CodingLab3_HaraldKugler_4258298_LeanderZimmermann_4165446

May 9, 2023

Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2023

Student name: Harald Kugler, Leander Zimmermann

1 Coding Lab 3

- **Data**: Download the data file `nds_cl_3_*.csv` from ILIAS and save it in a subfolder `../data/`.
- **Dependencies**: You don't have to use the exact versions of all the dependencies in this notebook, as long as they are new enough. But if you run "Run All" in Jupyter and the boilerplate code breaks, you probably need to upgrade them.

Two-photon imaging is widely used to study computations in populations of neurons. In this exercise sheet we will study properties of different indicators and work on methods to infer spikes from calcium traces. All data is provided at a sampling rate of 100 Hz. For analysis, please resample it to 25 Hz using `scipy.signal.decimate`.

```
[ ]: from __future__ import annotations
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     import numpy as np
     from scipy import signal
     from scipy.io import loadmat


     %matplotlib inline

     %load_ext jupyter_black

     %load_ext watermark
     %watermark --time --date --timezone --updated --python --iversions --watermark␣
       ↪-p sklearn
```

```
The jupyter_black extension is already loaded. To reload it, use:
  %reload_ext jupyter_black
The watermark extension is already loaded. To reload it, use:
  %reload_ext watermark
Last updated: 2023-05-09 16:05:55CEST

Python implementation: CPython
Python version       : 3.10.0
IPython version      : 8.12.0

sklearn: 1.2.2

matplotlib: 3.7.1
pandas    : 2.0.0
numpy     : 1.24.2
seaborn   : 0.12.2
scipy     : 1.10.1

Watermark: 2.3.1
```

[ ]:
```
plt.style.use("../matplotlib_style.txt")
```

1.1 Load data

[ ]:
```
# ogb dataset from Theis et al. 2016 Neuron
ogb_calcium = pd.read_csv("../data/nds_cl_3_ogb_calcium.csv", header=0)
ogb_spikes = pd.read_csv("../data/nds_cl_3_ogb_spikes.csv", header=0)

# gcamp dataset from Chen et al. 2013 Nature
gcamp_calcium = pd.read_csv("../data/nds_cl_3_gcamp2_calcium.csv", header=0)
gcamp_spikes = pd.read_csv("../data/nds_cl_3_gcamp2_spikes.csv", header=0)
```

[ ]:
```
ogb_calcium.shape, ogb_spikes.shape, gcamp_calcium.shape, gcamp_spikes.shape
```

[ ]:
```
((71986, 11), (71986, 11), (23973, 37), (23973, 37))
```

[ ]:
```
ogb_calcium.head()
```

[ ]:
```
          0         1         2         3         4         5         6  \
0 -0.234928  0.689186  2.578560  0.636770 -0.015140  0.819771  2.442105
1 -0.182441  0.869132  2.868601  0.596164  0.022464  0.831411  2.930201
2 -0.049257  0.949273  2.784989  0.711875  0.040028  0.877191  3.065872
3 -0.032876  0.848802  2.430735  0.940943  0.038616  0.975941  3.051014
4 -0.125313  0.665088  2.064211  1.168007  0.014035  1.156689  2.967063

          7         8         9        10
```

```
0  0.391310  0.223523  1.557675  0.625052
1  0.490213  0.116095  1.628054  0.638718
2  0.610873  0.154437  1.479382  0.552809
3  0.714453  0.335288  1.302766  0.420998
4  0.737085  0.550485  1.234470  0.367595
```

[ ]:
```
# set all nans to zero - otherwise signal.decimates outputs only nans for some␣
 ↪cells
ogb_calcium = ogb_calcium.fillna(0)
ogb_spikes = ogb_spikes.fillna(0)
gcamp_calcium = gcamp_calcium.fillna(0)
gcamp_spikes = gcamp_spikes.fillna(0)

# Resample to 25 Hz using signal.decimate from scipy

ogb_calcium_25hz = signal.decimate(ogb_calcium, 4, axis=0)
ogb_spikes_25hz = signal.decimate(ogb_spikes, 4, axis=0).clip(
    0, None
)  # clip negative values to 0
gcamp_calcium_25hz = signal.decimate(gcamp_calcium, 4, axis=0)
gcamp_spikes_25hz = signal.decimate(gcamp_spikes, 4, axis=0).clip(
    0, None
)  # clip negative values to 0

fs = 25  # Hz
dt = 1 / fs  # seconds
```

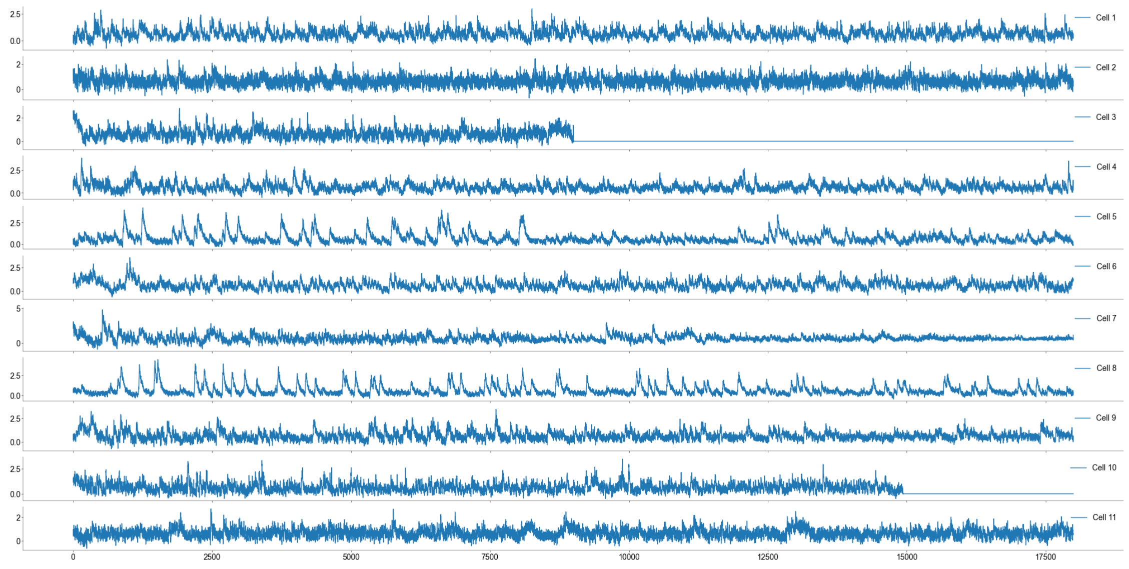
1.2 Task 1: Visualization of calcium and spike recordings

We start again by plotting the raw data - calcium and spike traces in this case. One dataset has been recorded using the synthetic calcium indicator OGB-1 at population imaging zoom (~100 cells in a field of view) and the other one using the genetically encoded indicator GCamp6f zooming in on individual cells. Plot the traces of an example cell from each dataset to show how spikes and calcium signals are related. A good example cell for the OGB-dataset is cell 5. For the CGamp-dataset a good example is cell 6. Zoom in on a small segment of tens of seconds and offset the traces such that a valid comparison is possible.

Grading: 2 pts

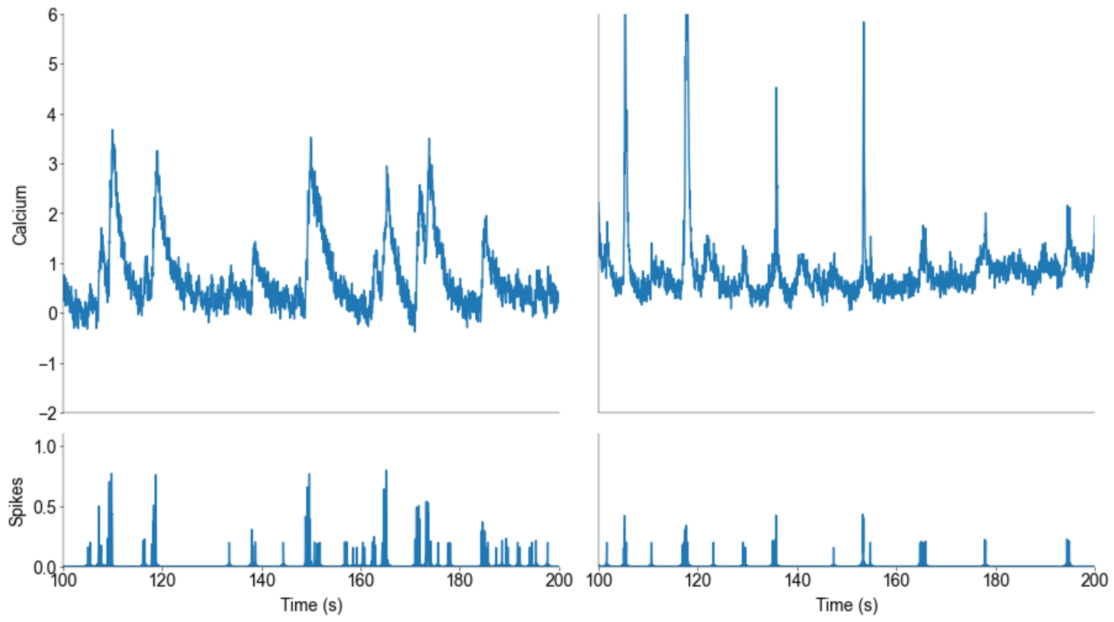
[ ]:
```
# Plot the whole ogb calcium dataset

fig, ax = plt.subplots(ogb_calcium_25hz.shape[1], 1, figsize=(20, 10),␣
 ↪sharex=True)

for i in range(ogb_calcium_25hz.shape[1]):
    ax[i].plot(ogb_calcium_25hz[:, i], label=f"Cell {i+1}")
    ax[i].legend()
```

```
plt.show()
```



[ ]:
```
# ----------------------------
# Plot raw calcium data (1 pt)
# ----------------------------
def plot_calcium(ax, calcuim_data, cell_i):
    cell_data = calcuim_data[:, cell_i]
    t = np.arange(cell_data.shape[0]) * dt
    ax.plot(t, cell_data, label=f"{cell_i}")


# --------------------------
# Plot raw spike data (1 pt)
# --------------------------
def plot_spikes(ax, spike_data, cell_i):
    cell_data = spike_data[:, cell_i]
    t = np.arange(cell_data.shape[0]) * dt
    ax.plot(t, cell_data, label=f"{cell_i}")
    ax.set_xlabel("Time (s)")


fig, axs = plt.subplots(
    2, 2, figsize=(9, 5), height_ratios=[3, 1], layout="constrained"
)

OGB_CELL = 4
GCAMP_CELL = 5
```

```
WINDOW = 100  # in seconds
START = 100  # in seconds

for i, ax in enumerate(axs.T):
    for a in ax:
        if i == 1:
            a.set_yticks([])
        a.set_xlim(START, START + WINDOW)
    ax[0].set_ylim(-2, 6)
    ax[1].set_ylim(0.0, 1.1)
    ax[0].set_xticks([])

axs[0, 0].set_ylabel("Calcium")
axs[1, 0].set_ylabel("Spikes")
# plot raw ogb data
plot_calcium(axs[0, 0], ogb_calcium_25hz, OGB_CELL)
plot_spikes(axs[1, 0], ogb_spikes_25hz, OGB_CELL)

# plot raw gcamp data
plot_calcium(axs[0, 1], gcamp_calcium_25hz, GCAMP_CELL)
plot_spikes(axs[1, 1], gcamp_spikes_25hz, GCAMP_CELL)
```



We see that the underlying spikes very clearly.

1.3 Task 2: Simple deconvolution

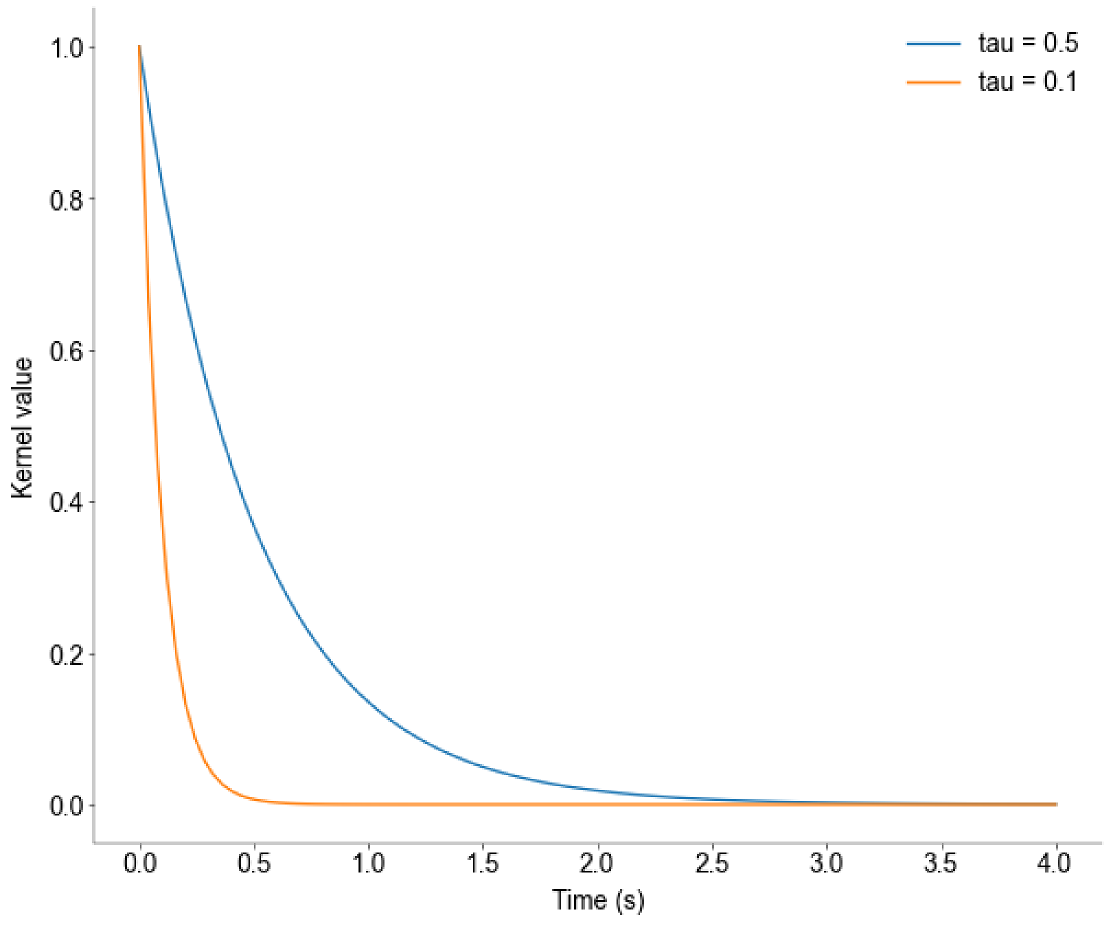
It is clear from the above plots that the calcium events happen in relationship to the spikes. As a first simple algorithm implement a deconvolution approach like presented in the lecture in the function `deconv_ca`. Assume an exponential kernel where the decay constant depends on the indicator ($\tau_{OGB} = 0.5s$, $\tau_{GCaMP} = 0.1s$). As we know that there can be no negative rates, apply a heavyside function to the output. Plot the kernel as well as an example cell with true and deconvolved spike rates. Scale the signals such as to facilitate comparisons. You can use functions from `scipy` for this.

Grading: 3 pts

[ ]:
```python
def deconv_ca(ca, tau, dt):
    """Compute the deconvolution of the calcium signal.

    Parameters
    ----------

    ca: np.array, (n_points,)
        Calcium trace

    tau: float
        decay constant of conv kernel

    dt: float
        sampling interval.

    Return
    ------

    sp_hat: np.array
    """

    # insert your code here

    # --------------------------------------------
    # apply devonvolution to calcium signal (1 pt)
    # --------------------------------------------

    kernel_len = 5  # in seconds (choose something that is much longer than the␣
 ↪kernel has support)
    x = np.linspace(0, kernel_len, kernel_len * int(1 / dt))
    kernel = np.exp(-x / tau)
    sp_hat, _ = signal.deconvolve(ca, kernel)
    return sp_hat.clip(0, None)
```

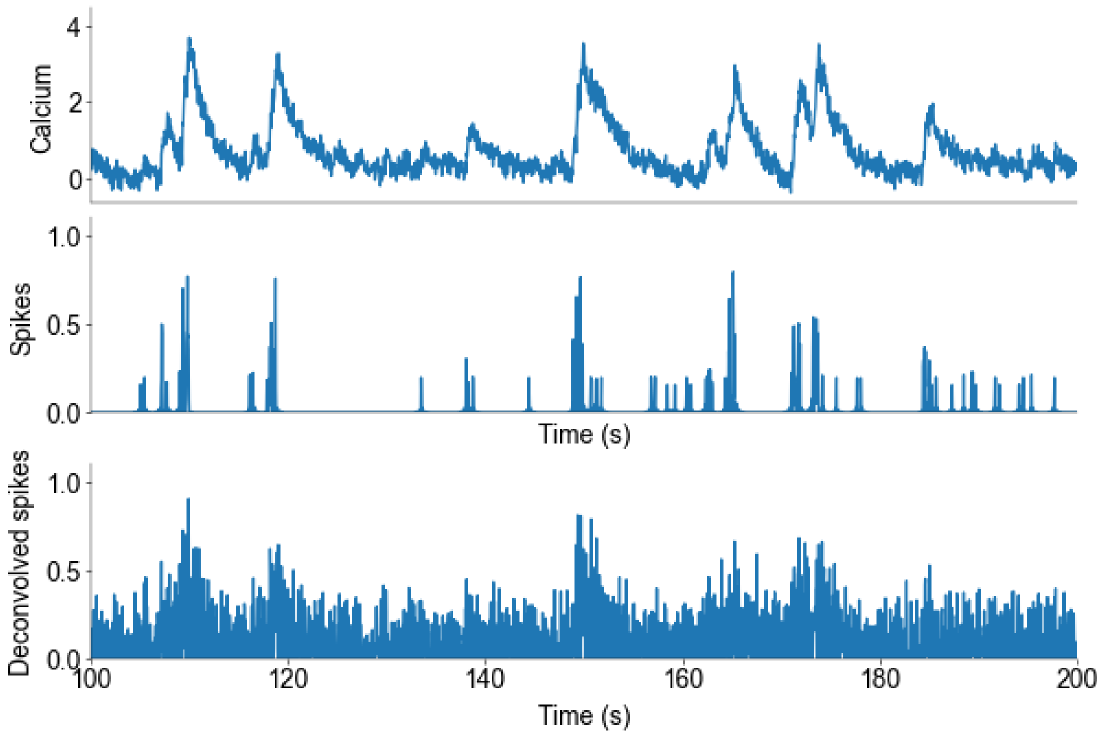
[ ]:
```python
fig, ax = plt.subplots(figsize=(6, 5), layout="constrained")
```

```
# -------------------------
# Plot the 2 kernels (1 pt)
# -------------------------

WINDOW = 4
x = np.linspace(0, WINDOW, fs * WINDOW)
kernel_1 = np.exp(-x / 0.5)
kernel_2 = np.exp(-x / 0.1)
ax.plot(x, kernel_1, label="tau = 0.5")
ax.plot(x, kernel_2, label="tau = 0.1")
ax.set_xlabel("Time (s)")
ax.set_ylabel("Kernel value")
ax.legend()

plt.show()
```



[ ]:
```
# ---------------------------------------------------------------------
# Compare true and deconvolved spikes rates for the OGB or GCamP Cell (1 pt)
```

```python
# -------------------------------------------------------------------------

fig, axs = plt.subplots(
    3, 1, figsize=(6, 4), height_ratios=[1, 1, 1], gridspec_kw=dict(hspace=0)
)

# OGB Cell
OGB_CELL = 4
GCAMP_CELL = 5
WINDOW = 100  # in seconds
START = 100  # in seconds

# apply convolution to calcium signal
sp_hat = deconv_ca(ogb_calcium_25hz[:, OGB_CELL], 0.5, dt)

# plot raw ogb data
axs[0].set_xlim(START, START + WINDOW)
plot_calcium(axs[0], ogb_calcium_25hz, OGB_CELL)
axs[0].set_xticks([])
axs[0].set_ylabel("Calcium")

axs[1].set_xlim(START, START + WINDOW)
plot_spikes(axs[1], ogb_spikes_25hz, OGB_CELL)
axs[1].set_xticks([])
axs[1].set_ylim(0.0, 1.1)
axs[1].set_ylabel("Spikes")

axs[2].set_xlim(START, START + WINDOW)
axs[2].set_xlabel("Time (s)")
axs[2].set_ylabel("Deconvolved spikes")
axs[2].set_ylim(0.0, 1.1)
axs[2].plot(np.arange(sp_hat.shape[0]) * dt, sp_hat)

plt.show()
```



1.4 Task 3: Run more complex algorithm

As reviewed in the lecture, a number of more complex algorithms for inferring spikes from calcium traces have been developed. Run an implemented algorithm on the data and plot the result. There is a choice of algorithms available, for example:

- Vogelstein: oopsi
- Theis: c2s
- Friedrich: OASIS

Grading: 2 pts

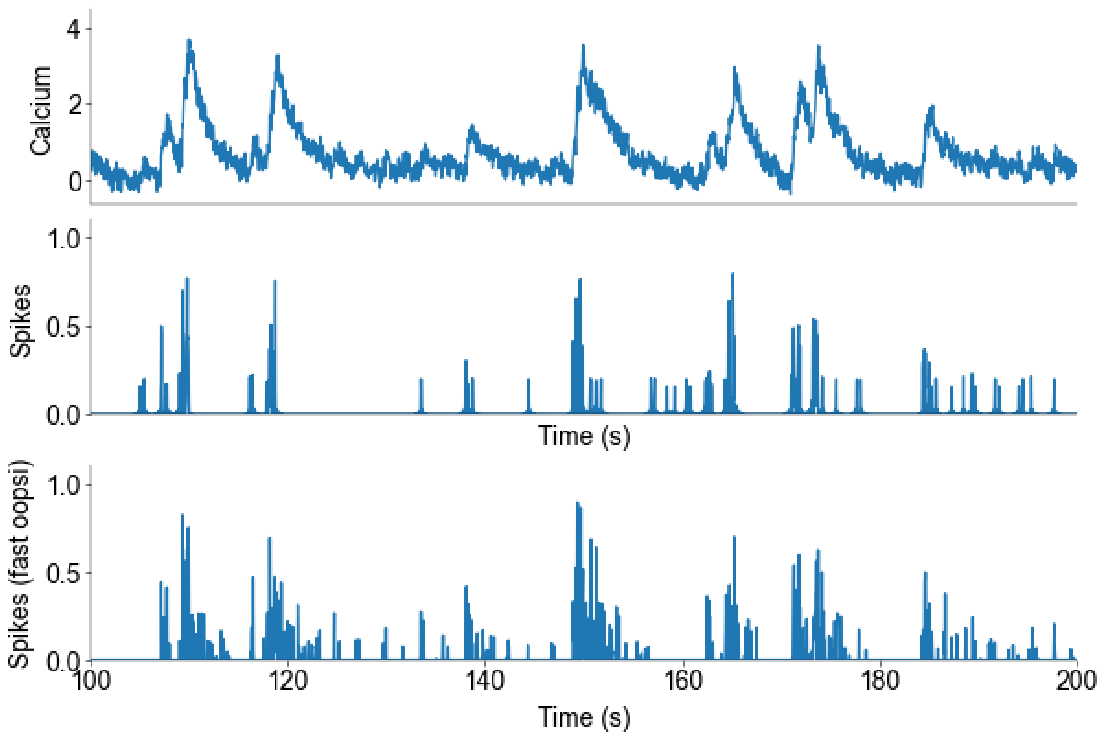
[ ]:
```
# run this cell to download the oopsi.py file if you haven't already mannually␣
  ↪downloaded it
# and put it in the same folder as this notebook
#!wget https://raw.githubusercontent.com/liubenyuan/py-oopsi/master/oopsi.py
```

[ ]:
```
import oopsi
```

[ ]:
```
# ---------------------------------------------------------------
# Apply one of the advanced algorithms on the OGB Cell (0.5 pts)
# ---------------------------------------------------------------


OGB_CELL = 4
```

```
res, _ = oopsi.fast(ogb_calcium_25hz[:, OGB_CELL], dt=dt, iter_max=10)
```

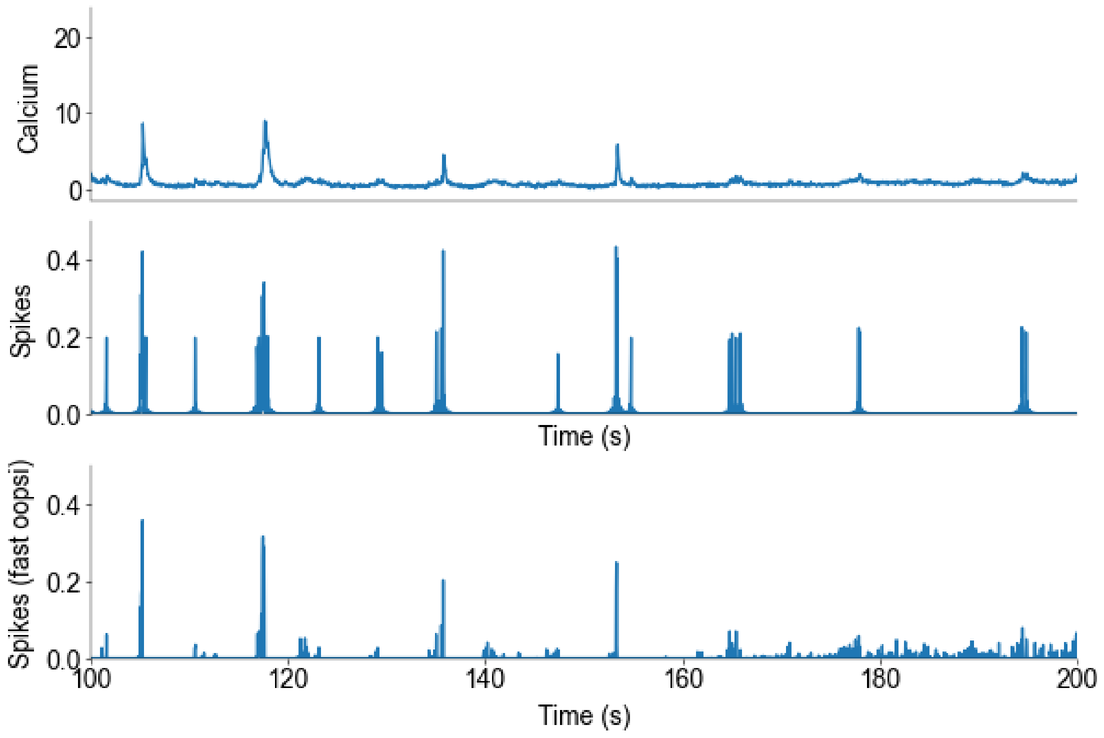
[ ]:
```
# ------------------------------------------
# Plot the results for the OGB Cell (0.5 pts)
# ------------------------------------------

fig, axs = plt.subplots(
    3, 1, figsize=(6, 4), height_ratios=[1, 1, 1], gridspec_kw=dict(hspace=0)
)

# OGB Cell
WINDOW = 100  # in seconds
START = 100  # in seconds

# plot raw ogb data
axs[0].set_xlim(START, START + WINDOW)
plot_calcium(axs[0], ogb_calcium_25hz, OGB_CELL)
axs[0].set_xticks([])
axs[0].set_ylabel("Calcium")

axs[1].set_xlim(START, START + WINDOW)
plot_spikes(axs[1], ogb_spikes_25hz, OGB_CELL)
axs[1].set_xticks([])
axs[1].set_ylim(0.0, 1.1)
axs[1].set_ylabel("Spikes")

axs[2].set_xlim(START, START + WINDOW)
axs[2].set_xlabel("Time (s)")
axs[2].set_ylabel("Spikes (fast oopsi)")
axs[2].set_ylim(0.0, 1.1)
axs[2].plot(np.arange(res.shape[0]) * dt, res)

plt.show()
```



```
[ ]: # -----------------------------------------------------------------
     # Apply one of the advanced algorithms on the GCamP Cell (0.5 pts)
     # -----------------------------------------------------------------

     GCAMP_CELL = 5
     res, _ = oopsi.fast(gcamp_calcium_25hz[:, GCAMP_CELL], dt=dt, iter_max=10)
```

```
[ ]: # ---------------------------------------------
     # Plot the results for the GCamp Cell (0.5 pts)
     # ---------------------------------------------

     fig, axs = plt.subplots(
         3, 1, figsize=(6, 4), height_ratios=[1, 1, 1], gridspec_kw=dict(hspace=0)
     )

     WINDOW = 100  # in seconds
     START = 100  # in seconds

     # GCamP Cell
     axs[0].set_xlim(START, START + WINDOW)
     plot_calcium(axs[0], gcamp_calcium_25hz, GCAMP_CELL)
     axs[0].set_xticks([])
     axs[0].set_ylabel("Calcium")
```

```
axs[1].set_xlim(START, START + WINDOW)
plot_spikes(axs[1], gcamp_spikes_25hz, GCAMP_CELL)
axs[1].set_xticks([])
axs[1].set_ylim(0.0, 0.5)
axs[1].set_ylabel("Spikes")

axs[2].set_xlim(START, START + WINDOW)
axs[2].set_xlabel("Time (s)")
axs[2].set_ylabel("Spikes (fast oopsi)")
axs[2].set_ylim(0.0, 0.5)
axs[2].plot(np.arange(res.shape[0]) * dt, res)

plt.show()
```



1.5 Task 4: Evaluation of algorithms

To formally evaluate the algorithms on the two datasets run the deconvolution algorithm and the more complex one on all cells and compute the correlation between true and inferred spike trains. `DataFrames` from the `pandas` package are a useful tool for aggregating data and later plotting it. Create a dataframe with columns

- algorithm
- correlation

- indicator

and enter each cell. Plot the results using `stripplot` and/or `boxplot` in the `seaborn` package.

Grading: 3 pts

Evaluate on OGB data

```
[ ]: ogb_calcium_25hz[:, 0].shape[0] - deconv_ca(ogb_calcium_25hz[:, 0], 0.5, dt).
      ↪shape[0]
```

```
[ ]: 124
```

```
[ ]: # -------------------------------------------------
     # Create dataframe for OGB Cell as described (1 pt)
     # -------------------------------------------------
     indicator = "OGB-1"
     algo = ["deconv_ca", "fast_oopsi"]

     # calcuate correlations for each algorithm
     c = np.zeros(len(algo) * ogb_spikes_25hz.shape[1])
     for i, alg in enumerate(algo):
         for j in range(ogb_spikes_25hz.shape[1]):
             if alg == "deconv_ca":
                 res = deconv_ca(ogb_calcium_25hz[:, j], 0.5, dt)
             elif alg == "fast_oopsi":
                 res, _ = oopsi.fast(ogb_calcium_25hz[:, j], dt=dt, iter_max=10)
             c[i * ogb_spikes_25hz.shape[1] + j] = np.corrcoef(
                 res, ogb_spikes_25hz[: len(res), j]
             )[0, 1]
```

Create OGB dataframe

```
[ ]: df_ogb = pd.DataFrame(
         {
             "algorithm": np.repeat(algo, ogb_calcium_25hz.shape[1]),
             "correlation": c,
             "indicator": np.repeat([indicator], ogb_calcium_25hz.shape[1] * 2),
         }
     )
     df_ogb.head()
```

```
[ ]:    algorithm  correlation indicator
     0  deconv_ca     0.385882     OGB-1
     1  deconv_ca     0.148623     OGB-1
     2  deconv_ca     0.181313     OGB-1
     3  deconv_ca     0.261496     OGB-1
     4  deconv_ca     0.294508     OGB-1
```

Evaluate on GCamp data

```
[ ]: # --------------------------------------------------
     # Create dataframe for GCamP Cell as described (1 pt)
     # ---------------------------------------------------#

     indicator = "GCamp6f"
     algo = ["deconv_ca", "fast_oopsi"]

     # calcuate correlations for each algorithm
     c = np.zeros(len(algo) * gcamp_spikes_25hz.shape[1])
     for i, alg in enumerate(algo):
         for j in range(gcamp_spikes_25hz.shape[1]):
             if alg == "deconv_ca":
                 res = deconv_ca(gcamp_calcium_25hz[:, j], 0.5, dt)
             elif alg == "fast_oopsi":
                 res, _ = oopsi.fast(gcamp_calcium_25hz[:, j], dt=dt, iter_max=10)
             c[i * gcamp_spikes_25hz.shape[1] + j] = np.corrcoef(
                 res, gcamp_spikes_25hz[: len(res), j]
             )[0, 1]
```

Create GCamp dataframe

```
[ ]: df_gcamp = pd.DataFrame(
         {
             "algorithm": np.repeat(algo, gcamp_spikes_25hz.shape[1]),
             "correlation": c,
             "indicator": np.repeat(indicator, gcamp_calcium_25hz.shape[1] * 2),
         }
     )
     df_gcamp.head()
```

```
[ ]:    algorithm  correlation indicator
     0  deconv_ca     0.629954   GCamp6f
     1  deconv_ca     0.562792   GCamp6f
     2  deconv_ca     0.579627   GCamp6f
     3  deconv_ca     0.541455   GCamp6f
     4  deconv_ca     0.239371   GCamp6f
```

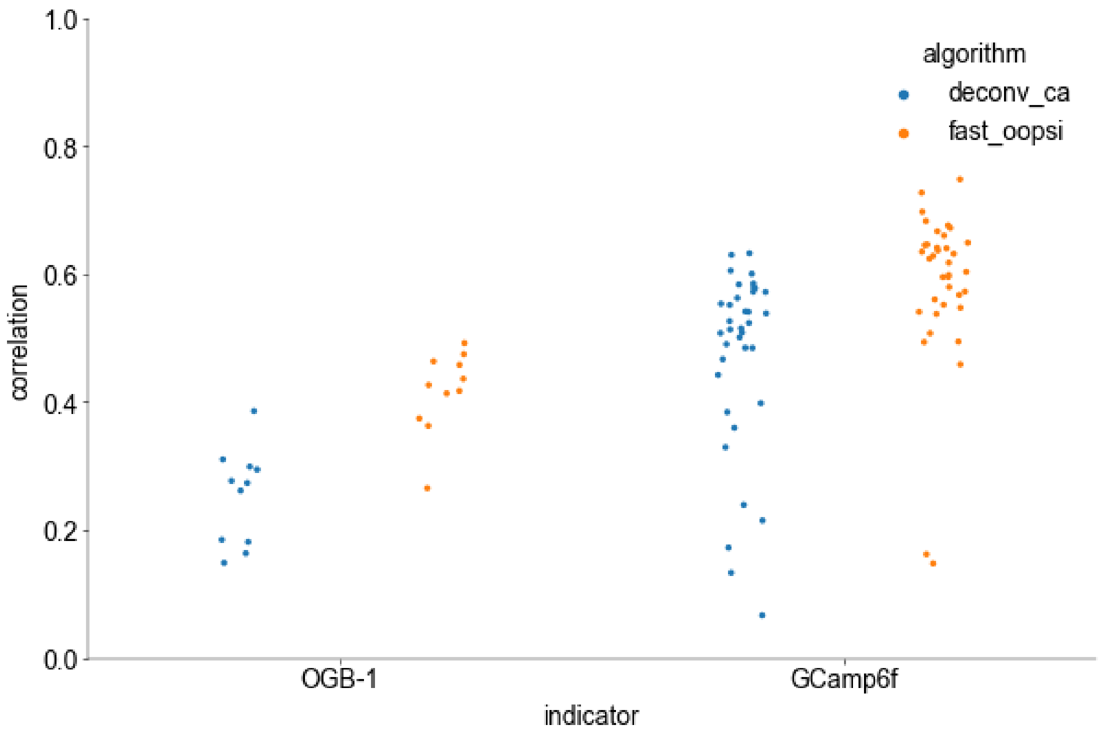
Combine both dataframes and plot

```
[ ]: # combine dataframes

     df_combined = pd.concat([df_ogb, df_gcamp])
     df_combined
```

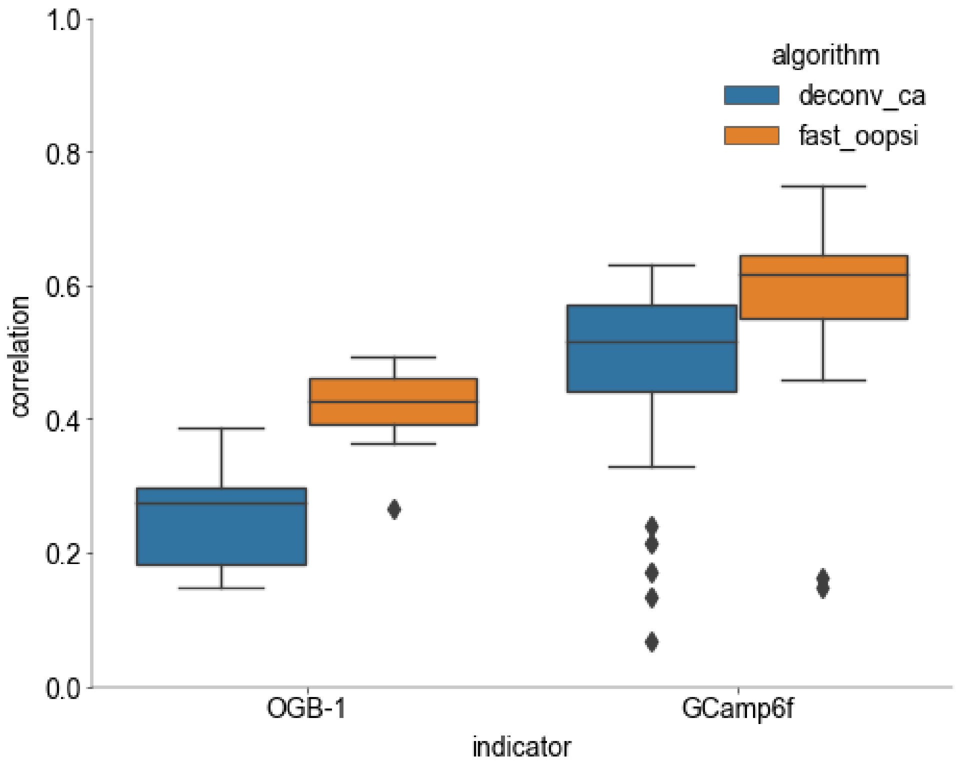
```
[ ]:      algorithm  correlation indicator
     0    deconv_ca     0.385882     OGB-1
     1    deconv_ca     0.148623     OGB-1
     2    deconv_ca     0.181313     OGB-1
```

```
3    deconv_ca     0.261496     OGB-1
4    deconv_ca     0.294508     OGB-1
..         …           …        …
69  fast_oopsi     0.636871   GCamp6f
70  fast_oopsi     0.631902   GCamp6f
71  fast_oopsi     0.635174   GCamp6f
72  fast_oopsi     0.598627   GCamp6f
73  fast_oopsi     0.640519   GCamp6f

[96 rows x 3 columns]
```

[ ]:
```
# ---------------------------------------------------------------------------
# Create Strip/Boxplot for both cells and algorithms Cell as described (1 pt)
# hint: you can seperate the algorithms by color
# ---------------------------------------------------------------------------

fig, axs = plt.subplots(figsize=(6, 4))
sns.stripplot(
    data=df_combined,
    x="indicator",
    y="correlation",
    hue="algorithm",
    ax=axs,
    dodge=True,
)
axs.set_ylim(0.0, 1.0)
plt.show()
```



```
[ ]: # create boxplot for the same data
     fig, axs = plt.subplots(figsize=(5, 4))
     sns.boxplot(
         data=df_combined,
         x="indicator",
         y="correlation",
         hue="algorithm",
         ax=axs,
         dodge=True,
     )
     axs.set_ylim(0.0, 1.0)
     plt.show()
```


algorithm
deconv_ca
fast_oopsi
1.0
0.8
0.6
0.4
0.2
0.0
correlation
OGB-1
GCamp6f
indicator

```
[ ]:
```