{len}(F)$
2: one-shot Newton-Raphson
3: for $i$ in $1 \cdots \text{iterMax}$ do
4: update parameters $P = \text{update}(n, C, F, P)$, see Algo 2
5: iterative through $n, C, L = \text{MAP}(F, P)$, see Algo 3
6: let $L^{(k)} = \{L_1, \cdots, L_k\}$
7: if $|L_i - L_{i-1}| < \text{ltol}$ or any$(|L^{(i)} - L_i|) < \text{gtol}$ then
8: break
9: end if
10: end for

VIII. WIENER FILTER (LINEAR REGRESSION, SIMPLE CONVEX OPTIMIZATION)

In the wiener filter, we approximate the Poisson distribution with a Gaussian distribution,

$$p(nt) \sim \mathcal{N}(\lambda\Delta, \lambda\Delta)$$ (37)

then, the MAP estimator yields,

$$\hat{n} = \arg\max_{n_t} \sum_{t=1}^{T} \left[ -\frac{1}{2\sigma^2}(F_t - \alpha C_t - \beta)^2 - \frac{1}{2\lambda\Delta}(n_t - \lambda\Delta)^2 \right]$$ (38)

and its matrix notation,

$$\hat{C} = \arg\max_{C} -\frac{1}{2\sigma^2}||F - \alpha C - \beta1||_2^2 - \frac{1}{2\lambda\Delta}||MC - \lambda\Delta1||_2^2$$ (39)

which is quadratic, concave in $C$.

Finally, we aim to optimize (minimize, quadratic, convex optimization),

$$\hat{C} = \arg\min_{C} L, \quad L = \frac{1}{2\sigma^2}||F - \alpha C - \beta1||_2^2 + \frac{1}{2\lambda\Delta}||MC - \lambda\Delta1||_2^2$$ (40)
Algorithm 2 Pseudo code of subroutine MAP
1: Initialize $n = 0.011$
2: Initialize $C(z) = 1/(1 - \gamma)N(z)$
3: Initialize $\lambda = \lambda \Delta 1$
4: for $z = 1.0, z > 1e-13, z = z/10$ do
5: calculate $L_z$
6: while $s > 1e-3$ or $||d|| > 5e-2$ do
7: Calculate $g, H$ and $d = \text{spsolve}(H, g)$
8: Find $s : h = -n/(Md), s = \min(0.99s[s > 0], 1.0)$
9: Initialize $L_s = L_z + 1$
10: while $L_s > L_z + 1e-7$ do
11: $C = C + sd$
12: $n = MC$
13: update $L_s$
14: decrease $s = s/5.0$
15: if $s < 1e-20$ then
16: break
17: end if
18: end while
19: end while
20: end for

Algorithm 3 Pseudo code of subroutine update
1: $\alpha = 1.0$
2: $\beta = \sum_i(F_i - C_i)/T$
3: $\sigma^2 = ||F - \alpha C - \beta||^2_2/T$
4: $\lambda = T/(\Delta \sum_i n_i)$
where $\mathcal{L}$ is convex in $C$. Using Newton-Raphson update, we find $C = C - d$, $Hd = g$ and $g = \nabla \mathcal{L}$, $H = \nabla^2 \mathcal{L}$. The gradient $g$ and Hessian $H$ are:

$$ g = -\frac{\alpha}{\sigma^2}(F - \alpha C - \beta 1) + \frac{1}{\lambda \Delta} [M^T(MC) + \lambda \Delta M^T 1] $$  \hspace{1cm} (41)

$$ H = \frac{\alpha^2}{\sigma^2} I + \frac{1}{\lambda \Delta} M^T M $$  \hspace{1cm} (42)

In the python implementation, we let $\alpha = 1.0$ and $\beta = 0.0$. Pseudo code can be found in Algo 4.

**Algorithm 4**  Pseudo code (python) for wiener filter

1. Initialize $F = (F - \text{mean}(F))/\max(|F|)$, $\sigma = 0.1 ||F||_2$
2. Calculate $\mathcal{L}_0$
3. for $i$ in $1 \cdots \text{iterMax}$ do
   4. Calculate $g$, $H$ and $d = \text{spsolve}(H, g)$
   5. Calculate $C = C - d$
   6. Calculate $\mathcal{L}_i$
   7. if $\mathcal{L}_i < \mathcal{L}_{i-1} + \text{gtol}$ then
      8. $n = N$
      9. $\sigma = \sqrt{||F - C||_2^2} / T$
   10. end if
11. end for
12. $n = n / \max(n)$

**IX. Simulation Results**

We generated synthetic calcium traces with $T = 2000$, $\Delta = 20\text{ms}$, $\lambda = 0.1$, $\tau = 1.5$. Randomized noise were added with 0.2 standard deviation. Py-oopsi and wiener filter are used to reconstruct the spikes from calcium fluorescence, where only $\Delta$ is known a prior. The results are shown in Figure 1.

**REFERENCES**

[1] J. T. Vogelstein, *OOPSI: A family of optimal optical spike inference algorithms for inferring neural connectivity from population calcium imaging*. THE JOHNS HOPKINS UNIVERSITY, 2010.

[2] J. Vogelstein, A. Packer, T. Machado, T. Sippy, B. Babadi, R. Yuste, and L. Paninski, “Fast nonnegative deconvolution for spike train inference from population calcium imaging.” *Journal of neurophysiology*, vol. 104, no. 6, pp. 3691–3704, 2010.
Fig. 1. Reconstruct spikes from calcium fluorescence. (a) The synthetic calcium trace. (b), (c), (d) are reconstructed spikes by py-oopsi, wiener filter and discretized binning, respectively.